DISSERTATION

submitted to the
Combined Faculties for the Natural Sciences and for Mathematics

of the
Ruperto-Carola-University of Heidelberg, Germany

for the degree of
Doctor of Natural Sciences

presented by
Dipl.-Phys. Martin Albrecht Trefzer
born in Lörrach, Germany

Date of oral examination: 13.12.2006

Evolution of Transistor Circuits

Referees: Prof. Dr. Karlheinz Meier
Prof. Dr. Fred A. Hamprecht

Evolution von Transistor Schaltungen

Der Entwurf von analogen Schaltungen ist ein Bereich der Elektronikentwicklung, der dem Entwickler ein hohes Maß an Wissen und Kreativität beim Lösen von Problemen abverlangt. Bis heute gibt es nur rudimentäre analytische Lösungen um die Bauteile von Schaltungen zu dimensionieren. Motiviert durch diese Herausforderungen, konzentriert sich diese Arbeit auf die automatische Synthese analoger Schaltungen mit Hilfe von Evolutionären Algorithmen. Als analoges Substrat wird ein *field programmable transistor array (FPTA)* benutzt, das ein Feld von konfigurierbaren Transistoren zur Verfügung stellt. Der Einsatz von echter Hardware bietet zwei Vorteile: erstens können entstehende Schaltungen schneller getestet werden als mit einem Simulator und zweitens funktionieren die gefundenen Schaltungen garantiert auf einem echten Chip. Softwareseitig eignen sich Evolutionäre Algorithmen besonders gut für die Synthese analoger Schaltungen, da sie keinerlei Vorwissen über das Optimierungsproblem benötigen. In dieser Arbeit werden neue genetische Operatoren entwickelt, die das Verständnis von auf dem FPTA evolutionierten Schaltungen erleichtern und außerdem Lösungen finden sollen, die auch außerhalb des Substrates funktionieren. Dies ist mit der Hoffnung verbunden, möglicherweise neue und ungewöhnliche Schaltungsprinzipien zu entdecken. Weiterhin wird ein mehrzieliger Optimierungsalgorithmus implementiert und verfeinert, um die Vielzahl von Variablen berücksichtigen zu können, die für die gleichzeitige Optimierung von Topologie und Bauteiledimensionierung notwendig sind. Die vorgeschlagenen genetischen Operatoren, sowie die mehrzielige Optimierung werden für die Evolution von logischen Gattern, Komparatoren, Oszillatoren und Operationsverstärkern eingesetzt. Der Ressourcenverbrauch der durch Evolution gefundenen Schaltungen wird damit vermindert und es ist möglich in einigen Fällen einen übersichtlichen Schaltplan zu erstellen. Ein modulares System für die Evolution von Schaltungen auf konfigurierbaren Substraten wurde entwickelt. Es wird gezeigt, dass mit diesem System FPTA-Architekturen modelliert und direkt für Evolutionsexperimente verwendet werden können.

Evolution of Transistor Circuits

Analog circuit design is a discipline of electronic design, that demands a lot of knowledge and experience as well as a considerable amount of creativity in solving diverse problems from the designer and there are to date only rudimentary analytical solutions for parameterizing circuit topologies. Motivated by the latter challenges, this thesis focuses on the analog design automation for FPTA architectures by means of evolutionary algorithms (EAs). The advantages of using real hardware for circuit evolution are the significantly faster evaluation of candidate solutions compared to a simulator and the fact that found solutions are guaranteed to work on a real-world substrate. On the software side EAs are particularly well suited for analog circuit synthesis since they do not require prior knowledge of the optimization problem. New genetic operators are developed within this thesis aiming to facilitate the understanding of evolved FPTA circuits and to find solutions that can be transfered to other technologies. A great hope is thereby to possibly discover unusual, new design principles. Furthermore, a multi-objective algorithm is implemented and refined, in order to allow for taking the numerous variables into account, that are required for optimizing the topology and the dimensioning of transistor circuits. The proposed genetic operators and the multi-objective approach are successfully applied to the evolution of logic gates, comparators, oscillators and operational amplifiers. It is achieved to reduce the resource consumption of evolved circuits and in some cases it is possible to generate clear schematics of good solutions. A modular framework for the evolution of circuits on configurable substrates has been developed, which is used to perform the experiments and is further demonstrated to be useful for modeling FPTA architectures and subsequently using them in evolution experiments.

Contents

Introduction

Analog circuits are the basis for any electronic device and have thereby tremendously influenced our lives during the last decades. Electronic equipment like satellites, mobile phones or digital cameras would simply not exist, if the technology for integrating hundreds of millions of transistors, which represent the building blocks of analog circuits, was not available. The success story of the transistor began with its invention in 1947 by William Shockley, John Bardeen and Walter Brattain and with its successful integration into a receiver circuit in 1958 [98]. Only the advent of the field effect transistor in 1971 has been the final breakthrough, due to this technology rendered the first microprocessor possible, namely the Intel 4004, which was built of 2.300 transistors. According to Moore's Law [61], the transistor count has exponentially grown during the last 35 years and this lead, for instance, to the Itanium 2 processor [38], which consists of the impressive number of more than half a billion transistors.

An inevitable consequence of the rapidly increasing density of transistors is the development of concepts and tools to effectively and efficiently organize such a great number of components. As a consequence, large circuit designs are usually divided into smaller subcircuits, which can be independently developed and facilitate the description of entire systems. Furthermore, an important low level solution to the problem is to introduce logic circuits as an abstraction layer over the transistors, which, on the one hand, greatly facilitates the design of digital circuits and, on the other hand, makes it possible to describe digital circuits independent from the technology. However, in practice, the underlying analog circuits define the specifications of the digital layer, e.g. speed, input voltage range and noise. According to the differentiation between analog and digital design, both approaches aim for different applications. Analog circuits are needed for any interaction with the real world, e.g. controlling valves or a motor, creating sounds or measuring physical quantities like light intensity, whereas almost every signal processing task can be more easily carried out with digital hardware.

As the need for fast, customizeable signal and data processing units further increased, field programmable gate arrays (FPGAs) emerged, which are meanwhile widely used in various applications. FPGAs are providing a huge number of various building blocks for logic circuits on a single substrate, which can be interconnected with a complex configurable routing scheme, in order to build large digital circuits. The design software, that comes along with those chips, indeed unfolds their real power as it is possible to describe a logic circuit with a hardware description language (HDL) on an abstract level and subsequently the FPGA can be configured in a way that it becomes a physical representation of this circuit. Additionally, it is possible to reconfigure the device an indefinite number of times, which offers the possibility to develop highly customizeable hardware. Once

a circuit is described in a HDL, it will immediately benefit from improvements of new technologies, if an appropriate new FPGA is available.

Driven by the intention of further automating the circuit design process, yet aiming for reducing it to merely a specification of the desired task alongside with a suitable heuristic, which is able to develop a solution circuit, field programmable transistor arrays (FPTAs) entered the discipline of evolvable hardware in the 1990s. The algorithms, that are applied for problem solving in the field of evolutionary computation (EC), derive their operation principles from natural evolution and are widely used as model-free heuristics for solving complex optimization tasks. Thus, they are denoted as EAs. In the case of evolvable electronics, the candidate solutions will be either represented by a configuration bit string, if real configurable hardware is targeted (*intrinsic evolution*), or by a netlist, if a simulator is used (*extrinsic evolution*). Moreover, the model-free nature of EAs is an advantageous property, since this makes it possible to apply them to a great variety of problems without including prior knowledge about possible solutions into the algorithm. As Higuchi stated in 1992 [34], the combination of configurable hardware and evolutionary algorithms suggests itself, as EAs rely on a great number of evaluations of candidate solutions, which can be performed at high speed with a hardware evolution system. Although a real chip is inherently limited by its fixed constraints, it offers considerable advantages: first, evaluation speed is significantly faster than in simulation and, second, it ensures an intrinsic realism of the found solutions, which are bound to work at least on their particular evolution platform. It shall be thereby mentioned that the field of evolutionary hardware is not restricted to electronics, but also exhibits examples of the successful evolution of e.g. antennas [7, 57, 58], that are put on a space mission, wing shapes for supersonic aircrafts [63, 66] and orbit trajectories with minimal coverage blackout for telecommunication satellites [99].

Eventually, the initial spark, that set especially the community of evolutionary electronics on fire, has been ignited by Thompson in 1999 [84–86], who not only achieved to successfully evolve a tone discriminator on a Xilinx FPGA, but happened to find a circuit with astonishing properties. For one thing the circuit operated without a clock signal, which is quite unusual from a human designer's point of view, and for another thing evolution seemed to exploit parasitic effects of the substrate in order to get the circuit working. Thompson found unconnected subcircuits which were nevertheless essential for proper operation. Inspired and encouraged by this work, researchers developed various ideas for using configurable architectures as evolution platforms for electronic circuits, e.g. certainly all sorts of field programmable analog arrays (FPAAs) (Anadigm, Cypress, Zetex and [52, 53, 76, 79, 107]) and even a liquid crystal display in the sense of a programmable matter array [30, 31, 60]. Additionally, EAs offered new possibilities for researching fault tolerance, build-in self test (BIST), self-recovery and adaptability [10, 29, 43, 44, 88, 104] to the field of evolutionary electronics.

The aforementioned efforts and results within this research field motivate the implementation of FPTAs as the analog counterparts to the already well-elaborated FPGAs and finally lead to the conception and design of the Heidelberg FPTA in 2001 [50–52], which represents the analog configurable evolution substrate for the experiments in this thesis. It is manufactured in a $0.6\,\mu$m complementary metal oxide semiconductor (CMOS) process and provides 256 configurable transistor cells, which can be variously interconnected and by this means it is possible to realize a great variety of transistor circuits

on the Heidelberg FPTA. Aside from application specific FPAAs, where the behavior can be tweaked, but the topology is fixed and an FPTA from the NASA Jet Propulsion Laboratory (JPL) [76,79,107], which already consists of high-level building blocks (configurable amplifier architecture), the Heidelberg FPTA is the only fine-grained substrate—in the sense of configurable on the transistor level—in the world. The philosophy behind preferring one of those architectures differs: the use of comparably complex cells aims to quickly find robust solutions for problems that fit the predefined structures, but those solutions are always bound to the constraints of the topology. Contrary to that, the single transistor cells provide a higher degree of freedom to the evolving circuits and therefore offer the possibility to discover new or unusual circuit topologies. However, it has to be kept in mind that finer-grained substrates suffer from increasing parasitic effects although this is not necessarily a shortcoming, as Thompson's results suggest.

Analog circuit design is a discipline of electronic design, that demands a lot of knowledge and experience as well as a considerable amount of creativity in solving diverse problems from the designer. Motivated by the latter challenges, new genetic operators are developed within this thesis aiming to facilitate the understanding of evolved transistor circuits and to make it possible to transfer them to other technologies, that is to verify them in simulation. Thereby, a great hope is to possibly discover any kind of new or innovative design principle. Furthermore, a multi-objective approach is adapted and refined, in order to allow for taking the numerous variables into account, that are required for optimizing the topology and the dimensioning of CMOS transistor circuits. It is shown that the proposed genetic operators and the multi-objective approach can be successfully applied to the evolution of logic gates, comparators, oscillators and operational amplifiers. It is achieved to significantly reduce the resource consumption of evolved circuits and in some cases it is indeed possible to generate a clear schematic of good solutions. Within this thesis, a modular framework for the evolution of circuits on configurable substrates has been developed, which is used to perform the various experiments and is further demonstrated to be useful for modeling FPTA architectures and subsequently assessing their evolvability.

The thesis is organized as follows: part I introduces the principles of CMOS design and modeling (chapter 1) and provides the ideas behind evolutionary algorithms while emphasizing the topics, that are relevant to this thesis (chapter 2). All components of the setup are described in part II: first, the architecture and the properties of the Heidelberg FPTA are shown in chapter 3. Second, chapter 4 provides an introduction to the simulation environment, which is used for off-chip verification of the found solutions. Third, the modular evolution software framework is presented in chapter 5 alongside with a description of how to customize it. Finally, the conducted experiments and the results are discussed in part III. Following the course of research, the new genetic operators are introduced in chapter 6 and their performance in evolving logic gates and comparators is compared to a straight forward implementation of the EA. Owing to the increasing complexity of the tackled problems, i.e. the evolution of oscillators and operational amplifiers, a multi-objective algorithm has been developed and is discussed in chapter 7. To conclude, the algorithm is applied to extrinsic evolution of comparators with a simplified simulation model of the real hardware in chapter 8. Based on these results and the experience gained during this thesis a proposal for an improved FPTA architecture is presented.

Part I

Foundations

Chapter 1

CMOS Analog Circuit Design

This chapter introduces the basic concepts of CMOS analog circuit design, which are relevant to this thesis. The focus is set on metal oxide semiconductor (MOS) transistors and transmission gates, since those are the basic building blocks of the field programmable transistor array (FPTA), namely the Heidelberg FPTA, which is extensively used for the experiments in this thesis and is described in chapter 3. Characteristics and parasitic effects of the latter devices are discussed, due to mainly the transmission gates (switches) significantly influence the circuits that are realized on the FPTA. Moreover, it is shown how MOS transistors can be used—to a certain extent—as capacitances and resistors, hence, can replace passive components. The reason for this is that no such components are directly provided by the configurable transistor array. An additional aim of this thesis is to verify FPTA circuits in simulation. Thus, CMOS transistor models, which are used by simulators for calculating the behavior of analog circuits are described. Thereby, in analog circuit design, since the aim is to reliably fabricate a fully operational chip, it is crucial to use a suitable and sufficiently accurate model of the respective process for simulation. Lastly, an overview of the CMOS design flow is given, since this work aims at automating several design steps by means of evolutionary algorithms.

During the last two decades, CMOS processes gained importance in the field of very large scale of integration (VLSI) technology, due to some great advantages: on the one hand, the structures in CMOS processes are steadily shrinking and therefore the density of transistors increases. Current technology makes it possible to integrate millions of transistors on one single die (chip). On the other hand, field effect transistors feature a zero current gate, thus, can be successfully applied as well for highly intergrated digital designs. The latter feature has become crucial for implementing systems on a chip, which consist of both, analog and digital circuits on the same substrate. Such systems are denoted as *mixed-signal systems*. The information presented in this chapter is mainly based on the foundations from [4, 81, 82].

1.1 Physical Representation of the CMOS Transistor

p-channel or n-channel devices

MOS transistors are four terminal electrical devices, which are either used as voltage controlled resistors or current sources in analog circuits and can be realized either as n-channel or p-channel devices in CMOS technology. A lightly doped p^- substrate is used in CMOS technology, thus, the n-channel transistor can be directly formed with two heavily doped n^+ regions, as can be seen from figure 1.1. In the case of the p-channel transistor, which is realized with two heavily doped p^+ regions, an additional lightly

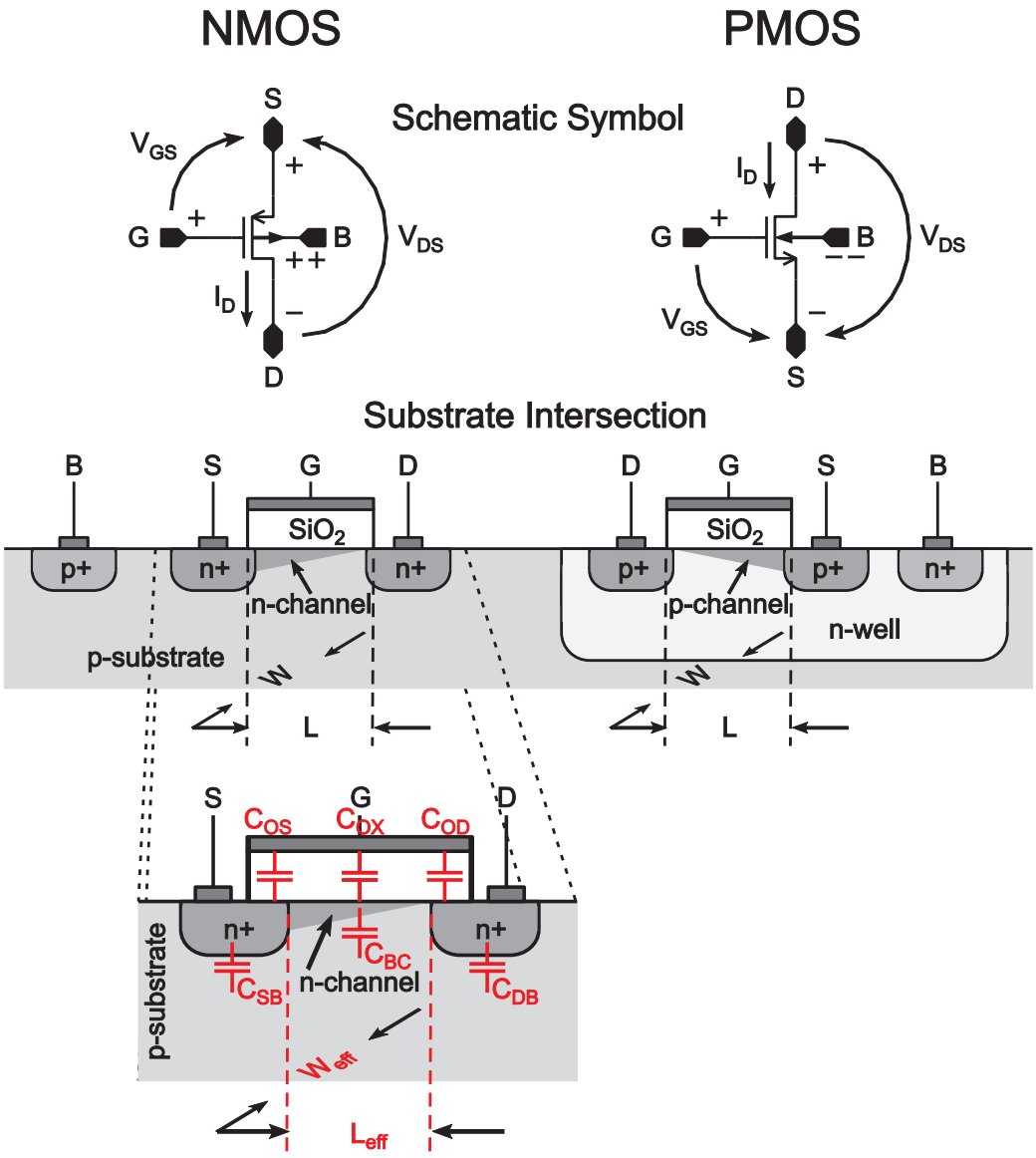


Figure 1.1: *Top: the symbolic view of a PMOS and an NMOS transistor. Middle: a substrate cross section through a PMOS and an NMOS transistor. Bottom: the parasitic device capacitances are shown, using the NMOS transistor as an example. Parts of the illustration are taken from [49].*

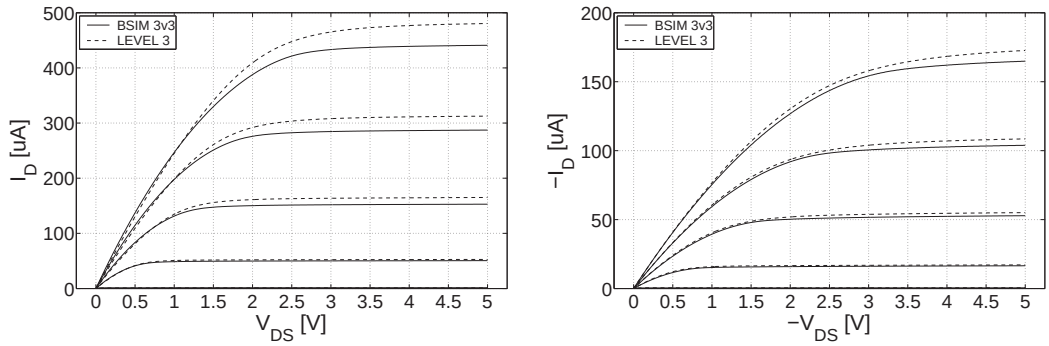


Figure 1.2: *Example output characteristics of a PMOS (left) and an NMOS transistor (right) with $W = L = 2\,\mu$m are graphed for $V_{GS} = 1,2,3,4,5\,$V. Thereby, the source voltage of the NMOS was gnd and the source of the PMOS was tied to vdd. Furthermore, the $i_D - v_{DS}$ characteristics for two simulation models, namely the SPICE level 3 model and the industrial standard BSIM3v3, are compared.*

doped n^- well is necessary. The heavily doped regions are named source (S) and drain (D), while the substrate resp. the n-well is named bulk (B). At the surface, a gate electrode (G), which is separated from the substrate by a dielectric material (silicon dioxide) lies between source and drain.

field effect transistor with isolated gate

Considering the n-channel transistor with all four terminals connected to ground, at equilibrium, the source and drain are separated by the depletion regions of the two back-to-back pn junctions. Hence, the resistance between source and drain is very high ($> 10^{12}\,\Omega$). Thereby, the gate and the substrate form a capacitor (C_{OX}), enclosing the silicon dioxide as dielectric. If a positive potential V_{GS} is applied between gate and source, holes will be pushed away from the $\mathrm{Si}-\mathrm{SiO_2}$ interface, thereby forming an additional depletion region, that is inverse to those from the pn-junctions. Considering the one-dimensional case, the charge density ρ of the latter depletion region is given by

back-to-back pn junctions

$$\rho = q(-N_A), \tag{1.1}$$

with the carrier charge q and the doping strength N_A. The resulting electric field can be obtained by applying Gausses law and determining the integration constant C by evaluating $E(x)$ at the boundaries of the depletion region ($x = 0$ at $\mathrm{Si}-\mathrm{SiO_2}$ and $x = x_d$ at the depth of the depletion region in the bulk).

electric field

$$E(x) = \int_0^x \frac{\rho}{\varepsilon} dx = \int_0^x \frac{-qN_A}{\varepsilon_{\mathrm{Si}}} dx = \frac{-qN_A}{\varepsilon_{\mathrm{Si}}} x + C \tag{1.2}$$

$$\text{with} \quad C = \frac{-qN_A}{\varepsilon_{\mathrm{Si}}} x_d \tag{1.3}$$

$$E(x) = \frac{qN_A}{\varepsilon_{\mathrm{Si}}} (x_d - x) \tag{1.4}$$

Relating the electrical field to the Fermi potential (Φ_F) and the surface potential (Φ_S) yields

$$-\int_0^{x_d} E(x) dx = \int_{\Phi_S}^{\Phi_F} d\Phi = -\frac{qN_A {x_d}^2}{2\varepsilon_{\mathrm{Si}}} = \Phi_F - \Phi_S, \tag{1.5}$$

thereby, the Fermi potentials of the semiconductor are given as

$$\text{p-type:} \quad \Phi_F = -V_t \ln\left(\frac{N_A}{n_i}\right) \tag{1.6}$$

$$\text{n-type:} \quad \Phi_F = V_t \ln\left(\frac{N_A}{n_i}\right). \tag{1.7}$$

Further, assuming $|\Phi_S - \Phi_F \geq 0|$, x_d can be calculated from equation 1.5.

$$x_d = \sqrt{\frac{3\varepsilon_{\text{Si}}|\Phi_S - \Phi_F|}{qN_A}} \tag{1.8}$$

The charge underneath the gate can be calculated by inserting the immobile charge of the acceptor ions

$$Q = -qN_A x_d = -\sqrt{2qN_A\varepsilon_{\text{Si}}|\Phi_S - \Phi_F|} \tag{1.9}$$

strong inversion

If a threshold voltage V_{TS} is applied to the gate, the substrate between source and drain will become inverted, hence, a n-type channel exists, that allows carriers to flow. This condition is referred to as *strong inversion*. In order to achieve this, the surface potential must become at least $\Phi_S = -\Phi_F$. Thus, if $v_{\text{GS}} = V_T$, Q_{b0} will be

$$Q_{\text{b0}} = -\sqrt{2qN_A\varepsilon_{\text{Si}}|-2\Phi_F|} \tag{1.10}$$

and in case of a reverse biased pn junction with an according v_{SB}, this becomes

$$Q_{\text{b}} = -\sqrt{2qN_A\varepsilon_{\text{Si}}|-2\Phi_F + v_{\text{SB}}|} \tag{1.11}$$

threshold voltage

Finally, an expression for the threshold voltage can be given by taking the components of V_{GS} into account that are necessary to achieve inversion: first, the difference in the work functions between the gate material (polysilicon) and the bulk silicon, denoted as Φ_{POLY}. Second, the gate voltage of $-2\Phi_{\text{F}} - \frac{Q_b}{C_{\text{OX}}}$, which is necessary to change the depletion-layer charge. Third, there is an additional voltage $-\frac{Q_{\text{SS}}}{C_{\text{OX}}}$, taking the additional V_{GS} into account that is caused by material impurities. Thus, using equations 1.10 and 1.11, V_T can be expressed as

$$V_T = V_{\text{T0}} + \gamma\left(\sqrt{|-2\Phi_F + v_{\text{SB}}|} - \sqrt{|-2\Phi_F|}\right) \tag{1.12}$$

$$\text{with} \quad V_{\text{T0}} = \Phi_{\text{POLY}} + -2\Phi_F - \frac{Q_{\text{b0}}}{C_{\text{OX}}} - \frac{Q_{\text{impurities}}}{C_{\text{OX}}} \tag{1.13}$$

$$\text{and} \quad \gamma = \frac{\sqrt{2q\varepsilon_{\text{Si}}N_A}}{C_{\text{OX}}}, \tag{1.14}$$

which is defined as the body-effect coefficient or the bulk-threshold parameter.

inversion of the substrate type

Consequently, if inversion is achieved and a voltage v_{DS} is applied between drain and source, a current i_D will be able to flow across the channel. Example I-V characteristics are depicted in figure 1.2. It is now assumed that the channel has the width W and v_{DS} is small. The charge per unit area dl of the channel length can then be expressed by

$$Q_i y = C_{\text{OX}}\left(v_{\text{GS}} - v(y) - V_T\right) \tag{1.15}$$

and therefore, the voltage drop along the channel length is

$$dv(y) = i_D dR = \frac{i_D dy}{\mu_n Q_i(y) W}. \tag{1.16}$$

Thereby, μ_n is the average carrier mobility in the channel. Solving the latter equation and integrating along the channel from 0 to L, resp. 0 to v_{DS}, results in

$$i_D = \frac{\mu_n C_{\mathrm{OX}} W}{L}\left[(v_{\mathrm{GS}} - V_T)\, v_{\mathrm{DS}} - \frac{v_{\mathrm{DS}}^2}{2}\right], \tag{1.17}$$

the Sah equation, drain current

which is called the Sah equation and has been developed by Shichman and Hodges [81] as a model for computer simulation. Note that equation 1.17 is only valid when $v_{\mathrm{GS}} \geq V_T$ and $v_{\mathrm{DS}} \leq (v_{\mathrm{GS}} - V_T)$. Thereby, the factor $\mu_n C_{\mathrm{OX}}$ is defined as the device-transconductance parameter, given as

$$K' = \mu_n C_{\mathrm{OX}} = \frac{\mu_n \varepsilon_{\mathrm{OX}}}{t_{\mathrm{OX}}}. \tag{1.18}$$

NMOS and PMOS transistors

For a detailed description of the physical CMOS model, especially consequences of latch-up, temperature and noise, the reader is referred to [4, 81, 82]. Note, that the n-channel devices are generally denoted as NMOS transistors, whereas the p-channel devices are denoted as PMOS transistors.

1.1.1 Operation Regions

Furthermore, the relation between v_{GS}[1] and v_{DS} defines the operation region of the transistor. Note that the conditions for v_{GS} and v_{DS} in the following are valid for an NMOS transistor, thus, in the case of a PMOS transistor, the relational symbols have to be inverted. A number of 4 operation regions are distinguished: first, the *cutoff* region, where $v_{\mathrm{GS}} = 0\,\mathrm{V}$ and the channel resistance is greater than $10^{12}\,\Omega$. Hence, there is no inversion at all and $i_{\mathrm{DS}} = 0\,\mathrm{A}$. Second, the *weak inversion* region, for which $0 < v_{\mathrm{GS}} < V_T + nV_{\mathrm{th}}$[2], thus, the absolute drain currents are relatively low and they are exponentially depending on the gate source voltage. Third, for $v_{\mathrm{GS}} \geq V_T + nV_{\mathrm{th}}$, the substrate is actually inverted and the transistor is in the *strong inversion* state, which features a parabolic characteristic for $v_{\mathrm{DS}} \leq v_{\mathrm{GS}} - V_T$. This operation region is also called *linear* or *ohmic* region, since the drain source current is almost linear for small v_{DS}. As the term *ohmic* already suggests, the transistor can be used as voltage controlled active resistor in this operation mode. Fourth, if $v_{\mathrm{DS}} \geq v_{\mathrm{GS}} - V_T > 0$, i_{DS} will not further increase and the transistor is in *saturation*. In this case, the transistor resembles a voltage controlled current source.

cutoff

weak inversion and strong inversion

saturation

The influence of the different operation regions is not limited to the so-called large signal model presented above. Higher order effects, dynamic behavior and the various intrinsic device capacitances are also depending on the operation mode of the transistor. Some important small signal parameters, e.g. the channel transconductances $g_m = \partial i_D / \partial v_{\mathrm{GS}}$ and $g_{\mathrm{ds}} = \partial i_D / \partial v_{\mathrm{DS}}$ strongly depend on the transistor's operation mode. Further, even

[1] A lower case letter indicates that the quantity is variable, whereas an upper case letter stands for a constant value.

[2] n is the so-called subthreshold slope parameter. It is process specific and it is needed for modeling the exponential behavior of the I-V characteristic in *weak inversion*.

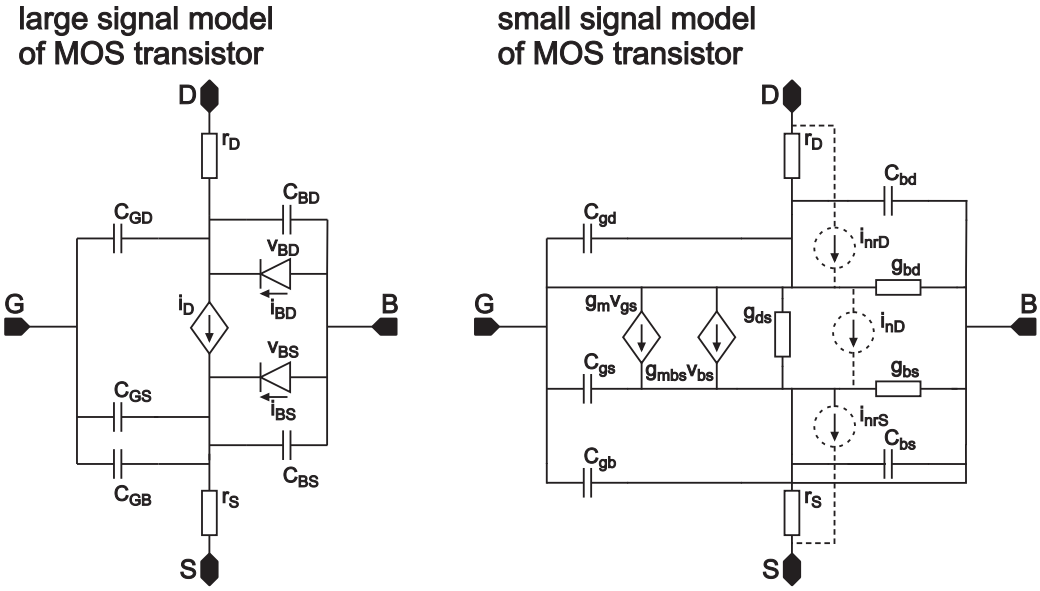


Figure 1.3: *Left: the large signal model of a transistor, which is used to find the DC operating point. Right: a linearized small signal model, on which the equations for computer simulations are based. Note that the small signal model equations are depending on the DC operating point.*

device mismatch on the silicon die[3] causes threshold voltage variations, that may lead to infeasible strong sensitivity to small variations of v_{GS}.

1.1.2 Parasitic Capacitances

large signal model

The presented large signal model includes several characteristics of the MOS transistor, such as parasitic capacitances of boundaries, channel resistance and noise. A complete large-signal model is depicted in figure 1.3. Thereby, the diodes are used for modeling the leakage currents and must always be reverse-biased for proper transistor operation. The leakage currents are given by

$$i_{\mathrm{BD,BS}} = I_s \left[\exp\left(\frac{q v_{\mathrm{BD,BS}}}{kT} \right) - 1 \right] \tag{1.19}$$

where I_s is the saturation current of a pn junction, q is the charge of an electron, k is the Boltzmann constant and T is temperature in Kelvin units.

The resistors r_D and r_S correspond to the ohmic resistance of source and drain and are only important for high drain currents. Typically, their resistances are in the order of $10\,\Omega$. The capacitors, formed by the several interfaces, are depicted in figure 1.1 and in figure 1.3. There are three types of such capacitances: first, C_{BD} and C_{BS}, which are formed by the depletion regions of the reverse-biased pn junctions of source and

capacitance of the depletion region

[3] A 'die' is a small piece of silicon, on which the fabricated circuit is located and which is cut out of the wafer. The dye has to be put in a package and to be connected to its pins, in order to operate it. The wafer, in turn, is a slice of an artificially grown silicon mono-crystal, on which the circuit is built by means of photo lithography and chemical deposition of the different layers (metal, polysilicon, silicon-oxide).

drain respectively. Those capacitances are a function of the reverse-bias voltage and are additionally influenced by their side boundaries to the depletion region, which is denoted as *sidewall effect*, and can be described with

$$C_{\mathrm{BD,BS}} = \frac{WL_{\mathrm{diff}}}{(1+V_{\mathrm{D,S}}/\Phi_b)^{MJ}} \cdot CJ + \frac{2(W+L_{\mathrm{diff}})}{(1+V_{\mathrm{D,S}}/\Phi_b)^{MJSW}} \cdot CJSW, \tag{1.20}$$

where, W and L_{eff} define the geometry of the terminal diffusion and $V_{\mathrm{D,S}}$ denotes the terminal potential. The parameters MJ, MJSW, CJ, CJSW, CGXO[4] and C_{OX} are process dependant constants and Φ_b is the working potential of the bulk. The first addend of 1.20 accounts for the area junction at the surface of the terminal diffusion, while the second addend represents the sidewall effects.

parasitic gate capacitances

Second, the capacitors related to the gate, which are dependent on the operating region of the transistor, namely C_{GD}, C_{GS} and C_{GB}. Those capacitances, in turn, can be calculated from C_{OS}, C_{OX} and C_{OD} according to

$$C_{\mathrm{GB}} = \begin{cases} C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} + C_{\mathrm{GS}} + C_{\mathrm{GD}} = C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} + L_{\mathrm{eff}}CGBO & \text{off} \\ C_{\mathrm{GS}} + C_{\mathrm{GD}} = L_{\mathrm{eff}}CGBO & \text{saturation} \\ C_{\mathrm{GS}} + C_{\mathrm{GD}} = L_{\mathrm{eff}}CGBO & \text{nonsaturated} \end{cases} \tag{1.21}$$

$$C_{\mathrm{GS}} = \begin{cases} C_{\mathrm{OS}}W_{\mathrm{eff}}L_{\mathrm{eff}} = W_{\mathrm{eff}}CGSO & \text{off} \\ C_{\mathrm{OS}}W_{\mathrm{eff}}L_{\mathrm{eff}} + \frac{2}{3}C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} = W_{\mathrm{eff}}CGSO + \frac{2}{3}C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} & \text{saturation} \\ C_{\mathrm{OS}}W_{\mathrm{eff}}L_{\mathrm{eff}} + \frac{1}{2}C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} = W_{\mathrm{eff}}(CGSO + \frac{1}{2}C_{\mathrm{OX}}L_{\mathrm{eff}}) & \text{nonsaturation} \end{cases} \tag{1.22}$$

$$C_{\mathrm{GD}} = \begin{cases} C_{\mathrm{OD}}W_{\mathrm{eff}}L_{\mathrm{eff}} = W_{\mathrm{eff}}CGDO & \text{off} \\ C_{\mathrm{OD}}W_{\mathrm{eff}}L_{\mathrm{eff}} + \frac{2}{3}C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} = W_{\mathrm{eff}}CGDO + \frac{2}{3}C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} & \text{saturation} \\ C_{\mathrm{OD}}W_{\mathrm{eff}}L_{\mathrm{eff}} + \frac{1}{2}C_{\mathrm{OX}}W_{\mathrm{eff}}L_{\mathrm{eff}} = W_{\mathrm{eff}}(CGDO + \frac{1}{2}C_{\mathrm{OX}}L_{\mathrm{eff}}) & \text{nonsaturation} \end{cases} \tag{1.23}$$

(1.24)

gate-source and gate-drain overlap

and account for the gate-source and gate-drain overlap as well as for the gate-bulk capacitor. Again, the values CGSO, CGDO, CGBO and C_{OX} are process dependent parameters with the unit $\frac{F}{m}$. Thereby, C_{OX} represents the third type of capacitance, which is depending on the device geometry—the polysilicon of the gate and the bulk silicon form a plate capacitance, with the SiO_2 as dielectric—, but not on the operation mode.

1.1.3 Large Signal and Small Signal Model.

linearized model

As yet, the large signal model of MOS transistors is considered, which is used for finding the DC conditions, hence, the DC operating point of the device. Additionally, a linearized small-signal model, graphed in figure 1.3 is available, in order to simplify calculations after the DC operating point is found. The parameters of both models are closely related albeit the small-signal model is only valid for small changes in the vicinity of a given operating point. Consequently, the values of the small-signal parameters strongly depend on the given DC operating point and can take on several alternate forms. Note that the

[4] MJ=bulk-source/drain grading coefficient, MJSW=MJ for sidewalls, CJ=zero bias, bulk-source/drain capacitance and CJSW=CJ for sidewalls.

corresponding large-signal subscripts will be lowercase in the small-signal case. Thus, the small signal channel transconductances are given as

$$g_{\mathrm{m}} = \begin{cases} \cong \sqrt{2K'I_D\frac{W}{L}} & \text{DC current} \\ — & \text{DC current and voltage} \\ \cong \frac{K'W}{L}(V_{\mathrm{GS}}-V_T) & \text{DC voltage} \end{cases} \tag{1.25}$$

$$g_{\mathrm{mbs}} = \begin{cases} — & \text{DC current} \\ \frac{\gamma\sqrt{(2I_D\beta)}}{2\sqrt{2|\Phi_F|+|V_{\mathrm{SB}}|}} & \text{DC current and voltage} \\ \frac{\gamma\sqrt{\beta(V_{\mathrm{GS}}-V_T)}}{2\sqrt{2|\Phi_F|+|V_{\mathrm{SB}}|}} & \text{DC voltage} \end{cases} \tag{1.26}$$

$$g_{\mathrm{ds}} = \begin{cases} \cong \lambda I_D. & \text{DC current} \\ — & \text{DC current and voltage} \\ — & \text{DC voltage} \end{cases} \tag{1.27}$$

$$\tag{1.28}$$

Thereby, $K' = \mu C_{\mathrm{OX}}$ is the transconductance, γ is the bulk threshold, $2|\Phi_F|$ is the strong inversion surface potential, λ is the channel length modulation and n is the subthreshold slope process parameter. In practice, process parameters are obtained from measuring the properties of the target technology and fitting the numerical model. The process parameters will be provided by the chip manufacturers, if a chip is fabricated with one of their processes.

1.1.4 Possible Configurations

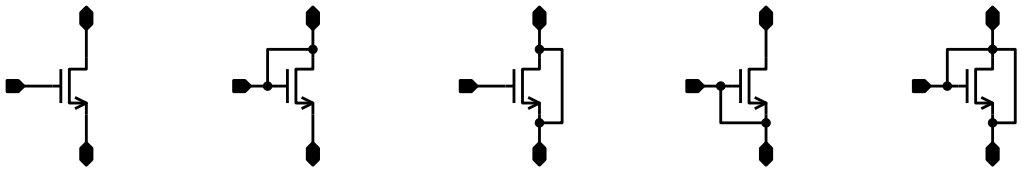


Figure 1.4: *As can be seen from the figure, a CMOS transistor can be configured as diode, capacitor or as pass transistor (switch, open/closed).*

diodes, capacitors and active resistors

In practice, the bulk of a transistor is connected to vdd (PMOS) or to gnd (NMOS), whereas the three remaining terminals source, drain and gate are used for assembling transistors to circuits. Possible configurations of the MOS transistor are depicted in figure 1.4. Transistors can be configured as diodes or capacitors, although those components could also be fabricated without using a transistor structure. Examples for the different configurations are depicted in figure 1.4. Contrary to that, it is not possible to integrate resistors with a sufficiently high resistance on the dye, due to the fact that the sheet resistance of polysilicon is very low in standard CMOS processes. If resistances greater than a few kΩ are desired, either a specialized process will have to be chosen, that features an

additional layer of high-resistive polysilicon, or transistors will have to be used as active resistors, as described in section 1.1.1. Thereby, the transistor has to be operated in the linear region, in order to achieve the desired I-V characteristic. Furthermore, it is possible to realize resistors as switched capacitors, which provide a high degree of linearity, although such circuits require nonoverlapping clock signals with a much higher frequency than the voltage variations. Switched capacitors are mostly used in telecommunication applications.

1.2 Realizing Switches with Transmission Gates

Transmission gates are one of the most important circuits of the FPTA chip, which is used for the experiments in this thesis and is introduced in chapter 3. The reason for this is that the manifold configuration options of the transistor array are provided by a large number of switches, hence, transmission gates. Thus, it is possible to realize a great number of almost freely scalable circuits on the FPTA simply by opening or closing the according switches. One pass transistor, which is closed by applying vdd/gnd (NMOS/PMOS) and opened by applying gnd/vdd to the gate (NMOS/PMOS), as depicted in figure 1.4, is the most simple realization of a switch. A major drawback of the latter approach is the limited input voltage range, due to the constraint $V_{GS} > V_T$ (NMOS) or $V_{GS} < V_T$ (PMOS). This can be overcome by combining both, NMOS and PMOS, in order to compensate the limited V_{GS} of the respective other transistor. Consequently, as can be seen from figure 1.5, transmission gates consist of a PMOS and an NMOS transistor in parallel, which are opened and closed with gate voltages of opposite polarity respectively.

realizing the configurability of the FPTA

1.2.1 Parasitics of Transmission Gates

As can be seen from the small signal model of transmission gates in figure 1.5, these switches possesses several parasitic capacitances and, in the on state, a finite parasitic resistance. Since, in the case of a transmission gate, the gate voltages are either vdd or gnd, the transistors can be considered to be either in the cutoff or in the linear operation mode. The on-resistance of a transmission gate can be obtained from equation 1.17:

$$r_{on} = r_{DS} = \left(\frac{\partial i_D}{\partial v_{DS}} \right)^{-1} = \left(K' \frac{W}{L} (vdd - v_S - V_T - v_{DS}) \right)^{-1}. \tag{1.29}$$

Thereby, the channel length modulation λ is omitted. The total parasitic capacitance is obtained by simply adding the according parasitic capacitances from section 1.1.2.

The parasitics of transmission gates are of great importance to this thesis, due to the fact that they are extensively used as switches for realizing the configuration options of the FPTA, which is used for the presented experiments and is introduced in chapter 3. The on-resistance is considered to greatly influence the circuits, that are configured on the transistor array, since the mean on-resistance of the switches is about 330 Ω[5]. Due to this fact, the latter resistances have to be considered for testing FPTA circuits in SPICE simulations, in order to obtain the correct behavior. The SPICE simulations are described

on-resistance

[5] The mean on-resistance of the gate switches is 2310 Ω. However, the influence of the gate resistance is not as significant as the source and drain resistances.

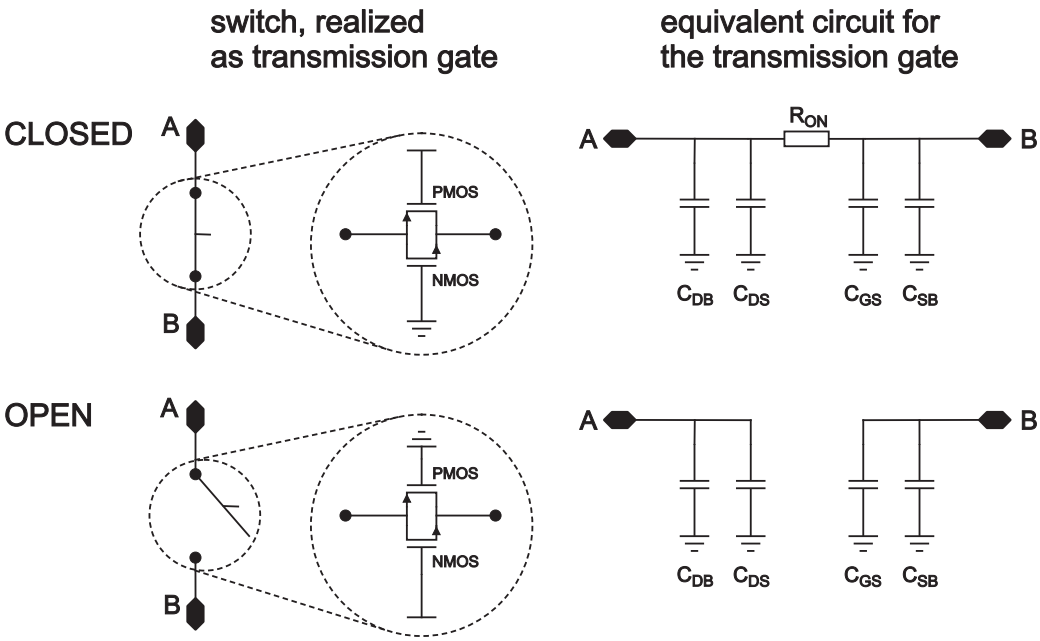


Figure 1.5: *The realization of an open and a closed CMOS switch as transmission gates is depicted. Additionally, small signal models of an open and a closed switch are provided respectively, in order to illustrate the parasitic capacitances and resistances of such switches.*

parasitic capacitances

in chapter 4, section 4.4. Contrary to that, it is assumed that the parasitic capacitances are negligible, since their signal bandwith is estimated to be $f_{\text{-3 dB}} = 66.7$ MHz (see [49], chapter 1), while the configured circuits are operated at a maximum speed of 4 MHz.

1.3 CMOS Transistor Modeling

The large-signal model of the MOS transistor, which is previously described in section 1.1, is useful for getting an insight into the operation principles of those devices and for hand calculations, although important second-order effects are not covered. Thus, more accurate simulation models are required for successfully fabricating application specific integrated circuits (ASICs). Consequently, the fabrication facilities provide according simulation parameters for their target processes to the designer. As yet, SPICE simulators support more than 60 different models, which cover different technologies and complexity levels. Each model consists of parameterized mathematical equations and according target technology dependent parameters, that are extracted by researchers at the respective fabs. While the mathematical model is included in the SPICE simulator and is most often public domain, the parameters themselves are company property.

simulation models with increasing complexity and accuracy

The main shortcoming of the large-signal model from section 1.1 is the fact that it is no longer valid for small device sizes down to 0.8 μm and moreover does not include effects like velocity saturation and intrinsic parasitic resistances. Consequently, more complex models have been developed, which provide the possibility to accurately describe the behavior of devices with sizes down to 0.1 μm. Thereby, important models are: first,

the SPICE level 3 model, which covers the range down to about $0.8\,\mu$m. Second, the BSIM3v3 model, which covers the range down to $0.25\,\mu$m and, finally, the BSIM4v4 model, which is valid for device sizes down to about $0.1\,\mu$m. Two models are briefly introduced in the following, namely the SPICE level 3 model, which is already quite accurate and is not yet too complex, and the BSIM3v3 model, which is used for the process, with which the FPTA of chapter 3 is fabricated. The BSIM model is not presented in detail, due to the volume of complex equations, that are necessary to describe it. A more detailed description of the BSIM equations can be found in [9, 24].

1.3.1 SPICE LEVEL3 Simulation Model

Contrary to the basic large-signal model of the MOS device, narrow and short channel effects ($< 3\,\mu$m) as well as temperature effects are considered in the SPICE level 3 model. Therefore, numerous additional parameters have to be taken into account for the calculations and also the equations have to be accordingly modified. The main parameters and considerations of the level 3 model are shown in table 1.1. Despite the fact that the level 3 model already includes numerous process parameters, like substrate properties and operating conditions, e.g. temperature and saturation, it is only valid for structures not smaller than $0.8\,\mu$m. Consequently, as structures of up-to-date processes are shrinking down to 60 nm and also the FPTA chip, which is used for the experiments in this thesis, is fabricated with a smaller $0.6\,\mu$m process, the BSIM3v3 model is briefly introduced in the next section.

valid down to $0.8\,\mu$m

1.3.2 BSIM3v3 Simulation Model

The equations of the BSIM3v3 model are far more complex, due to the fact that all relevant parasitic effects down to structures of $0.25\,\mu$m are included. As a consequence of this, 93 parameters are necessary to describe a PMOS or NMOS transistor with the BSIM3v3 model. In addition to the device parameters, capacitances and resistances of metal lines are also taken into account for the parasitic extraction of circuits. Moreover, the BSIM3v3 model can be successfully applied for simulating both analog and digital circuits and therefore has become the industry standard MOS transistor model. The most important deep-submicron effects, that are included in the BSIM model are listed in the following:

valid down to $0.25\,\mu$m

- threshold voltage reduction
- mobility degradation due to a vertical field
- carrier velocity saturation effects
- drain-induced barrier lowering
- channel length modulation
- subthreshold (*weak inversion*) conduction
- parasitic resistance of source and drain
- hot-electron effects on output resistance

characteristic	considerations
drain current	W/L is replaced with the effective size of the device, by subtracting the gate overlap regions. For the calculation of the effective drain-source voltage ($v_{\mathrm{DS}}^{\mathrm{eff}}$), the narrow-width threshold adjustment factor for the channel, the substrate doping concentration and the actual extent of the diffusion and metallurgical junction is taken into account.
threshold voltage	In addition to the geometrical implications, which are described for the drain current calculation, the intrinsic threshold, based on the work function of the substrate and the static feedback threshold adjustment is considered.
effective carrier mobility	Depending on $v_{\mathrm{DS}}^{\mathrm{eff}}$, the mobility of the minority charge carriers degrades. Thus, it is accounted by an effective carrier mobility. Moreover, the carrier mobility is temperature dependend, which is considered by a temperature coefficient.
saturation voltage	Again, the implications of $v_{\mathrm{DS}}^{\mathrm{eff}}$ are presumed. Additionally, the reduced charge carrier mobility is considered for the calculation of the saturation voltage.
channel length modulation	The channel length variation depends on the difference between $v_{\mathrm{DS}}^{\mathrm{eff}}$ and $v_{\mathrm{DS}}^{\mathrm{saturation}}$ and therefore a saturation field factor is included in the model.
weak inversion	The transition between the *cutoff* and the *strong inversion* region of a transistor, namely the *weak inversion* is also included in the level 3 model. This is achieved by providing additional equations and a subthreshold slope factor, which are used to model the exponential behavior of i_D in the *weak inversion* region. Thereby, *weak inversion* will be reached, if v_{GS} is close to the threshold voltage (V_T).

Table 1.1: *The main parameters and considerations of the level 3 model are shown. Although, the level 3 model already considers numerous process parameters, it is only valid for processes above* $0.8\,\mu$m.

1.4 CMOS Design Flow

behavioral model

schematic

Nowadays, there are actually four independent 'realities' in which a circuit exists: a behavioral model, the schematic, the layout and the final ASIC. First, the behavioral model describes the desired properties of the circuit in an abstract modeling language. Second, the schematic is a symbolic sketch of interconnected components. It contains the information about the circuit architecture, the type of the components and their basic component parameters, e.g. the W/L ratio of transistors and the values for capacitors and resistors.

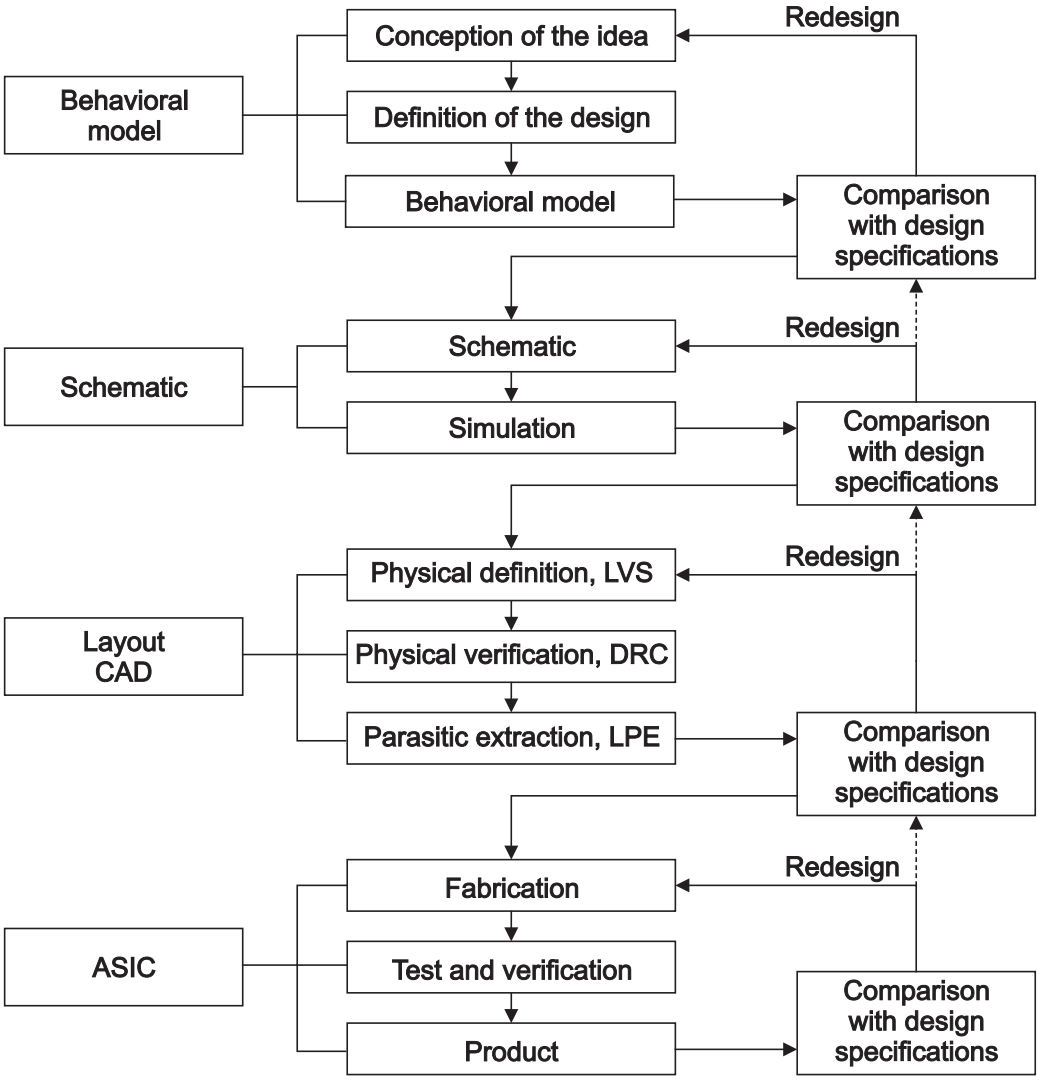


Figure 1.6: *The design flow for an ASIC is shown. The first step represents the conceptional stage, where the desired behavior of the chip is described. The necessary types of circuits are designed and are divided and structured in hierarchical schematics in the second step. First simulations can be performed with both the behavioral model and the schematics, hence, with the first two design loops it is achieved to remove conceptional faults. Subsequently, an according layout of the circuit has to be drawn, in order to define the physical representation of the ASIC and to be able to extract the whole parasitic information from it. Thereby, the third design loop will be closed, if all specifications are met. Finally, after the manufacturing facilities have fabricated the real chip, it can be tested in the real world and shipped, once the fourth design loop of testing is completed.*

At this level, it is already possible to predict the behavior of the drawn circuit by using the equations from section 1.1, although the circuit is not yet specified for all physical properties of a target technology in the real world. Third, the layout of a circuit is a floor- *layout*
plan of the different layers of metal, silicon-oxide, polysilicon and diffusion, that shall be

piled up on a silicon substrate (wafer), in order to fabricate the fourth representation of the circuit, namely the actual ASIC. For a successfull transfer of a circuit from the schematic to an operational chip, powerfull design tools are available, in order to deal with the—still increasing—high complexity of current systems, for which millions of transistors are intergrated on one single die. Those tools consist of editors for creating schematics, CAD tools for drawing the layout and comprehensive simulation environments, which allow for creating various test benches and for considering all significant parasitic effects. In order to achieve this, numerous steps are necessary: first, a validity check of the schematic and a design rule check (DRC) have to be successfully performed. Second, the components of the schematic have to be identified in the layout. This process is denoted as layout versus schematic (LVS). Subsequently, all device parasitics can be extracted from the layout with the layout parameter extraction (LPE). If the LVS and the LPE are done, simulations of back-annotated layouts[6] can be carried out and therefore crosstalk, parasitic layer and wire capacities as well as device mismatching are included in the simulation. Those extensive simulation and verification methods provide an elaborate design flow, graphed in figure 1.6, that reliably yields fully operational ASICs.

real world chip

[6] The term 'back-annotated layout' denotes a layout in which, on the one hand, each component is identified with its respective representation in the schematic and, on the other hand, all parasitic effects, which are extracted from the layout, are provided for simulation.

Chapter 2

Evolutionary Algorithms

The operation of an EA is inspired by the principles of natural evolution. This chapter introduces these principles on the macroscopic level, where organisms have to cope with the challenges of their environment, and on the microscopic level, which provides and develops the construction plans for building those organisms. Furthermore, it is shown how the ideas, which are drawn from the mechanisms of natural evolution, can be formulated as an evolutionary algorithm. The main constituents of such an algorithm are described and some extensions to the basic principle are presented. Theoretical considerations of desireable properties of an evolutionary approach are made, in order to substantiate their suitability for a great variety of tasks. The performance of evolutionary approaches as global optimizers and their properties as model-free heuristics are discussed. Moreover, the consequences of feasible and infeasible solutions are described and it is shown how infeasible solutions can be handled. A solution, of which not all properties can be tested in a given environment, is thereby denoted as infeasible solution. The focus is set on analog circuit evolution whenever examples are given in this chapter.

Evolutionary algorithms became very popular during the last 20 years, due to their reputation for being general purpose automated problem solvers. The advent of genetic programming (GP) [47] even lead to the opinion that EAs can be used as sources of invention. Indeed, EAs have been successfully applied to a great variety of optimization problems as, for instance, search, engineering design, scheduling and neural network training. The most intriguing advantage of using EAs is their generality, i.e. good solutions can be found without prior knowledge about the tackled problem. The latter generality will be an important property, if either the problem is too complex to be able to develop a simple solution, or it is desired to find unconventional or alternate designs.

2.1 Inspiration from Natural Evolution

The manifold algorithmic approaches within the field of EC are inspired from natural evolution mechanisms. Hence, the vocabulary is also borrowed from natural genetics.

the variety of species

Natural evolution achieved to produce an enormous variety of species, which are perfectly matched to their ecological niches. It is noticeable that about 2 million different species are currently living on earth and it is estimated that, including those which already again vanished, there existed in the order of a billion more. Natural evolution has obviously been able to develop these species in a manner that they are able to coexist by fitting into different ecological niches. However, there is interference and concurrence between different species. Moreover, the environment is steadily changing and therefore, each species has to be continuously adapted, in order to be able to survive. Looking at nature's achievements from an algorithmic point of view, evolution found solutions, represented by the individuals of the different species, to a given problem specification, namely to survive in a present environment. Thus, it is an interesting idea to see natural evolution as a general optimization algorithm. The question is now, on the one hand, how the natural concepts can be implemented in an algorithm and, on the other hand, if such an evolutionary algorithm will be suitable for solving problems. First of all, it is necessary to take a more detailed look at how natural evolution actually works and to derive its main principles.

natural evolution as a general optimizer

2.1.1 Darwinian Evolution

Charles Darwin proposed his theory *"On the Origin of Species"* in 1859 which is, alongside with the insights of molecular genetics, the foundation of evolutionary biology. An essential statement of Darwin's model of natural evolution is the survival of the fittest, i.e. those individuals, which are best adapted to their environment will more likely reproduce and survive. This phenomenon is denoted as natural selection and will necessarily occur, if a population of individuals has to compete for a limited amount of ressoures and has to escape from the same predators. Thereby, the basic driving forces of reproduction and the will to survive is presumed to exist—at least to a certain extent—in all individuals.

natural selection: survival of the fittest

Furthermore, if there was always only the same pool of individuals producing offspring, the individuals would, on the one hand, be able to specialize to their ecological niche but, on the other hand, the population would not be able to develop truly new features in order to adapt to a changing environment, since no really new genetic information would arise within the individual's genomes. Nevertheless, species are able to adapt to changing or different environments, due to a second important phenomenon, that actually introduces random changes to the individuals, namely mutation. The latter effect takes place with a certain probability during reproduction and results in a variation of traits in the offspring generation. As a consequence of iterated reproduction, mutation and natural selection, the traits of the individuals of the current generation, which produce offspring, are preserved. Thus, their traits are newly combined and slightly modified present in the next generation. The various individuals of the new generation then have to face the challenges of their environment, hence, solely the advantageous traits of those, which survive long enough to again produce offspring, are preserved, while other features are discarded.

variation through mutation

All higher life-forms reproduce by sexual mating. Thus, all individuals, which are

able to produce offspring with each other belong to the same species. Consequently, it will be principally possible that new species arise, if the diversion of subpopulations of one species reaches a point where sexual reproduction between those subpopulations is no longer possible. Geographical separation can be a reason for the latter effect. It is believed that fundamental diversification of the species has already taken place at a vary early stage of life and once there were no free ecological niches left, there was no room for new species. Therefore, nowadays it has become unlikely albeit not impossible that entirely new species evolve.

diversification of the species

Concluding, a species evolves by means of reproduction, random variations and natural selection. Thereby, natural selection affects the individuals, while the population as a whole is evolved. The three stated mechanisms of evolution are already featuring algorithmic characteristics, due to the presence of an iterated evolution loop, modification and selection operations. The individuals can be considered as candidate solutions to a given problem, although the suitable data structure can only be delivered by looking at the construction plan of the individuals themselves that is their genome.

2.1.2 The Genetic Level

The Darwinian principles, that describe a macroscopic view on evolution, go together with molecular genetics, which describe the evolution mechanisms from the microscopic point of view. First of all, it is an important fact that all living organisms actually feature two representations of themselves: first, the physiological and morphological appearance, conjoined with the organisms behavioral traits is denoted as phenotype. Second, the genotype, which contains the genetic encoding of the complete information of how to develop the phenotype from one single zygote [1]. All information about the organism, which is stored in the genotype, resp. the genome, can be inherited by its offspring and is thereby subject to random crossover and mutation. Additional skills, that the phenotype gains during its lifetime, are not influencing the genome and are therefore lost by its death. Its genetic information will also be lost unless the organism produces again offspring and passes its genome to the next generation. Thus, from molecular genetics point of view, the organism with its skills and experiences is merely a prototype, the construction plan of which will only be worthy to be preserved, if the organism succeeds in the challenges of the natural environment.

genotype and phenotype representation of organisms

variation of the genotype, selection of the phenotype

The genome consists of a set of genes, which are linearly arranged in several chromosomes. The number of chromosomes may vary from species to species, e.g. the human genome features 23 of them. Depending on the number of copies of the chromosomes, which are kept within each cell of an organism, it is called haploid (one copy) or diploid (two copies). Higher life-forms usually include two copies—a maternal and a paternal copy—of each chromosome in their cells, except for the gametes, which contain either only the paternal or only the maternal copy. Note that the gametes are haploid cells, i.e. egg cells and sperms, which are specialized to reproduction and contain only one version of the chromosome of the otherwise diploid cells. As a consequence of this, the reproduction mechanisms of haploid and diploid organisms differ. In the first case, the haploid

genes and chromosomes

haploid gametes

[1] A zygote is the haploid stem cell, which is formed by the course of fertilization where the haploid sperm cell and the haploid egg cell are merged.

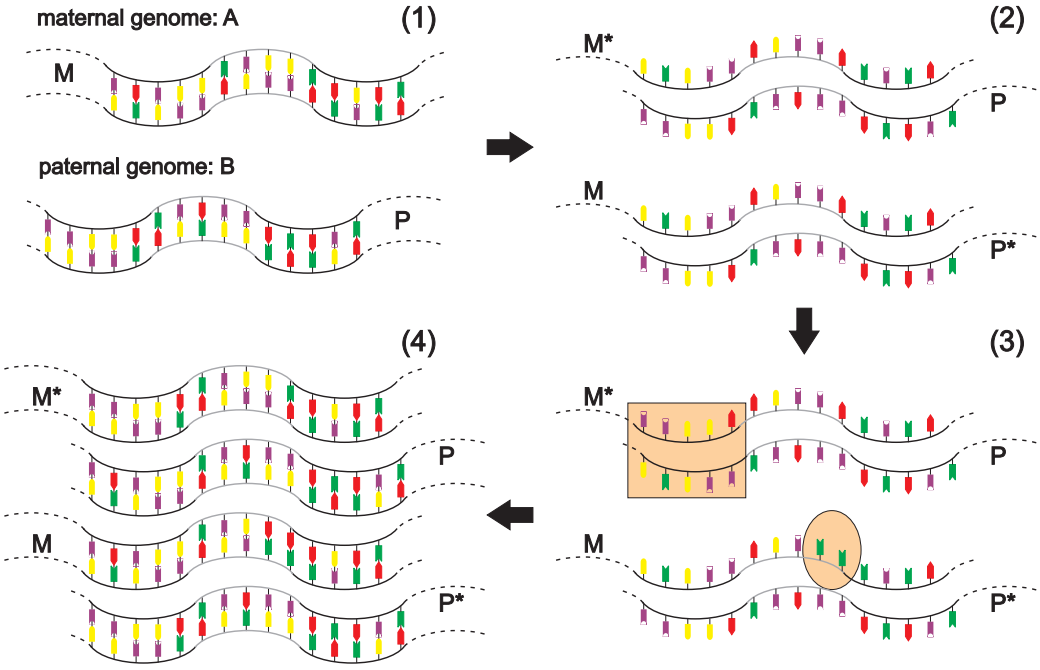


Figure 2.1: *The process of meiosis is illustrated in 4 steps, which are clockwise arranged. (1) The gametes of the mother and the father form a new cell, which then contains maternal and paternal genomes. (2) Subsequently, the chromosomes are split into chromatides and are aligned in that way that the same genes form pairs. (3) In this state, the chromosomes are subject to crossover processes between maternal and paternal genes, which is marked with the box. Additionally, mutations occur due to a certain probability as indicated with the ellipse. (4) The chromatides are completed with the respective base pairs and finally yield chromosomes for 4 new haploid cells. Note that the genetic information is typically stored as a long DNA chain, which, in turn, consists of four base pairs: guanine (G)-cytosine (C), thymine (T)-adenine (A), C-G and A-T.*

cell first duplicates its genome and subsequently divides itself into two genetically identical cells, by providing each half one copy of the genome respectively. This process is denoted as mitosis. Reproduction is a little bit more complex for haploid cells, due to a female and a male organism are necessary for sexual mating. The latter process is called

meiosis

meiosis and is shown in figure 2.1. Meiosis takes place in several steps: first, a female and a male gamete have to combine their chromosomes, in order to form a new diploid cell, which contains both a maternal and a paternal chromosome. Second, the chromosomes are duplicated and are aligned, i.e. those, which contain the same genes, form pairs. Third, the aligned chromosomes are split into identical halves (chromatides) and subsequently, genes are swapped between maternal and paternal chromosomes copies by means of crossover. Thereby, the crossing points are chosen randomly. Finally, the 4 counterparts to the chromatides are built and are recombined to four chromosomes, hence suffice for either four haploid or two diploid cells. Furthermore, during meiosis, the duplication and recombination processes of the chromosomes are subject to infrequent errors, causing slight random modifications in the genes of all four resulting chromosomes. Those various types of modification processes are denoted as mutations.

2.1.3 From Genotype to Phenotype

As described in the previous section, recombination (crossover) and mutation affects exclusively the genome, whereas only the phenotype is subject to natural selection. It is important to realize that information is only passed from genotype to phenotype and never vice versa. Thus, the developmental process from genotype to phenotype is itself of utmost importance to the success of natural evolution. Very complex self-organization processes, which are not yet fully understood, are building the phenotype with the information stored in the genome. It is amazing that the set of rules for development are organized in a manner that the complete set of genetically encoded properties is reflected by the resulting phenotype. However, due to the inherent randomness of self-organization, no phenotype will turn out exactly like any other, even if their genomes are similar or even identical. The latter process provides rich possibilities for natural selection to sort out the fittest individuals.

one-way information passing

rules for development

This developmental process is denoted as ontogenesis and is driven by amino acids. Those amino acids are generated by first transcribing snippets of the genetic encoding, i.e. creating copies of it. Note that the genetic information is most often referred to as the desoxyribonucleic acid (DNA), whereas the copied parts are called messenger ribonucleic acid (RNA). Subsequently, the RNA is translated into different amino acids, which are then concatenated to a great variety of proteins. An important property of those proteins is their ability to perform various tasks within the cell by taking on different shapes. The generation of proteins is a self-organized process; A certain concentration of a protein may excite or inhibit the generation of other proteins. By this means, the cells are able to differentiate, thus, become e.g. a liver, blood or skin cell, which is crucial for creating complex organisms. Since this genotype phenotype mapping is a fascinating process, albeit equally complex, a lot of research in the field of evolutionary computation is done to create suitable rule sets for self-organization.

proteins cause self-organizing cell differentiation

2.2 Building Evolutionary Algorithms on Nature's Concepts

From an algorithmic point of view, the achievements of natural evolution suggest to use its principles for solving optimization problems, since this is what evolution successfully did and still does: optimizing organisms in order to empower them to best cope with the challenges of their environment. In order to transfer the mechanisms of evolution, an algorithmic counterpart has to be defined. Thereby, the individuals represent candidate solutions and are, according to natural evolution, optimized in groups of various different individuals. This group is called the population. The test environment for the individuals is given by a fitness function, which measures their performance and assigns an according fitness value. Most often, better performance results in a greater fitness value, although, especially in the field of evolvable hardware, the fitness will sometimes be minimized, if it represents the deviation from a desired behavior. Furthermore, in the case of multi-objective optimization, more than one fitness value is assigned to each individual. For the reason that multi-objective optimization is a topic of this thesis, it is more closely described in chapter 7, section 7.1. Based on their fitness values the individuals are ranked and subsequently, the parents, that shall produce offspring, are selected. Offspring itself is produced by applying crossover and mutation variation operators to the genotype of

organisms as candidate solutions

fitness function as model for the environment

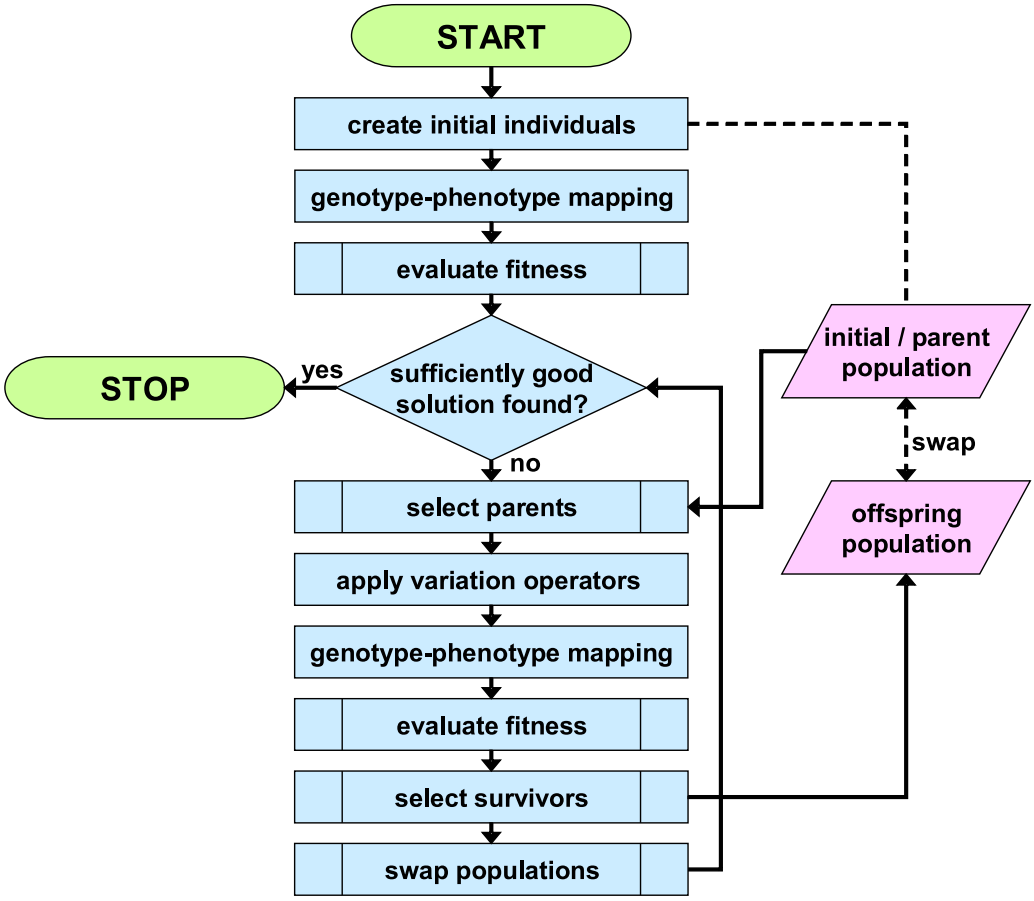


Figure 2.2: *A generic operation principle of evolutionary algorithms is shown in the figure. The implementations of the different modules are described in section 2.2.2.*

the parents. Consequently, in analogy to natural optimization of organisms, the algorithm continuously improves the ability of the individuals to solve problems, defined by the fitness functions. The algorithm stops as soon as a predefined termination condition is fullfilled, e.g. the desired target fitness is reached by one (or more) individuals or simply a maximum number of generations is exceeded (iteration limit is reached). The operation principle of an EA is illustrated in figure 2.2.

2.2.1 Historical Roots and Current Subareas

dialects of evolutionary computation

The roots of evolutionary computation are reaching back to the forties of the 20th century and then, since the 1960s the research field started to grow and spawned 4 main approaches to evolutionary computation. Three of them have been independently developed very early, namely evolutionary programming [12,22,23], genetic algorithms [27,37] and evolutionary strategies [11]. However, genetic programming, the fourth branch, was invented around 1990 by Koza [47]. Initially, evolutionary programming aimed for generating artificial intelligence by simulating learning processes, whereas genetic algorithms

and evolutionary strategies have been developed for parameter optimization. In that sense, genetic programming can be seen as an extension of genetic algorithms, due to the genetic encoding itself is designed for directing a developmental post-process in order to create the phenotype. For a survey of the current entire research field, the reader is referred to [21].

2.2.2 Modules of Evolutionary Algorithms

This section gives an overview of the main constituents of evolutionary algorithms and points out some examples of possible extensions to the basic modules. Five modules of the EA have to be specified, in order to define a particular implementation: first, the genotype, which is represented by a data structure, that contains all relevant parameters for expressing a candidate solution. This part is subject to the actual optimization. Second, a mapping algorithm, which is able to build the complete phenotype from its genotype. Alternatively, the phenotype can be created by a developmental process. In the latter case, the mapping algorithm is implemented as a set of rules for self-organization. Third, the variation operators, namely crossover and mutation, that describe how genes from parent individuals are combined in order to produce offspring and which random perturbations are introduced. Fourth, the fitness function is an important part of the algorithm, since it describes on the one hand the individuals test environment and, on the other hand assesses their performance by assigning fitness values. Last, an implemented selection scheme chooses the individuals, which shall be allowed to survive or to produce offspring, according to their fitness values. An overview of those modules and some examples are provided in the following and in figure 2.2. According to the focus of this thesis, given examples are biased towards analog circuit synthesis.

genotype & phenotype

variation operators

fitness & selection

Genotype.
The genotype contains all variables and parameters, that are necessary to completely describe a candidate solution to a given problem. Usually, data types are chosen, that allow for an efficient encoding, i.e. in the case of analog components, an integer identifier for the component type and for the target nodes respectively, and float values for the component size and properties. In the case of configurable hardware, it is also possible to directly work with configuration bit strings for the target substrate.

encoding candidate solutions

Phenotype mapping.
The genotype-phenotype mapping has to perform the task of constructing the phenotype representation of an individual, based on its genetic information.

building the phenotype

Fixed architecture. Consider configurable hardware as an example of this. In this case, the architecture is fixed and the task for the mapping function is to activate or deactivate features of the particular hardware.

Self-organization. In the case of a general circuit, the components and the connectivity of a circuit is given, but a place and route algorithm has to realize it with available ressources.

Variation operators.
Variation operators define the way in which parent individuals produce offspring and to

crossover (recombination) and mutation

which random perturbations the genomes are subjected. There are two types of variation operators, namely crossover (recombination) and mutation. A great variety of implementations is possible with respect to the data structure of the genotype. Thus, examples of popular, general approaches are given.

N-point crossover. The genomes are aligned and N randomly positioned crossing points are selected. Subsequently, the genome is cut into $n+1$ pieces, which are recombined after exchanging all odd (or even) parts.

Uniform crossover. Again, the genomes A and B are aligned and two new genomes A' and B' are created by successively randomly deciding whether the gene A_n is added to A' and B_n is added to B', or A_n is added to B' and B_n to A'.

Block crossover. Randomly sized and positioned blocks of genes are exchanged between A and B. Note that generally the blocks are randomly sized, but the size is the same for A and B.

Fitness function.

assessing the performance of individuals

The fitness function evaluates the individuals by assessing their performance and therefore it also contains the description of the problem to be solved. In the case of analog circuits, the problem is often specified by a desired target voltage characteristic and the fitness value is calculated as the deviation from the current behavior of a candidate solution. It is an important and at the same time difficult task to implement a suitable fitness function. This is due to the various possible shapes of the fitness landscape. If, for instance, the fitness landscape features numerous local optima, it will be unlikely to find the global optimum, although EAs are considered to relatively efficiently sample the search space, at least if the population is randomly initialized and sufficiently large. During the course of evolution, the whole population improves towards better fitness. Thus, it is on the one hand important to avoid premature convergence and on the other hand, it is desired to quickly find a solution. Consequently, the evolution process is inherently split into two phases: the phase of exploration, where the individuals are randomly spread over the whole search space and the phase of exploitation, where all individuals have already converged, hence, are sampling a region of good fitness.

Selection Schemes.

parent and survivor selection

The selection scheme chooses individuals, that shall be allowed to survive or to produce offspring. Thereby, selection is based on the fitness values of the individuals. The parent selection picks two or more individuals for producing offspring, whereas the individuals, which are passed unchanged to the next generation are picked by the survivor selection scheme. Principally, it is possible to perform numerous mathematical operations on the fitness values, resulting in a great variety of imagineable selection schemes. Thus, only the selection schemes, which are relevant to this work are presented.

Fitness proportional. In this case, the fitness values are mapped to the interval $[0,1]$ thereby stretching and compressing different regions of fitness. The selection pressure can be manipulated with arbitrary mapping functions. The individuals are then selected according to equally distributed random numbers between 0 and 1. As a consequence of that, it is possible to emphasize desired regions of fitness.

Rank based. The rank based selection does not take into account whether the fitness values are spread over wide ranges or are all similar. Once the ranking is done, selection

depends only on this rank.

Tournament. According to the tournament size, a number of individuals is randomly picked and the champion is determined by comparing their fitness values. Only the champion is finally selected, whereas all unsuccessfull competitors are discarded, although the latter individuals can still participate in other tournaments.

Diversity. A metric for measuring the diversity between the individuals of the population is introduced and selection is based on this crowding distance within the fitness landscape. The aim is to maintain diversity within the population and thereby efficiently explore the search space.

Elitism. Elitism is a simple and popular possibility to ensure that the best individual always survives. After the offspring generation is created and evaluated, the worst N individuals are replaced with the best N individuals of the parent generation.

Non-dominated sorting. This selection scheme works only with multiple fitness values per individual. The aim is to assess different tasks with different fitness values and allow individuals to survive as long as they are superior in at least one task. Since this selection scheme is used and refined in this thesis, it is described in detail in chapter 7, section 7.1.

2.2.3 Extensions to Evolutionary Algorithms

In addition to the basic modules of an evolutionary algorithm, described in the previous section, numerous extensions are investigated in the research field. An interesting approach is to implement different population models, e.g. island models or deme (subpopulation) models. If such an alternative population model shall be implemented, the selection scheme will have to be changed accordingly. In the case of island models, the algorithm has to independently evolve multiple separated populations, whereas the selection mechanism—to a certain extent—exchanges individuals between those island populations (migration) [72,73,97]. The difference between the island and deme model is that the demes are partly overlapping and mating is always restricted to the same subpopulation. Thus, individuals can only get to a fitter (or less fit) deme by diffusing through the overlapping region.

alternative population models

Another possibility is to replace the generation based evolution loop with a steady-state model. Therefore, individuals of different ages coexist in the same population and are able to produce offspring. The evolution loop is modified in a way that only a couple of individuals are replaced for each generation. In this case, an additional selection module has to decide, which individuals shall be discarded. Principally, it is also possible to manage populations with a variable population size as long as memory and computation limits are not exceeded.

a steady-state evolution loop

2.3 Characterization of Evolutionary Algorithms

As natural evolution, and therefore EAs, are considered as robust algorithms, that can be easily applied to find solutions for a great variety of problems, this section intends to point out general properties of such algorithms within the field of optimization. Due to the basic concepts of EAs, there are inherent limitations for what can be expected to achieve with an evolutionary approach. General properties of evolutionary algorithms are: first, the population based approach allows for simultaneously sampling different points

general properties

within the fitness landscape. Second, crossover, mutation and selection contain random decisions. Third, it is expected that, with a probability greater zero, recombination forms improved individuals and mutation introduces slight beneficial changes. Consequently, those three features influence the course of the optimization process.

2.3.1 Features for Global Optimization

In general, the shape of the fitness landscape defines the complexity of a given problem. Thus, only if it is possible to state a fitness function, which linearly depends on the input parameters, it will be possible to deterministically find the global optimum. Since this is usually not the case, hill-climbing algorithms and gradient-based approaches most likely get stuck in local optima. This will not necessarily happen, if an evolutionary algorithm is used. As described in section 2.2.2, the course of evolution has two phases, namely the exploration phase and the exploitation phase. If the population is carefully initialized, i.e. the individuals are uniformly spread over wide ranges of the search space, it will be more unlikely that the EA gets stuck in a local optimum during the exploration phase. Although, as the population continuously improves by means of selection, all individuals sooner or later end up in the vicinity of a good solution, which is not necessarily the global optimum. Thus, the probability of finding the over-all best solution can be increased by maintaining diversity within the population and thereby avoiding the so-called premature convergence of the solutions by prolonging the exploration phase.

exploration and exploitation phase

premature convergence

EAs will be a good choice, if multi-objective problems shall be tackled, due to the fact that the population is able to provide numerous solutions with different properties. Thus, if a clearly defined global optimum does not exist, because the problem will only be solved by trade-off solutions, it will become even more difficult to distinguish those solutions from local optima. In order to achieve this, it is necessary to implement a selection scheme, that in fact preserves different solutions (see chapter 7).

the population as a source of various solutions

2.3.2 Is Any Convergence Guaranteed?

It will be important to know, if at least any convergence of an EA can be guaranteed. On the one hand, it is desired to quickly find a good solution, while on the other hand, it is equally important to know if any solutions can be found at all. It is difficult to understand the optimization process of an EA as a whole, whereas it is easy to track the application of the variation operators for one generation.

understanding the optimization process

Thus, one possibility is to model EAs as finite Markov Chains, which are defined as a set of all possible states $= 1,\ldots,k$ and start $\varepsilon\,[0,1]^k$ with $\sum_{i=1}^{k}$ start$_i = 1$ are the probabilities for all states to be the initial state. Note that in Markov Processes, the state at the time t only depends on the state of $t-1$ and is independent from any earlier state, just like the new population is created from only the current one. The probability for state$_i$ to directly become state$_j$ is given by $\sum_{i=1}^{k} 1,\ldots,k \times 1,\ldots,k = 1$. As shown in [96], it can be proven that for a simple EA with a population size of 1 and a linear fitness function the optimum is found after $\leq \mathcal{O}(n \log n)$ steps.

Markov Chains

A second possibility is to use stochastic arguments, in order to predict convergence [96]. Considering a population of infinite size, the initial fitness values are distributed according to the Gaussian distribution. After creating the offspring population by applying the

stochastic predictions

variation operators once, the variation of the moments of the assumed Gaussian distribution can be calculated. Thus, it will be possible to predict the course of optimization to a certain extent, even if the implications of populations of limited size are taken into account. In the simplest case, the success rate of the variation operators can be obtained by comparing the number of improved individuals with the number of worsened individuals. Based on the latter considerations, the performance of the variation operators of chapter 6, section 6.3 is tested.

a nose for convergence

Admittedly, reliable statements can yet only be made for simple problems. Despite this, the latter considerations are useful for developing the nose for assessing problem specific implementations of variation operators. Furthermore, the fact that convergence can indeed be guaranteed for simple problems suggests that also for complex ones at least a local optimum can always be found.

2.3.3 Model-Free Heuristics

no problem specific knowledge is included

Only two of the modules of EAs, that are introduced in section 2.2.2, have indeed to be implemented in a problem specific way, namely the representation of the phenotype and the fitness function. Every other module is completely independent from the problem definition. Thus, neither further information about the optimized system nor prior knowledge about how a candidate solution actually works is included in the algorithm. Due to this, EAs are often denoted as model-free heuristics or black-box approaches. As a consequence of this, unlike most other heuristics, which rely on such problem specific knowledge, EAs can be successfully applied to a great variety of optimization problems, although the performance is possibly not as good in special cases.

two reasons for using an evolutionary approach

In the case of the evolution of analog circuits, the usage of model-free evolutionary algorithms is motivated by two main reasons: first, it is not genrally possible to formulate an optimization strategy for all tasks of analog circuit synthesis. Second, an unconstrained search could possibly discover previously unknown design principles or unusual solutions, which are intersting to investigate.

comparison with other optimizers

Aside from EAs, there are other model-free heuristics as, for example, random search, hillclimbing and simulated annealing [21, 59, 96]. Simulated annealing can be considered as a special implementation of an EA, since it is principally possible to implement the modules of the algorithm in a way that the cooling function is represented by a variable selection pressure and a population size of 1 is chosen. Random search samples the search space randomly and does not take any information of the fitness landscape into account, whereas hillclimbing, that always follows the steepest ascent in the neighborhood, is only suitable for performing local searches. The comparison between the different algorithms reveals additional advantages of the evolutionary approach, which optimizes a whole population of candidate solutions in parallel. On the one hand, it is possible to efficiently sample the search space during the exploration phase, as described in section 2.3.1. On the other hand, the modularity of EAs and the independently processed candidate solutions allow for an easy parallelization.

2.3.4 No Free Lunch Theorem

During the last decades, various implementations of EAs have been successfully applied to a great variety of tasks and are therefore rightly considered as general problem solvers, which are only outperformed by dedicated algorithms for particular tasks. Nevertheless, it has to be mentioned that evolutionary approaches are not proven to always find a sufficiently good solution in a reasonable amount of time. The No Free Lunch theorem by Wolpert and Macready [100], however, states that, if averaged over the space of all possible optimization problems, all nonrevisiting, model-free algorithms will perform equal. Thereby, algorithms, that are sampling the same point in the search space only once, are denoted as nonrevisiting. The latter constraint can be easily implemented in evolutionary algorithms with a sufficient amount of memory. In simple terms, if any algorithm is biased towards a particular set of problems, there will always be another set of problems for which it will perform accordingly worse. Moreover, it is only possible to break the limitations of the No Free Lunch theorem by incorporating problem specific knowledge into the algorithm, hence, by dismissing its model-free properties. Besides, numerous implementations of EAs can anyway not be considered as model-free, since it is questionable, in how far the problem specific design of the genotype and the variation operators already violates the model-free nature of the algorithm.

averaging over all problems, model-free algorithms perform equal

On the one hand, theoretical analyses of the performance of EAs are as yet either only available for too simple problems or the drawn conclusions remain on a too general level. On the other hand, the heretical question can be asked, if it is in practice relevant that a particular EA will perform bad on certain problems, since generally, it is rather important to quickly and reliably find good solutions for a specific task, than being able to solve all possible problems at once.

is practical application impeded?

2.3.5 Feasible and Infeasible Solutions

The fitness function serves as the only link between the problem and the algorithm in evolutionary computation approaches. As a consequence of this, it is necessary to 'perfectly' characterize the problem with the fitness evaluation process. In particular, it is important to correctly handle feasible and infeasible solutions. The population may contain such infeasible individuals, due to the fact that, in many cases, the genetic encoding offers richer possibilities than can be correctly handled in the given environment. Very clear examples from the field of circuit evolution are solutions, that contain unconnected components, are not even connected to an input/output terminal or are divided into unconnected parts. Nevertheless, such solutions possibly contain useful, well operating parts or, as in the case of the experiments in this work, represent feasible solutions in one test environment, but are infeasible in another. In general, the genotype maps to a set of solutions which is divided into the subset of feasible and infeasible solutions. Thus, it is moreover possible that the path to the desired feasible solution leads through the subset of infeasible ones. Note that the feasibility of an individual is not necessarily correlated to its fitness value in a given environment.

'perfectly' characterizing the problem

feasibility depends on environment

There are several ways of handling infeasible solutions: the simplest possibility is to reject infeasible solutions, which is a popular technique, although not very satisfying. Furthermore, in the case of non-convex feasible search spaces, it is only possible to reach

reject infeasible solutions

repair infeasible solutions

the feasible optimum by crossing infeasible regions. It is also possible to repair infeasible solutions in a post-processing stage before actually evaluating them, although it has to be mentioned that this implies additional problem specific algorithms. The latter process is related to a combination of evolution and learning and is denoted as *Baldwin effect* [59]. In this case, the question is whether the infeasible individuals are replaced with their repaired version or not (*Lamarckian evolution* [59]). Further, it is possible to use an aggregated fitness value, where infeasible solutions get a penalty offset. However, it is quite difficult to balance and implement aggregated fitness values effectively. Hence, if feasible and infeasible properties are separately evaluated, it will be more elegant to use a multi-objective optimization approach. Finally, one of the most reasonable heuristics for dealing with feasibility is to use specialized genetic representations and variation operators to inherently maintain the feasibility of individuals. Thereby, it is either possible to include according constraints in the genotype or the operators, or even encode a build instruction for the phenotype in its genes. Evolutionary programming and genetic programming techniques are examples of the latter category, whereas feasibility maintaining variation operators are developed in this thesis (see chapter 6, section 6.1.2).

aggregated fitness / multi-objective optimization

design care-taking genome and variation operators

Part II

Analog Circuits Evolution Framework

Chapter 3

The FPTA: an Analog Evolvable Hardware Substrate

This chapter introduces the architecture and the operation principle of the Heidelberg FPTA, a field programmable transistor array (FPTA) consisting of 16×16 *programmable transistors. The ASIC*[1] *is fabricated in CMOS technology. Furthermore, the hardware environment is described in which the chip is operated. It is possible to realize a great number of transistor circuits on the FPTA, simply by loading an appropriate configuration bit string into the chip. Due to the fact that reconfiguring the FPTA and subsequently measuring the characteristics of the configured circuit can be performed much faster than a simulation of the same circuit, the chip is a well suited substrate for highly iterative chip-in-the-loop measurements. Thus, in this thesis, the FPTA is operated for circuit evaluation alongside with an evolutionary algorithm, which requires a great number of evaluations of candidate solutions. The aim is to synthesize transistor circuits from scratch and—in the ideal case–without prior knowledge of analog electronics. Unfortunately, the prize for using real configurable hardware for circuit testing instead of a simulator, is the increasing influence of parasitic effects. The consequences of this are discussed in section 3.3. As yet, there are two research groups using custom made configurable ASICs for analog circuit synthesis, namely the JPL and the Electronic Vision(s) group from Heidelberg. A short comparison of the two approaches is given in section 3.1.3.*

There are yet two main approaches to evolvable analog arrays in the research field: first, the one of the group of Adrian Stoica at the JPL labs, who focusses on architectures that provide programmable and interconnectable cells of high complexity [2] [75, 76, 79, 106, 107]

[2] Every single cell contains as much complexity as to be configured as one operational amplifier (OP).

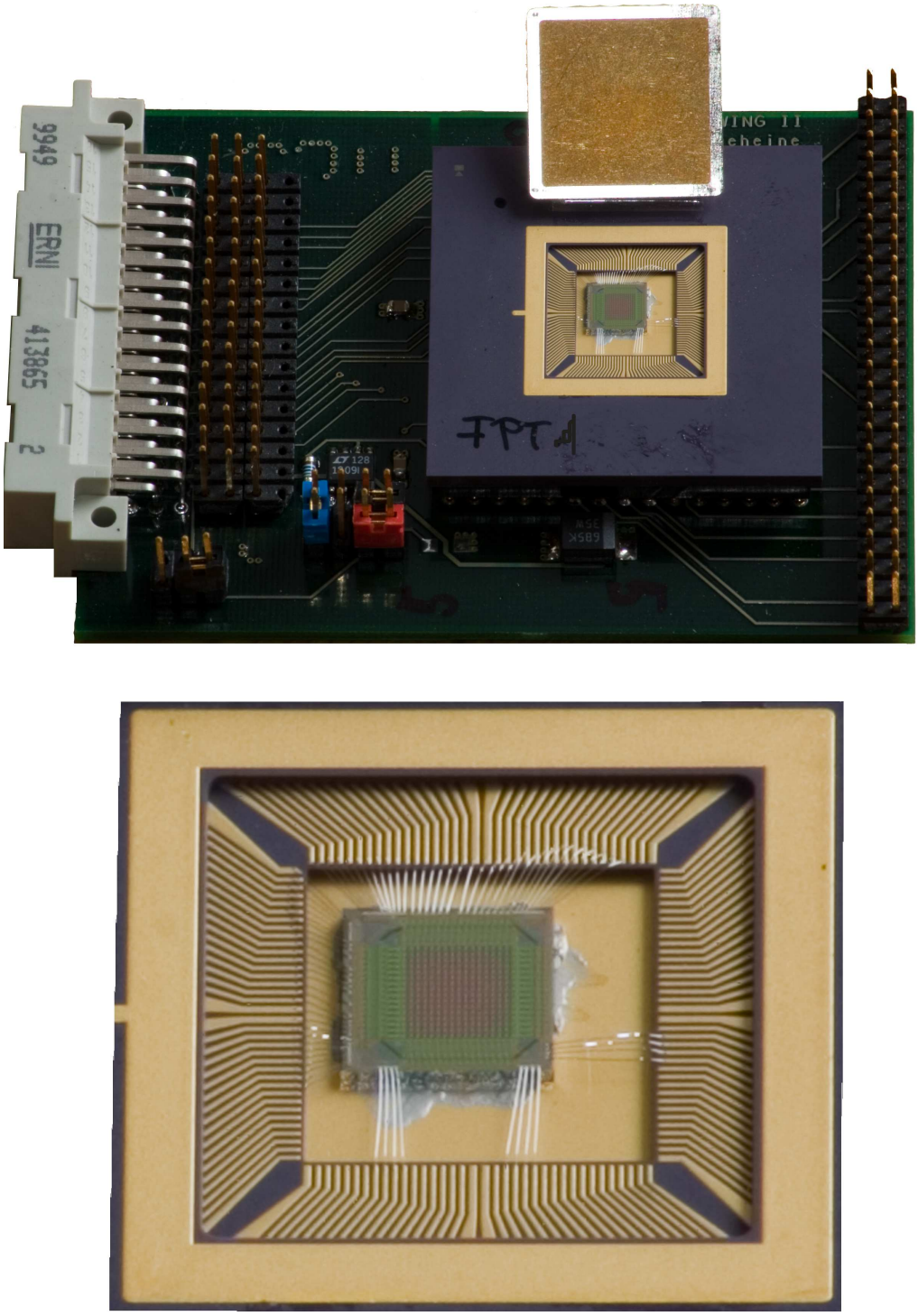


Figure 3.1: *Top: a photograph of the packaged FPTA chip, mounted on a test board. The test board can be attached to the PCI real-time test environment, described in section 3.2. Bottom: an enlarged view of the FPTA chip. The array of* 16×16 *configurable transistor cells (red area), the IO circuitry (green area) and the bond wires are recognizeable. The FPTA chip is a full-custom ASIC designed by Jörg Langeheine [49].*

and second, the one of the Electronic Vision(s) group of Professor Meier at the University of Heidelberg, where the Heidelberg FPTA was built, consisting of a large number of basic transistor cells. For convenience, I will refer to the Heidelberg FPTA simply as the FPTA throughout the remainder of this thesis. FPAAs[3] from other groups are marked with the name of the respective group. The idea of the latter topology is to provide a fine-grained substrate serving as a *Silicon Primordial Soup* [52][4]. The philosophy behind the two approaches differs: while the use of comparably complex cells aims to quickly finding robust solutions for problems that fit the predefined structures, the single transistor cells provide a higher degree of freedom to the evolving circuits and therefore offer the possibility to discover new circuit topologies. Nevertheless, finer grained substrates suffer from increasing parasitic effects and—due to the larger search space—slower convergence of the optimization algorithm.

3.1 The FPTA's Architecture

The very idea of the FPTA arose from the successful evolution of tone discriminators by Thompson [84, 85, 87] who achieved to distinguish two square waves of different frequencies by exploiting parasitic effects of a FPGA by means of an evolutionary algorithm. Since the underlying analog nature of the FPGA is the origin of its parasitic effects, Thompson's results suggest that a configurable analog substrate is possibly even more suitable for evolution experiments. The FPTA chip, which is used for all presented intrinsic hardware evolution experiments in this thesis, is a full-custom ASIC designed by Jörg Langeheine [49, 50, 52]. Fabrication technology of the FPTA is the Austria Micro Systems International AG (AMS) $0.6\,\mu$m CMOS process. A photograph of the chip is shown in figure 3.1.

the very idea of the FPTA

3.1.1 Configurable Transistor Cell

The configurable transistor cell is the basic building block for circuit synthesis on the FPTA. Each cell contains a matrix of either 20 differently sized NMOS or PMOS transistors, which behave like one single variable transistor to the outside world. Internally, the transistors are arranged in 4 columns with increasing width (W) and 5 rows with increasing length (L). Hence, it is possible to select one of the (almost) logarithmically spaced values $0.6, 1, 2, 4, 8\,\mu$m for the transistor length, by enabling the corresponding row of the transistor matrix. Due to the four transistors of one row are connected in parallel, the resulting width is programmable by enabling different combinations of those transistors. Since the selectable transistors feature the values $1, 2, 4, 8\,\mu$m, the resulting width can take on linearly spaced values between 1 and 15 μm. Note that it is possible to connect transistors in parallel in order to achieve a similar characteristic as one transistor with the sum of the sigle widths, whereas the length is not properly scalable by just serializing components.

internals of the configurable transistor cell

[3] Field Programmable Analog Arrays consist of configurable analog building blocks, that can be interconnected in various ways and are usually arranged as a matrix. These substrates aim at providing the analog counterpart to the widely applied FPGAs.

[4] For a more detailed summary about both approaches and analog design automation in general, the reader is referred to [26, 28, 49, 53].

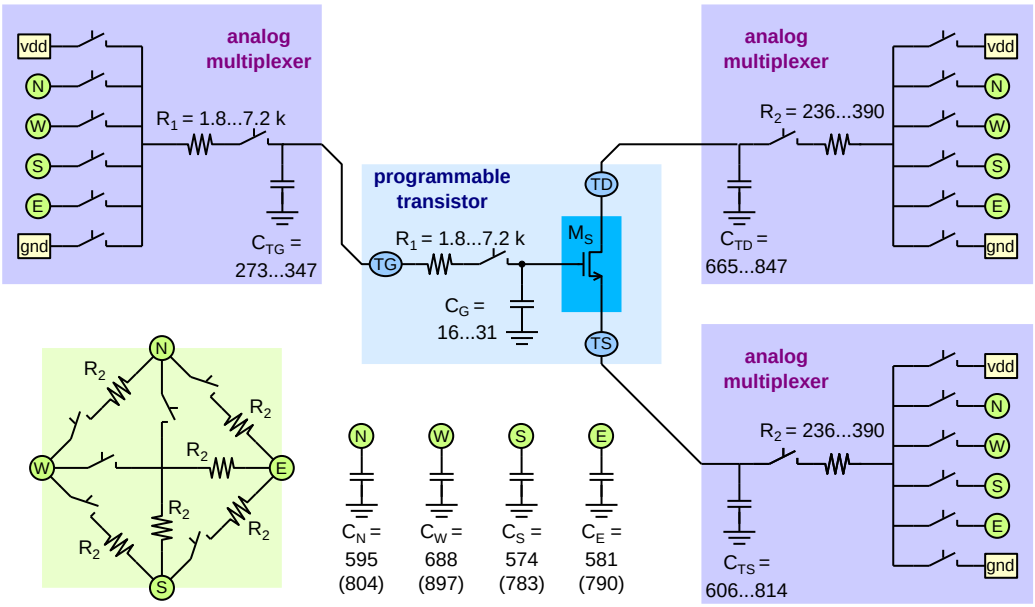


Figure 3.2: *Small signal equivalent circuit model of the complete programmable transistor cell. All drawn switches are assumed to be ideal. The resistor and capacitor values are understood to be quantified in* Ω *and* fF *respectively. The figure is taken from [49].*

terminal connections and routing switches

All inner transistors that are not contributing to the resulting W/L ratio are electrically turned off by pulling the gate to gnd (NMOS) or to vdd (PMOS). According to figure 3.2, all source and drain terminals of the inner transistors are directly connected to a common source and drain node that can be connected to one of the four available outside connections (N,S,W,E), vdd or gnd. Additionally, the FPTA cells provide 6 routing switches, which offer the possibility to directly connect any of two outside connections (nodes). The transistor array is designed in a way that the configuration of short-circuits is inherently impossible, thus, the substrate is self-destruction safe.

parasitic configuration circuitry

A consequence of the high configurability of an FPTA cell is the presence of a considerable amount of configuration circuitry, namely transmission gates, logic gates and analog multiplexers, as depicted in figure 3.2. Hence, the programmable transistors, represented by the FPTA cells, feature a different behavior than a standard transistor with equal W/L. Example characteristics[5] are depicted in figure 3.5. The consequences arising from this for the whole chip are discussed in 3.3. Further details about the configuration circuitry and the cell layout can be found in [49, 50].

3.1.2 Transistor Cell Array

16×16 NMOS/PMOS cells, arranged in a checkerboard pattern

The transistor cell array represents the actual evolvable substrate and consists of an array of 16×16 configurable CMOS transistor cells. As can be seen from figure 3.3, half of the cells are designed as NMOS transitors and the other half is designed as PMOS tran-

[5] The characteristics are obtained from simulation using the AMS 0.6 μm process parameters with which the FPTA is manufactured.

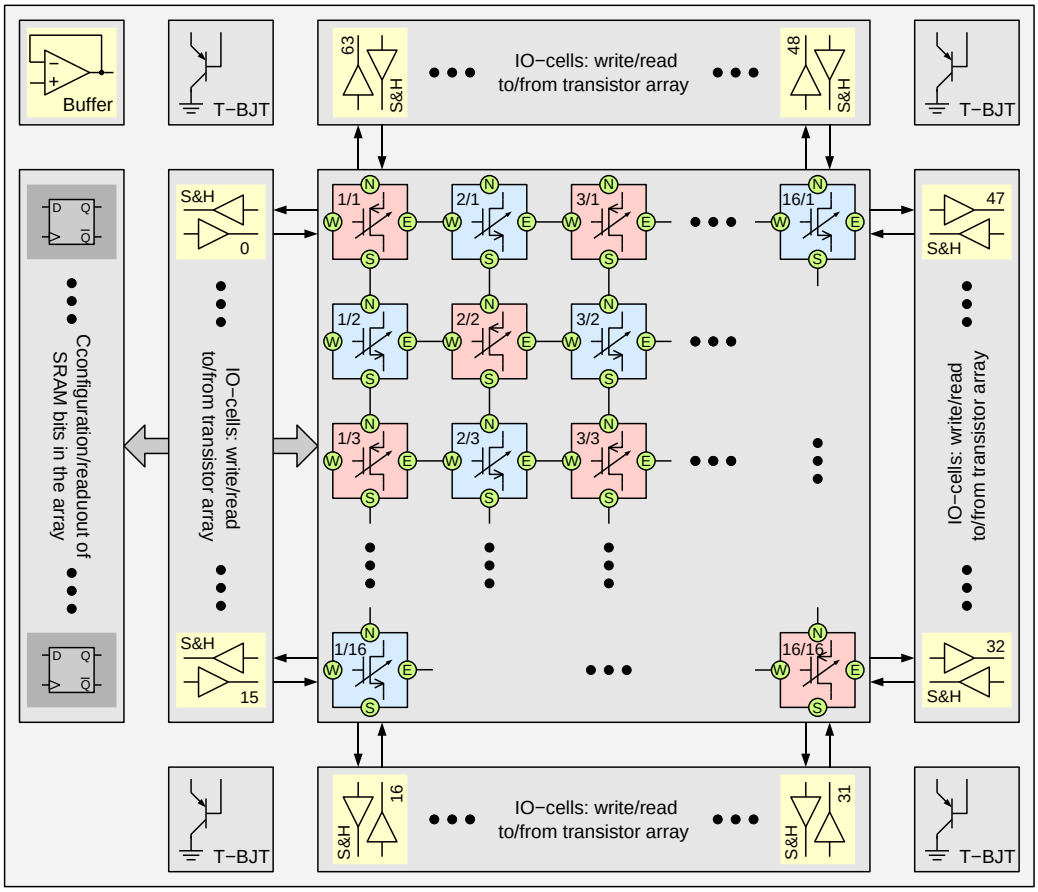


Figure 3.3: *A simplified architecture of the FPTA is depicted above. PMOS cells are colored light red whereas NMOS cells are colored light blue. The transistor array is enclosed by IO cells which are connected to the border transistor cells. Input voltage patterns as well as measured voltages are buffered in the IO cells and subsequently applied to the transistor array resp. read out by the controller. The figure is taken from [49].*

sistors. The latter two types of cells are arranged in a checkerboard pattern. External nodes (N,S,W,E) of adjacent cells are hard-wired. Combined with the abilities of the configurable transistor cells (section 3.1.1), it is possible to realize a great variety of CMOS transistor circuits on the FPTA. All possible circuits are represented by a corresponding configuration bit string.

FPTA circuits are represented by configuration bit strings

Furthermore the transistor array is enclosed by sample and hold IO cells, which can either apply input voltage patterns to every outer node of the border cells, or measure the output voltage characteristics of a configured or evolved, circuit. Up to 8 of these IO cells can be used at the same time, while the remaining, unused IO cells are switched to 'passive mode'—which means their internal sample and hold amplifiers are powered down and disconnected from the transistor cell array—in order to minimize their thermal and parasitic effects.

border cells for voltage IO

The IO cells are consecutively configured with new voltage samples or read out by the controller. As a consequence, the maximum sample and hold frequency with which the

	sampling frequency [MHz]								
input(s)	1			2			6		
output(s)	1	2	6	1	2	6	1	2	6
fastest settings	20.00	10.00	3.33	8.00	5.71	2.67	2.35	2.11	–
slowest settings	0.17	0.08	0.03	0.08	0.06	0.02	0.03	0.02	–

Table 3.1: *An overview of example IO setups for the FPTA and the resulting sampling frequencies for a system clock of* 40 MHz*. The possible values between the lower and the higher sample frequency cannot be programmed continuously but depend on the FPTA's timing scheme.*

sampling time for IO

input test patterns can be applied or voltages can be measured depends on the maximum settling times of the IO cells. Therefore, the maximum sample frequency decreases with increasing number of IOs cells. Thus, a trade-off between sampling time and accuracy has always to be found. In general, slowly varying voltage patterns can be applied with a higher frequency due to the fact that the sample and hold stages have to perform only small voltage changes in this case. Timing diagrams and an exhaustive description of the FPTA's internals can be found in [49]. Typical sample frequencies, used for the experiments of this thesis, are listed below in table 3.1. The system clock is set to 40 MHz for all presented experiments.

3.1.3 Comparison with the JPL FPTA

configurable analog arrays

Since, to the author's knowledge, the JPL FPTA is to date the only reconfigurable analog substrate that has been designed with the same intentions as the Heidelberg FPTA, it is of particular interest to present a short comparison of those two ASICs. Obvious similarities are that both chips are analog, highly reconfigurable and consist of basic cells, which are interconnected and arranged in a matrix. Thus, both substrates—presumed that the respective FPTA is attached to a suitable controller, which is able to perform reconfiguration, application of input voltage patterns and measuring of the outputs—are suitable for hardware evolution experiments, although both approaches feature considerably different architectures. An overview of the most important similarities and differences between both chips is given in table 3.2, in order to facilitate comparison. The answer to the question, which approach is more suitable depends on the goals that shall be reached. If higher convergence speed, complexity and robustness of the evolving circuits is desired, the JPLs approach will probably be the better choice, despite of the fact that the resulting circuits will inherently always be biased towards already known solutions. However, with the intention of finding entirely new design concepts, that might actually exploit parasitic effects of a particular substrate instead of suffering from them, the Heidelberg FPTA is more suitable. Additionally, a freely configurable substrate can still be pre-structured by software to enhance the applied search algorithm, e.g. with the possibility to use predefined building blocks [49, 54]. Yet, this will not diminish the higher amount of parasitic devices in the resulting circuits.

complex cells vs. fine grained substrate

	JPL FPTA2	Heidelberg FPTA
no. of programmable cells	8×8	16×16
no. of transistors per cell available to the synthesized circuits	44	1 programmable transistor which is realized with 20 real transistors
basic cell structure	fixed and predefined topology and components which are based on approved human (OP) designs	one programmable transistor, either NMOS or PMOS
components used	transistors, capacitors and resistors	exclusively transistors
configurability	the topology is configurable by opening or closing numerous switches	inner-cell routing capabilities as well as the possibility to select various different values for W/L ratio of the programmable transistor
total no. switches per cell	44	28
interconnection of cells	in both cases each cell is connected to the nearest neighbor in all four directions (N,S,W,E)	
input / output	inputs and outputs can be applied to arbitrary cells	inputs and outputs can only be connected to the border cells of the transistor array
signal flow	due to dedicated inputs and outputs the signal flow is constrained	free signal flow

Table 3.2: *A short summary of the most important similarities and differences between the JPL and the Heidelberg FPTA. The considered FPTA from the JPL is already in its third generation and is referred to as JPL FPTA2. Predecessors of the FPTA2, namely FPTA0 and FPTA1, featured similar architectures, although there were less programmable cells available.*

3.1.4 An Overview of FPAAs from Industry

Table 3.3 provides an overview of some examples of FPAAs sold by industry. These chips consist of reconfigurable analog circuitry, which can be readily applied to various problems of analog signal processing. Most often, the respective manufacturer also provides a design tool, which is able to generate configurations for the respective component. Although it is hardly possible for the FPTA to compete with those solutions from industry in the specific tasks, for which they are designed, there is no other substrate than the

manufacturer	device	capabilities
Anadigm	AN221E04, AN10E40	cells can be configured as: OPs, filters, DACs, ADCs and signal generators are present
Cypress	CY8C21x-XX, CY8C29-XXX	analog cells can be configured as: OPs, filters, DACs, ADCs digital cells can be configured as: counters, timers, PWMs, UARTs
Ilmenau	patent (D)	a German patent for a configurable transistor array architecture
Lattice	ispPAC10	configurable OPs and tunable filters
Motorola	MPAA-XXX	various different types of configurable filters
Zetex	TRAC*	configurable analog arithmetic functions

Table 3.3: *An overview of some examples of FPAAs which are sold by industry. There is some work in the field, where the FPAA from Zetex [69] and the FPAA from Motorola [70] are used for intrinsic hardware evolution. *TRAC stands for: totally reconfigurable analog circuits.*

industrial strength and suitability for research

FPTA, which provides the possibility of interconnecting single transistors. Consequently, the FPTA is more valuable in the case of research on unconventional and new circuit topologies.

3.2 The Hardware Evolution Setup

The hardware evolution setup consists of the FPTA chip and a standard personal computer (PC) that runs the EA and organizes the voltage test patterns. Additionally, a custom made Peripheral Component Interconnect (PCI) interface card, which is referred to as *DarkWing* and described in [13], represents a flexible FPGA based controller for the FPTA and provides an interface between the configurable transistor array and the PC. The EA is implemented in a flexible manner using the C++ programming language, combined with the gnu C compiler (gcc) and is more closely described in chapter 5. Therefore, the combination of the components above provides a flexible realtime measurement system for hardware evolution experiments. Since the generation, variation and management of a pool of configuration bit strings can be performed by an EA, the FPTA is, alongside with the latter measurement system, a suitable substrate for chip-in-the-loop analog hardware evolution.

a realtime measurement system for hardware evolution

chip-in-the-loop

3.2.1 The Controller: a Standard PC Hosting a FPGA-Based PCI Card

The *DarkWing* provides power supply voltages (3.3 V and 5 V) to the custom made ASIC in use. Furthermore, a digital-to-analog converters (DACs) offers the possibility to apply analog voltages to the chip and an analog-to-digital converter (ADC) is available for measuring analog voltages from the chip. Main controlling entity is a Xilinx Virtex-E FPGA with a total of 2 Mbyte of on-board static random access memory (SRAM) at hand. The latter components provide a real-time mixed signal test environment for various custom

made ASICs. As yet, the *DarkWing* board is successfully used within three different research projects of the Electronic Vision(s) group, namely the training of neural network chips [35, 67, 68], high dynamic range sensors [14] and analog hardware evolution ([49] and this thesis). For all presented experiments in this thesis the programmable system clock of the PCI board is set to 40 MHz.

FPGA-based controller for custom made ASICs

The PC contains an Intel Pentium 4 processor at a clock frequency of 2.4 GHz and the operating system is SuSE Linux 8.2 running kernel version 2.4.21 [6]. Additional software, which provides a device driver (linux kernel module) called *WinDriver6* [41] from JUNGO [7], is needed for the communication with the *DarkWing* board via the PCI bus.

setup of the host PC

Any communication between the EA (actually the PC) and the FPTA is carried out by calling the respective methods of a state machine, which is running on the FPGA. The source code for the state machine is written in very high speed integrated circuit (VHSIC) hardware description language (VHDL) and subsequently compiled with the appropriate tools from Xilinx [101, 102] for the FPGA used. Both, PC and FPGA, access the local SRAM of the *DarkWing* board where the input voltage patterns, configuration data for the FPTA and the measured results are stored. As a consequence of this, one measurement loop is carried out in 3 steps: first, the PC writes configuration data and input voltage patterns for the FPTA to the local SRAM of the PCI card. Second, those methods of the state machine which configure the FPTA and use the on-board DACs and ADCs to perform the measurements and write the results to the local SRAM are carried out. Third, the measured results are read back by the PC and are provided to the EA for evaluation. It is also possible to read back the current configuration of the transistor array using the same mechanisms, which is an essential feature for debugging.

communication between PC, controller and FPTA

3.3 Characteristics of the FPTA and the Hardware Environment

The following section shall give a short overview over the characteristics of the FPTA, that are of concern for the presented experiments. Accuracy and speed of the whole mixed signal measurement system is predetermined by the ADCs and DACs on the *DarkWing* board (section 3.2.1) and the accuracy of the FPTA's sample and hold buffers. For a detailed description of internal functionality and performance of the chip, please refer to [49].

3.3.1 Bandwidth

The bandwith of the evolving circuits is measured by applying an input voltage pattern of sine waves with increasing frequency to the west side and measuring the output at the east side of the FPTA. Input and output cell are interconnected by using the routing capabilities of the transistor array, i.e. the west-east routes of the 16 transistor cells between input

[6] For more information about the SuSE Linux distribution or the installation and usage of Linux kernels, the reader is referred to www.suse.com and www.kernel.org.

[7] The JUNGO windriver module provides a communication library for PCI communication via the PLX90xx chipset family [64]. Further information is available on www.jungo.com and www.plxtech.com.

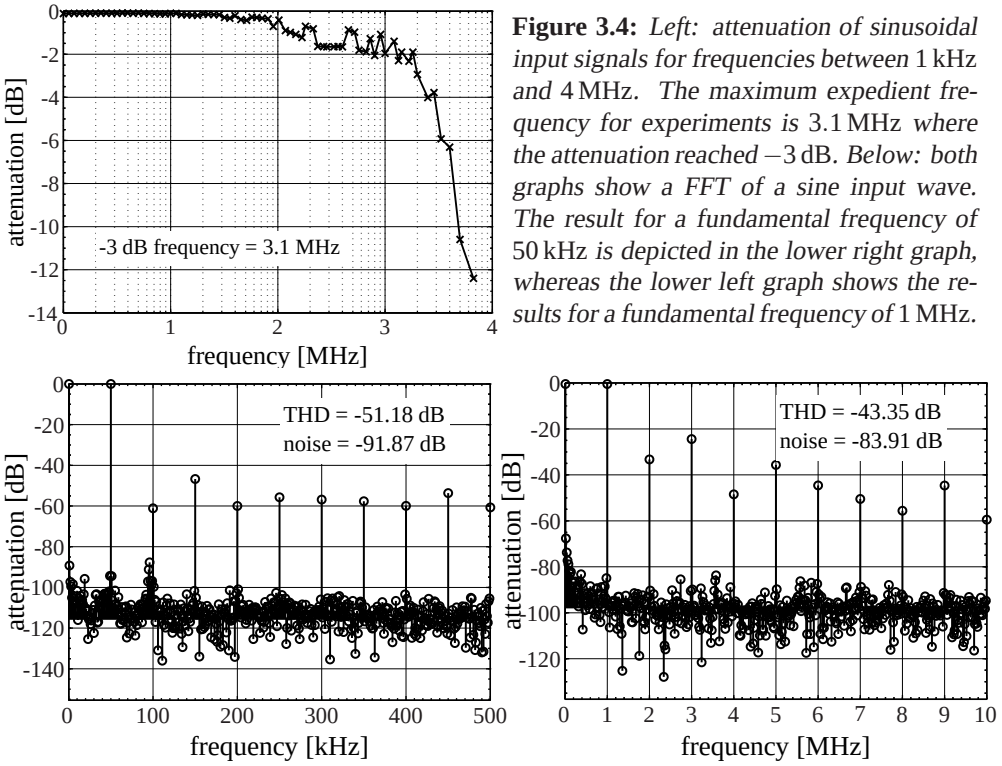


Figure 3.4: *Left: attenuation of sinusoidal input signals for frequencies between* 1 kHz *and* 4 MHz*. The maximum expedient frequency for experiments is* 3.1 MHz *where the attenuation reached* −3 dB*. Below: both graphs show a FFT of a sine input wave. The result for a fundamental frequency of* 50 kHz *is depicted in the lower right graph, whereas the lower left graph shows the results for a fundamental frequency of* 1 MHz*.*

maximum expedient frequency for synthesized circuits

and output cell are enabled. It can be seen from figure 3.4 that the maximum expedient frequency, at which the synthesized circuits are supposed to work, is about 3.1 MHz. Yet, a maximum frequency of 1 MHz is chosen for the experiments in this thesis, for the reason that total harmonic distortion and noise significantly increase at higher frequencies, as can be seen from figure 3.4, right.

3.3.2 Noise, Distortion and Accuracy

Experimental setup and input voltage patterns are equal to those of the previous section (section 3.3.1). Values for total harmonic distortion (THD) and THD + noise (THD+N), which are defined as the root mean square (rms) sum of all harmonics, divided by the amplitude of the fundamental frequency, are obtained by performing an FFT on the measured data and by calculating

$$\text{THD} = \frac{1}{|M(k_0)|} \sum_{l=2}^{N_{k_0}/2} \sqrt{M^2(lk_0)} \ , \tag{3.1}$$

$$\text{THD+N} = \frac{1}{|M(k_0)|} \sum_{\substack{k=1 \\ k \neq k_0}}^{mN_{k_0}/2} \sqrt{M^2(k)} \ , \tag{3.2}$$

$$\text{noise} = \frac{1}{|M(k_0)|} \sum_{\substack{k=1 \\ k \neq k_m}}^{mN_{k_0}/2} \sqrt{M^2(k)} \ . \tag{3.3}$$

Thereby, $M(k_0)$ represents the fundamental frequency component of the input sine wave and the $M(k_m)$ are the frequencies of the harmonics. The absolute amplitude of the input signal is 2.5 V. Assuming a linear behavior of the system, only the first 5 to 9 harmonics are considered for the calculation of the THD. In this case, the contributions of higher orders are negligible. Contrary to the THD, where only harmonics are taken into account, the value for THD+N is calculated from all fourier components, for the reason that noise is present in all frequency components whereas distortion effects are merely present in the harmonics.

THD and THD+N

Since the FPTA is bound to the hardware environment described in section 3.2, the measurements, depicted in figure 3.4, include the contributions of noise and distortion of the whole mixed-signal hardware setup. Consequently, the measured values for THD and noise represent the over-all performance of the hardware evolution system. THD and noise are measured for the two example frequencies 50 Hz and 1 MHz and the results can be seen from figure 3.4, right. Due to the fact that the value for THD already increased by about 8 dB at 1 MHz, the latter frequency is chosen as maximum for the experiments within this thesis. The value for noise is significantly below the value for THD in both cases. An additional reason for limiting the bandwidth to 1 MHz is the maximum sampling frequency of 20 MHz (see table 3.1) per sample. This will imply a maximum input signal frequency of 4 MHz, if the minimum number of 4 samples for one period is chosen, which is not sufficient to form an input sine wave. Hence, if a minimum of 20 samples per period is desired the maximum frequency is moreover limited for technical reasons.

feasible bandwith

sampling a sine wave

An easy specification for the accuracy of the hardware measurement system is to calculate the equivalent no. of bits (ENOB). This specification is originally used to provide an overall performance measure for data acquisition boards using an ADC [5, 56]. The ENOB will be obtained by evaluating the output of the system under test, if a sinusoidal input is applied and subsequently the FFT of the output is calculated. One way of expressing ENOB is through a correlation to signal-to-noise-ratio (SNR), given in dB as

ENOB: specification for over-all accuracy of the measurement system

$$\frac{1}{\text{THD+N}} \equiv SNR \tag{3.4}$$

$$\text{SNR} = ((\text{ENOB} \cdot 6.02) + 1.76) \quad \text{dB} \ , \tag{3.5}$$

$$\text{error} = \left(\left(\frac{1}{2}^{\text{ENOB}}\right) \cdot 100\right) \quad \% \ . \tag{3.6}$$

Using the results for THD+N from figure 3.4 as a measure for SNR, a value of ENOB=8.2 bits for 50 kHz and a value of ENOBs=8.0 bits for 1 MHz is calculated using equation 3.5. As a result, an error of 0.39 % can be obtained using equation 3.6, thus, an accuracy of $\Delta V = \text{error} \cdot 5\,\text{V} = 20\,\text{mV}$ is assumed for the entire measurement system for a power supply voltage of 5 V.

systematic voltage error +/- 20 mV

3.3.3 Influence of Configuration Circuitry

The output characteristic of plain transistors differs from the output characteristic of configurable FPTA transistors. Examples for different W/L ratios and transistor types are illustrated in figure 3.5. The I-V characteristic of a plain transistor is compared to a configurable transistor with transmission gates and to a configurable transistor where the

transmission gates are approximated by their mean on-resistance. The deviation of the latter characteristics significantly increases for greater W/L ratios.

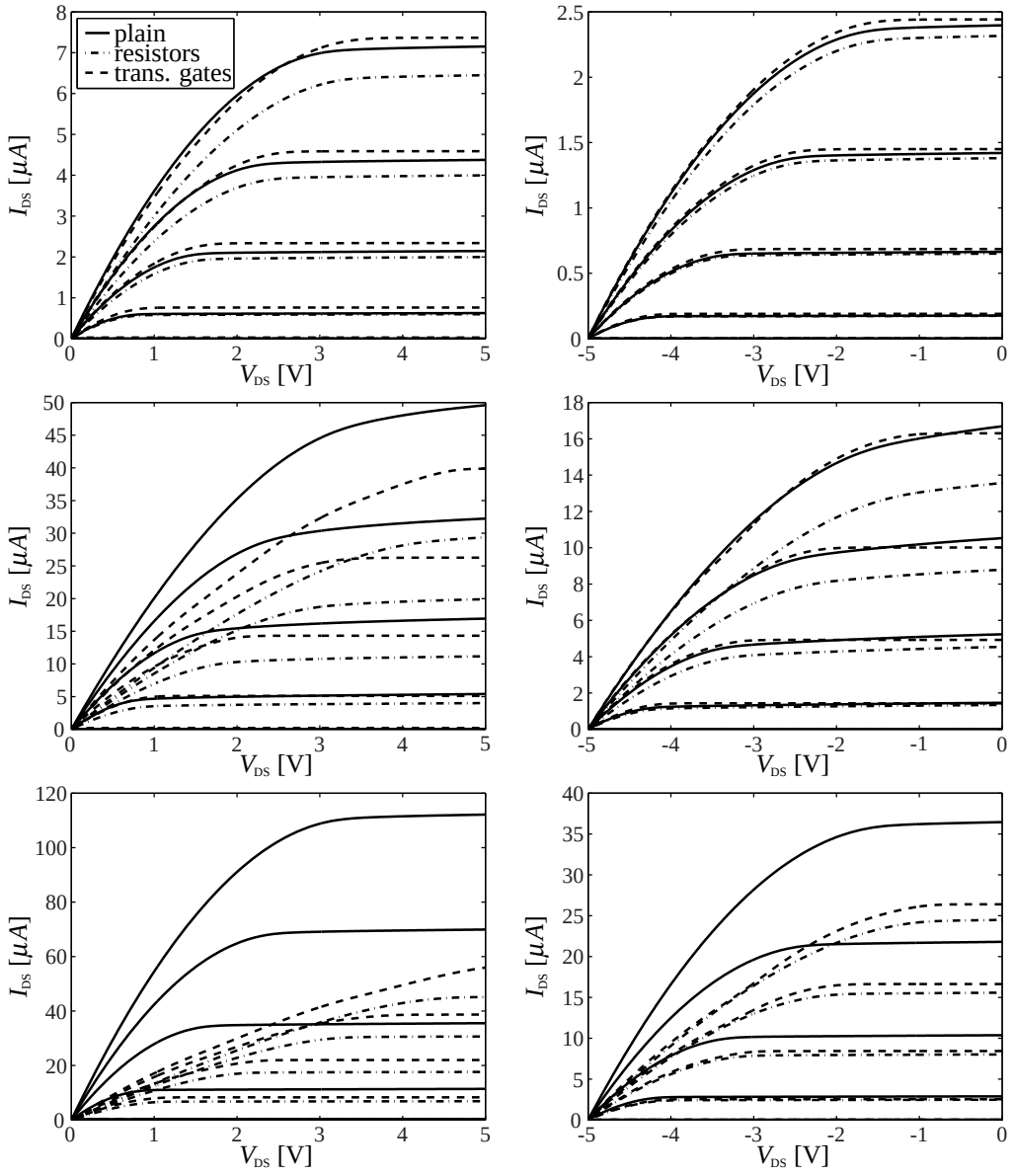


Figure 3.5: *The characteristics of NMOS (left) and PMOS (right) transistors are graphed above. In all cases, the characteristics are plotted for different gate-source voltages. Furthermore, the characteristics of plain transistors are compared to those of the actual transistor matrix on the FPTA. In the latter case, two types of simulations are carried out: one with all transmission gates included and one with transmission gates replaced with their mean on-resistance. The W/L ratios of both top graphs are 1/8, those of both graphs in the middle are 1/1 and those of both graphs at the bottom are 15/8.*

3.4 Why Hardware in the Loop for Circuit Evolution?

In contrast to the domain of digital circuits, where FPGAs have already reached a high level of complexity and provide a great number of ressources in combination with powerful software tools, which are capable of efficiently synthesize high-performance digital circuits for those substrates, the analog counterparts, namely FPAAs, are still in their infancy. Although configurable field programmable analog arrays for dedicated tasks, as shown in section 3.1.4 are already available, there is still a lack of multi-purpose substrates which could be used either as OP, filter or controller by simply using the according configuration bit-string.

lack of multi-purpose FPAAs

Other important applications for (re)configurable analog hardware are fault tolerance and BIST [10, 25, 44, 83, 104]. Fault tolerance can be easily implemented by reconfiguring damaged circuits using spare ressources. In the case of a BIST, the candidate circuit can be consecutively extended with various test circuitry with the intention of reducing ressource consumption of redundant test structures. Moreover, there are two main arguments for using hardware-in-the-loop for evolution experiments of analog circuits instead of an analog circuit simulator: first, a circuit, that is evolved on a real chip is obviously proven to work at least on just that substrate. Second, the simulation of complex analog circuits is highly non-trivial, thus, a very time consuming task. Unlike the simulator, once the inputs are applied to the FPTA, its outputs can be almost instantly measured. Nevertheless, the price that has to be paid using real hardware is to forgo the unconstrained configuration possibilities of simulation. Although, considering the smaller search space resulting from less configuration options, this can also be an advantage as long as the given topology does not inherently exclude desired solutions. Unfortunately, the latter question can not be answered until experiments that investigate evolvability of a given substrate have actually been carried out.

robustness and fault-tolerance

intrinsic reality of real hardware

faster circuit evaluation than in simulation

Chapter 4

Analog Circuit Simulator

This chapter intends to give an overview over the operation principle of analog circuit simulators. Different types of analyses, e.g. DC, transient or alternating current (AC), are introduced as well as SPICE netlists themselves, which represent circuits in a format that can be processed by circuit simulators. Important goals of this thesis are, first, to use the Berkeley SPICE3f5 [1] *analog circuit simulator as simulator-in-the-loop for the extrinsic* [2] *evolution of analog circuits and, second, to use it for verifying and further testing of the circuits, which are evolved on the FPTA. Thereby, extrinsic and intrinsic hardware evolution are terms which have been created by researchers from the JPL labs: intrinsic hardware evolution denotes experiments which are carried out on real hardware, whereas evolution experiments, in which a hardware simulator is used, are denoted as extrinsic. Additionally, a brief introduction to the Berkeley SPICE3f5 simulator is given and the Cadence analog design framework is shortly introduced. Furthermore, the procedures of automatically generating SPICE netlists and simplified schematics from intrinsically evolved circuits are described in this chapter. The automatic creation of schematics is realized with the silicon compiler interface language (SKILL), which is included in the Cadence software for the purpose of design automation.*

4.1 Introduction to Circuit Simulators

VLSI technology for CMOS processes

mixed-signal systems

The development of VLSI technology makes it possible to integrate millions of transistors on one single die (chip). Further, CMOS processes facilitate the combination of analog and digital subcircuits on the same substrate. The term *mixed-signal* is widely accepted for such systems. With increasing complexity, it becomes more and more important to accurately verify circuit designs in computer simulations before fabricating expensive

simulation program with integrated circuits emphasis

chips. This task is carried out by using circuit simulators, which are mostly based on SPICE. The original SPICE was developed by Larry Nagle and Donald Pederson in 1975 at the Electronics Research Laboratory of the University of Berkeley and the current release is SPICE3f5 [65]. Since the first versions of SPICE are implemented in Fortran, a Fortran-like circuit description is still remaining and most of the commercial SPICE versions are compatible with the Berkeley syntax, although extended with vendor specific features which limit the compatibility to concurrent products. Important commercial, industry standard simulators are offered by *Cadence Design Systems* (PSPICE, Spectre, UltraSim), *Synopsys* (HSPICE), *Silvaco* (SmartSpice) and *Mentor Graphics* (Eldo) and are often integrated into a comprehensive mixed signal design framework. Additionally, free SPICE versions (open source) are available for non-commercial use, namely NGSPICE, tclSPICE and SPICE3f5.

4.2 Operation Principle of Analog Circuit Simulators

nodal analysis of circuits

Circuit simulators are using nodal analysis to predict the behavior of circuits. Thereby, all nodes of the circuit are enumerated and the voltage of each node is stored in a variable. The first step is to determine initial values for each node. Those nodes, which are directly connected to a voltage source, are initialized with the voltages provided by the respective voltage source. For the remaining nodes, the voltages are found by solving the corresponding simultaneous equations, obtained from Kirchhoff's Laws. This procedure is iterated until the solutions for all nodes converge. Different algorithms (mesh analysis, current branch method) are used to partition the circuit analysis problem into multiple simpler problems of calculating an operating point of a linear circuit which can be described by linear simultaneous equations. Once the operating point is calculated, nonlinear components—in this case transistors—can be linearized by replacing their large signal model with a linearized small signal model, suitable for the determined operating region.

partitioning large circuits

calculation of the operating point and linearization

4.2.1 CMOS Device Modeling

Before a circuit can be successfully integrated in CMOS technology, it is necessary to verify its functionality in simulation. The quality of the prediction of the circuits behavior depends on the accuracy, with which the model of the target technology describes the physical substrate. Models of current technologies achieve a high degree of reliability and the prediction of simulation is, even for circuits consisting of thousands of transistors, within a feasible range of measuring. Possible representations of CMOS models are parameterized mathematical equations, behavioral descriptions or lookup tables. There are more than 60 different SPICE models available, referred to as SPICE level, which cover different technologies and levels of complexity. Examples for models of increasing accuracy and therefore high complexity are SPICE level 1, SPICE level 3 and BSIM. While the level 1 model is quite simple, the level 3 model already includes many parasitic effects that have to be considered in short-channel technologies down to about $0.8\,\mu$m as well as subthreshold conduction. Finally, the BSIM3v3.2.4 model has become the current standard for computer simulations of deep sub-micron processes down to $0.25\,\mu$m and accurately describes physical effects of small structures. For device sizes down to

predicting a circuits behavior with a physical model

different SPICE levels for different technologies and levels of complexity

industry standard: the BSIM models

0.1 μm some physical mechanisms need to be characterized even better and consequently BSIM4v4 has been developed, which provides a geometry-dependent parasitics model and, for instance, considers carrier quantization effects. A complete description of the SPICE models can be found in [1].

An example of a simple CMOS model is given in part I in chapter 1, section 1.1 and 1.3. The equations of the BSIM models are far more complex, due to the fact that all relevant parasitic effects are included. As a consequence of this, a total of 93 parameters are necessary to describe a PMOS or NMOS transistor with the BSIM3v2 model. In addition to the device parameters, capacitances and resistances of metal lines, vias and metal-poly-oxide contact areas are included in the technology model in order to perform a parasitic extraction of the whole layout. The process parameter files can be obtained from the manufacturer of the target technology, whereas the model has to be provided by the respective circuit simulator. Unlike the actual process parameters, which may not be published, the models are most often public domain, for the reason that communication, simplifying technology and sharing productivity is desired. Thus, in order to characterize a circuit for a specific process, the correct SPICE level (model) has to be selected and the parameter file has to be included in the simulation.

CMOS parameters for different fabrication processes

4.2.2 The SPICE Netlist: Circuit Description and Simulation

Most of the current analog circuit simulators are compatible with the original SPICE syntax. Thus, the basic syntax of *Berkeley SPICE3f5* is chosen to describe the structure of an analog circuit netlist. Furthermore, the SPICE3f5 simulator is used as simulator-in-the-loop for the experiments in this thesis. The full SPICE netlist can be divided into four main parts: first, the header, where the voltage sources and inputs are defined. Second, the circuit components and subcircuits. Third, the setup and execution of circuit analysis and fourth, the CMOS model definition. An example netlist, which represents an operational amplifier and carries out an open-loop DC analysis is discussed in fig. 4.2.

the structure of a SPICE netlist

In addition to this, the SPICE syntax for CMOS transistors and example setups for DC, AC and transient simulations are described, which are mainly used for the experiments in this thesis. According SPICE netlist code is provided for all examples after the respective paragraph. For a full description of the SPICE functionality, the reader is referred to the SPICE manual [65] or the *Cadence* documentation [16].

description of the SPICE syntax

In case of transistors, the syntax is reduced to the relevant parameters, that are width and length. Additional parameters, e.g. the areas of drain and source diffusions, are determined by the respective device model. Transistors are labelled with an `m` followed by a unique device number. `source`, `drain`, `gate` and `bulk` take on node numbers, depending on their position within the circuit. The `device model` determines the simulation model (PMOS or NMOS) of the target technology, while `with` and `length` specify the gate/channel dimensions.

representation for a transistor in SPICE

```
mxxx source gate drain bulk device_model length width
```

A DC analysis can be performed by sweeping either one or two input voltages, hence,

DC analysis: setup and initial conditions

either one resulting transfer curve or a set of resulting transfer curves can be measured. The DC analysis is carried out with capacitors open and inductors shorted. First, the input voltage sources `voltage_source` and `voltage_source2` have to be defined. This is done by assigning two nodes and a voltage value, which represents the voltage difference between those two nodes, to the respective source. Most often, one of the nodes of a voltage source is set to gnd. Second, in the case of non-convergence, initial conditions, exclusively for the DC analysis, can be set for each node with the `.nodeset` statement. Finally, the `.dc` line defines the voltage sources to sweep, the sweep limits and the voltage increment, hence, the number of samples and, in case of two inputs, the number of curves.

```
voltage_source node1 node2 voltage
voltage_source2 node3 node4 voltage2

.nodeset voltage(node1)=value1 voltage(node2)=value2 ...

.dc voltage_source v_start v_stop v_incr [voltage_source2
v2_start v2_stop v2_incr]
```

transient analyses: setup and initial conditions

definition of input voltage patterns

Principally, the number of varying input voltage sources is not limited in case of the transient analysis, thus, this type of analysis offers the highest degree of freedom in creating custom input voltage patterns. Although several types of time dependant voltage and current sources (exponential, sinusoidal, pulse) are available, the piece-wise linear (`pwl`) source is considered as example. A set of time/voltage or time/current pairs, with increasing time, define the input voltage pattern for the `pwl` source by using linear interpolation on the input values. Analog to the `.nodeset` statement for the DC analysis, the `.ic` statement can be used to specify initial conditions exclusively for the transient analysis in case of convergence problems. Thereby, the initial conditions will be only considered, if the `uic` statement is added to the `.tran` line. Finally, the `.tran` line defines the setup for the transient analysis by setting start, stop and increment values for the time. Additionally, a time precision `time_prec` can be specified, in order to ensure a minimum precision for calculation.

```
voltage_source_pwl node1 node2 voltage pwl(time1 voltage1
[time2 voltage2 ...])

.ic voltage(node1)=value1 voltage(node2)=value2 ...

.tran time_step time_stop [time_start [time_prec]] [uic]
```

AC analysis: setup

If an AC analysis is desired, the keyword `ac` has to be added to at least one independent voltage/current source in combination with values for magnitude and phase of the AC input signal. In the given example, the independent source can also be used for a transient analysis with sinusoidal input. The `.ac` line specifies, whether the input frequency is varied in decades, octaves or linearly as well as the number of samples per interval and the frequency range.

```
voltage_source_sin node1 node2 voltage ac magnitude phase
sin(v0 vampl freq delay damping)

.ac dec (oct,lin) no_samples freq_start freq_stop
```

4.2.3 Floating Nodes and Initial Conditions

reasons for simulation failure

The extraction of a given circuit into a valid SPICE netlist is a crucial precondition for carrying out a successful simulation. However, this is not always possible for evolved circuits. If the circuit, and therefore the netlist, contains floating nodes—i.e. nodes which are neither connected to any other component, nor to any independent voltage/current source—or the initial state of a node cannot be determined, e.g. if two gates are connected to nothing but each other, the circuit analysis will not converge, i.e. the simulation will fail. Usually, floating nodes are avoided by the designer, since circuits, which contain floating nodes, are generally meaningless. Nevertheless, in the case of automated circuit synthesis, it is necessary to be able to successfully simulate even such invalid circuits in the following cases: first, for circuit synthesis by means of evolutionary algorithms where it is desired not to abandon partly good solutions, i.e. a complete circuit would be lost due to only one faulty subcircuit. Second, for circuits that have been developed on a complex configurable hardware and shall be verified by using a simpler simulation model, which lacks some specific substrate properties and therefore possibly fails. The latter case is of particular importance to this thesis, due to the evolution of transferrable circuits is tackled.

avoiding ambiguous states by defining initial conditions

The initial conditions are usually calculated or approximated by the simulator and are necessary for solving the simultaneous equations of the respective model. If the simulation does not converge due to nodes in undefined or ambiguous states, initial conditions for each node (voltage or current value) can be included into the netlist of the respective circuit or subcircuit by adding a `.nodeset` line for DC analysis and a `.ic` line for transient analysis:

```
.nodeset v(node0)=value0 v(node1)=value1 v(node2)=value2
...
.nodeset i(node0)=value3 i(node1)=value4 i(node2)=value5
...

.ic v(node0)=value6 v(node1)=value7 v(node2)=value8 ...
.ic i(node0)=value9 i(node1)=value10 i(node2)=value11 ...
```

overcoming the problem of floating nodes

Contrary to that, it is not as obvious how to handle floating nodes, since usually the designer would have to connect those nodes in order to carry out a successful simulation. Basically, there are three possibilities to solve this: first, the simulation model has to be more complex, i.e.—in case of the FPTA—all switches and configuration circuitry, hence, all parasitic effects have to be considered for simulation. This often solves the problem of floating nodes, due to the fact that those nodes are connected to structures on the chip, which were previously not included in the simplified netlist. Second, although a node is connected on the chip, it is not intended to include the configuration circuitry in the netlist. Most often, the target circuitry is a multiplexer or a closed switch and therefore, such floating nodes can be connected to a reverse-biased drain-bulk diode:

replacing configuration circuitry with drain-bulk diodes

```
mxx floating_node 0 0 0 nmos w=small l=large
```

considering ASIC integration of synthesized circuits

Consequently, the simulation of the circuit succeeds and, albeit simplified, the parasitic influence of e.g. a closed switch is modelled. Furthermore, from a designers point of view, it is always a good idea to use structures, that can be easily realized on a chip, because it should be possible to integrate automatically synthesized circuits on an ASIC. An example of using an NMOS transistor as a reverse-biased bulk diode is shown above. Third, since, again, the worst case of a really floating node can only be solved by connecting it to at least one minimal conductance, the only possibility in this case is to make a good guess of where to connect it. Consequently, it is recommended to inherently avoid the third type of floating nodes, regardless of which technology is used for circuit synthesis: either configurable hardware or a software simulation[3].

4.3 Simulator in the Loop: Berkeley SPICE3f5, NGSPICE

freely available circuit simulators

The *Berkeley SPICE3f5* [65] and *NGSPICE* [2] are chosen for practical reasons: first, both simulators are available as open source or can be used under the *GNU Public License*. Second, both simulators are freely available for a great variety of operating systems (in this case *Linux*). Finally, state-of-the-art BSIM3v3 CMOS device models (SPICE level

[3] There are some exceptional cases in the area of analog circuit design, for which it possibly makes sense to leave a node unconnected; e.g. if the goal is to shield components from the noise of surrounding potentials.

8,49,53 depending on the simulator) are included in both packages and therefore, components of the FPTA can be accurately simulated by using the corresponding AMS $0.6\mu m$ process parameters, that are describing the technology the FPTA is designed in.

Unfortunately, to the author's knowledge, there is yet neither a C++ SPICE library, nor one for any other programming language, which can be directly linked to the hardware evolution framework application, which is written in C++. As a consequence, SPICE has to be run as an external process(es) and ASCII [4] files are used for data exchange with the evolution software. In simpler words, the evolution software generates netlists and runs external SPICE processes which use those netlists. After a simulation is completed, SPICE saves the simulation results to an ASCII file, which is read by the evolution software. Since the simulation time usually dominates the time overhead of file communication, it is not a great slow-down to use ASCII files for data transfer.

ASCII files are used for data exchange

NGSPICE has to be mentioned as an alternative to *Berkeley SPICE3f5*, due to the fact that it is fully compatible with *Berkeley SPICE3f5* and additionally includes the latest BSIM4v4 CMOS device models. Furthermore, for the reason of licensing conflicts, rather *NGSPICE* instead of *Berkeley SPICE3f5* is innately included in current LINUX distributions.

4.4 Extracting Netlists from FPTA Results

Three different levels of complexity for the conversion of the FPTA circuit representation to a SPICE netlist are defined in this section, which must not be mixed up with the SPICE level of the CMOS model: first, on the simplest level, the active programmable transistors are considered as directly connected single transistors, which is useful for quick checks and for investigating the operation principle of the evolved circuit. The second complexity level takes the mean on-resistances of the active transmission gates into account. Those transmission gates are, on the one hand, connected to the transistor terminals and, on the other hand, used for realizing the routing within the FPTA cells. Finally, the simulation on the third and highest complexity level includes both, the transistors that represent the actual configurable transistor and the transistors that are needed for configuration and routing. Thus, the latter simulation level reproduces the measurement on the FPTA with the highest precision. Yet, the disadvantage of more complex simulations is the significantly increasing simulation time.

three levels of complexity for netlist extraction

It has to be remarked that the highest possible accuracy of the simulation would be achieved by using the back-annotated layout with full parasitic extraction and the simulator from the design software. This is not done in this thesis for the reason that, on the one hand, the accuracy of the level 3 simulations is sufficient for the current experiments and, on the other hand, such simulations are complicated and time consuming and thus, go beyond the scope of this work.

[4] A file format for text: american standard code for information interchange (ASCII).

4.4.1 Level 1: Simulation with Plain Transistors

plain transistors

On the least complex simulation level, the active FPTA cells are considered as single transistors of given W/L ratios, hence, any parasitic effects of additional configuration circuitry are neglected. An example for such a plain netlist is created by algorithm 4.1.

Algorithm 4.1: This algorithm converts circuits from the FPTA to basic netlists, in which all active configurable transistors are considered as directly connected plain transistors.

```
find all transistors, that are connected to the active circuit
enumerate and create a list of those active transistors 0..K
for all active transistors ← 0 to K do
    for all transistor terminals ← 0 to 2 do          // source,drain,gate
        follow wires and find target node
        if target node has a node number then
            assign the target node number to the current terminal
        else if target node has no node number then
            create a new unique node number
            assign this node number to the current terminal
        end if
    end for
end for
detect input and output nodes of the circuit
create a netlist using the list of all active transistors
```

4.4.2 Level 2: Simulation Including Resistances of Switches

transmission gates modeled as resistors

The medium complexity level takes the mean on-resistances of the active transmission gates into account. Transmission gates are used for connecting the transistor terminals to one of the cell's four outside connections (N,S,W,E) and for directly connecting any of those outside connections, which provides the routing capabilities. The according algorithm 4.2, that creates this kind of netlist, is listed in the following. The values for the resistances between the transistor terminals and the outside connections are taken from the thesis of Jörg Langeheine ([49]), who designed the FPTA.

4.4.3 Level 3: Simulation Including the Whole Configuration Circuitry

transistor matrix and transmission gates are included in simulation

Level 3 represents the highest level of netlist complexity. Consequently, both, the transistors that represent the actual configurable transistor and the transistors that are needed for configuration and routing are included in the netlist. This is realized by creating subcircuits for the FPTA's PMOS and NMOS cells, that contain the configuration circuitry as well as the transistor matrix which represents the configurable transistor. Those subcircuits feature a great number of inputs/outputs, namely the four outside connections (N,S,W,E), 40 nodes for the configuration of the switches that select the W/L ratio, 36 nodes for the connection of the transistor terminals to one of the four outside connections and 12 nodes for the configuration of the routing. The total number of those nodes does not correspond to the number of configuration bits for one cell, but to twice the number of

Algorithm 4.2: This algorithm converts circuits from the FPTA to netlists of medium complexity, in which the resistances of all active switches are included. N,S,W and E (north, south, west and east) are the external nodes of the FPTA cells.

```
create a list of all FPTA cells used 0..K
n = 16 transistor cells per row
for all FPTA cells used ← 0 to K do
    assign node numbers to N S W E,
    depending on the x/y position within the transistor array
    N = y · (2n + 1) + x + 1;
    S = N + 2n + 1;
    W = N + n;
    E = N + n + 1;
    connect the gate to its target (NSWE) via 2310 Ω
    connect source and drain to its targets (NSWE) via 330 Ω
    insert routing by connecting according nodes (NSWE) via 330 Ω
    create a subcircuit netlist entry for the current FPTA cell
end for
detect input and output nodes of the circuit
create a netlist using the subcircuit entries of all FPTA cells used
```

actual switches—transmission gates need two voltages, either vdd-gnd of gnd-vdd—that have to be opened or closed for realizing a certain configuration. Consequently, once the subcircuits for the PMOS and the NMOS cell are available, only the configurations for their inputs have to be created for the netlist, according to the desired configuration.

4.5 Cadence Design Framework

The Cadence software is a comprehensive design framework, which provides industry standard design and verification tools for custom analog and digital VLSI design. Besides, the FPTA chip has been designed with the Cadence design framework and the process setup for AMS 0.6 μm. Thus, the Cadence software is used to validate the simulation results, obtained from the SPICE simulator for the experiments in this thesis. It has to be remarked that simulations are exclusively performed with schematics, i.e. no layouts are generated from the resulting circuits of the presented experiments. Nevertheless, if a schematic for a new design was found, it could be adapted to a certain technology and realized on a chip in the next step.

industry standard design tools for VLSI design

4.5.1 Circuit Simulation in Cadence

Cadence is able to work with a great variety of circuit simulators, e.g. *cdsSPICE*, *HSPICE*, *Spectre* or *UltraSim*, to name just some important ones. It is possible to create flexible simulation test benches as a combination of schematics, HDL and netlists. Principally, it is possible to consider all environmental conditions—such as noise, temperature and worst/best case design corners—that are covered by the process model in simulation. Device matching effects on the die are considered with a *Monte Carlo* analysis. Further-

environmental influence

simulation of back-annotated schematics

more, simulations of back-annotated schematics can be carried out and therefore parasitic effects like crosstalk, parasitic layer and wire capacities can be included in the simulation. Thereby, the term 'back-annotated schematic' denotes a schematic in which, on the one hand, each component is identified with its respective representation in the layout and, on the other hand, layout dependent parasitic effects are added to the schematic. The layout has to successfully pass three tests before a valid back-annotation is achieved: First, a DRC has to be performed to ensure that no geometrical constraints of the fabrication process are violated. Second, a successful LVS check guarantees that all devices of the schematic are present in the layout and vice versa. Third, all relevant electrical parameters are calculated from the LPE. Those extensive simulation and verification methods target at maximizing the yield of fully operational ASICs.

design rule check and layout parameter extraction

4.5.2 Automatic Schematic Generation Using SKILL

a scripting language for Cadence

SKILL[5] is a very powerful scripting language, which is included in the Cadence design framework and allow to automate any functionality of the Cadence software. The manual is part of the *Cadence Open Book* [15]. Thus, it is possible to implement custom macros that generate e.g. a parameterized layout block which is frequently needed or certain schematic building blocks. In this thesis, the SKILL language is used for automatically generating schematics for Cadence from circuits that are evolved on the FPTA. Due to the fact that the visualization of the resulting circuits, which is provided by the circuit editor of the evolution software (fig. 4.1), is rather based on the transistor array architecture than on readability. Consequently, it is necessary to reorganize the evolved circuits in order to improve their readability and understandability, although this is not an easy task, since the genetic algorithm often produces unconventional circuits. In this case, the reorganization of the schematic is carried out in four steps: First, a simple netlist is generated from the FPTA circuit where all nodes and transistor terminals are enumerated. Second, all transistors are placed on a 2-dimensional grid, with the node number of the gate as x-coordinate and the node number of the source as y-coordinate. Third, all gates can be connected with straight vertical wires and all sources can be connected with straight horizontal wires. Fourth, the remaining drain terminals are connected to their respective vertical or horizontal target net. As can be seen from fig. 4.1, the resulting schematic can be more easily analyzed. Despite this, more intelligible schematics are manually drawn for particular example results, which are presented in chapter 7. The schematic-creating SKILL script is automatically generated by the evolution software. Example SKILL commands are listed below. Those commands are combined in a script file with the extention `*.il`, which then provides the new functionality encapsulated in macros:

automatic schematic generation

[5] "The name was originally an initialism for silicon compiler interface language (SCIL), pronounced 'SKIL,' which then morphed into 'SKILL,' a plain English word that was easier for everyone to remember." From *John Gianni*, usenet: comp.cad.cadence.

```
procedure(CreateSchematic()

/* create a new schematic view */
cv=dbOpenCellViewByType("FPTA_EVO" "Test_exp" "schematic"
"schematic" "w");
...
)  /* end of procedure */
```

implementing custom macros

New functionality is encapsulated in procedures, which can be executed within the *Cadence* software. In this case the new macro is named `CreateSchematic()` and the first command `dbOpenCellViewByType(...)` creates a new, empty schematic view named `Test_exp` within the *Cadence* design library `FPTA_EVO`.

```
netID=dbOpenCellViewByType("analogLib" "vdd" "symbol");
inst=dbCreateInst(cv netID "Vdd" X:Y "rotation");
netID=dbOpenCellViewByType("analogLib" "gnd" "symbol");
inst=dbCreateInst(cv netID "Gnd" X:Y "rotation");

cmosID=dbOpenCellViewByType("PRIMLIB" "nmos" "symbol"
"" "r");
inst = schCreateInst( cv cmosID "instance_name" X:Y
"rotation" );
dbSetq(inst 12u w);
dbSetq(inst 1u l);
```

inserting components

The above examples show how to insert new components (`vdd`,`gnd`,`nmos`) into the active schematic. The `dbCreateInst` and `schCreateInst` insert, place and add labels to the respective component and create a handle `inst` which is used to change component parameters with the command `dbSetq`, e.g. a transistor's W and L. Thereby, a unique `instance_name`, the `X:Y` position and the `rotation`=R0(0°), R1(90°), R2(180°), R3(270°) must be specified.

```
wireID=schCreateWire(cv "route" "full" list(X0:Y0 X1:Y1)
0.0625 0.0625 0.0 );
wireID=schCreateWire(cv "route" "flight" list(X0:Y0 X1:Y1)
0.0625 0.0625 0.0 );
wireID=schCreateWire(cv "draw" "direct" list(X0:Y0 X1:Y1)
0.0625 0.0625 0.0 );
```

interconnecting components

There are three different possibilities for adding a wire from `X0:Y0` to `X1:Y1` to the schematic: First, the options `route`,`full` advise *Cadence* to carry out auto-routing for

the new wire. Second, the options `route,flight` use a direct rubber band connection, which is fully functional for simulation but has to be manually routed in order to obtain a plain schematic. Third, a fixed straight connection can be forced by using the options `draw,direct`.

```
pinCVID = dbOpenCellViewByType( "basic" "ipin" "symbol" ""
"r" );
pinId = schCreatePin( cv pinCVID "In0" "input" X:Y
"rotation" );
pinCVID = dbOpenCellViewByType( "basic" "opin" "symbol" ""
"r" );
pinId = schCreatePin( cv pinCVID "Out0" "output" X:Y
"rotation" );
pinCVID = dbOpenCellViewByType( "basic" "iopin" "symbol"
"" "r" );
pinId = schCreatePin( cv pinCVID "Out0" "in/output" X:Y
"rotation" );
```

creating IO pins

Finally, the circuit has to be connected to other subcircuits or the outside world. This is achieved by inserting input/output pins into the schematic and by connecting them to the respective nodes. The command `schHiCheckAndSave()` has to be invoked at the end of the procedure in order to verify the resulting schematic and to save all changes.

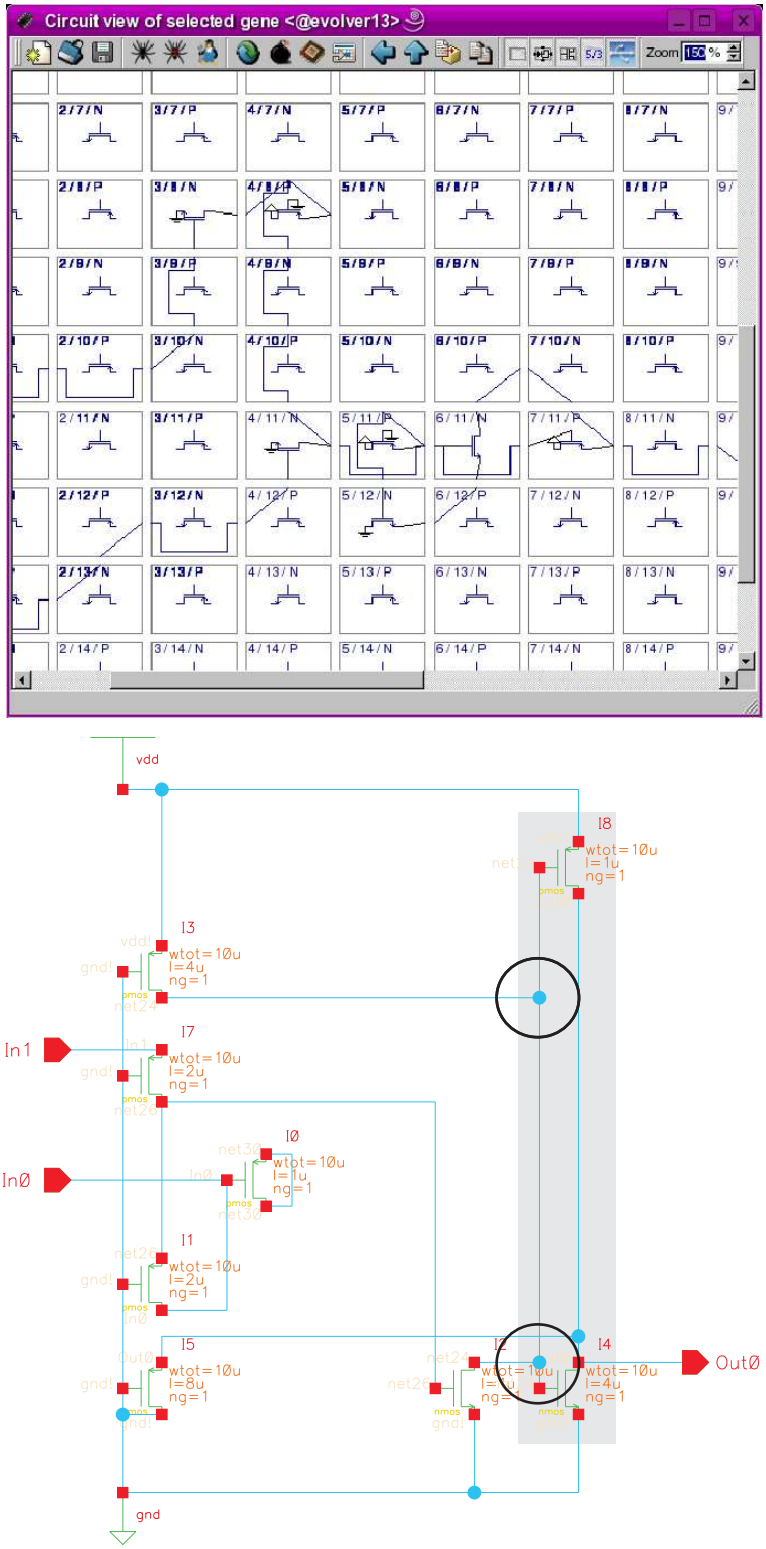


Figure 4.1: *Top: A screenshot of the circuit editor of the evolution software is juxtapositioned to (bottom:) a schematic which is automatically generated by an according SKILL script in Cadence. As can be seen by comparing the two figures, the readability of the schematic on the right hand side is greatly improved. This becomes even more important for more complex circuits, which consist of a greater number of transistors.*

```
* Title:OpAmp open-loop DC
vcc 4 0 5v
vin0 1 0 0.00v
vin1 2 0 0.00v
*vout0 3
```

```
x0 1 2 3 4 OpAmp

.subckt OpAmp 5 4 8 10
m1 3 4 6 10 modp l=1u w=2u
m2 3 5 7 10 modp l=1u w=2u
m3 6 6 0 0 modn l=1u w=4u
m4 7 6 0 0 modn l=1u w=4u
m5 10 2 3 10 modp l=1u w=3u
m6 8 7 0 0 modn l=1u w=10u
m7 10 2 8 10 modp l=1u w=5u
m8 10 2 2 10 modp l=1u w=3u
m9 8 10 9 0 modn l=1u w=10u
m10 8 0 9 10 modp l=1u w=10u
r1 2 0 100k
c1 7 9 6p
.ends OpAmp
```

```
.dc vin0 0 5 .02 vin1 1 4 .25

.save v(1) v(2) v(3)
.end
```

```
.model nmos modn level=3
.model pmos modp level=3
```

Voltage sources and circuit inputs are described in the header by specifying two nodes and the voltage value between them. By definition, the value of each voltage source is fixed. The ground potential (0 V) is always represented by the node number 0 and needs no further specification, whereas `vdd` (or `vcc`) are defined as voltage sources. Comments are preceded by an asterisk.

In this example, the OP is realized within a subcircuit, which, once defined, behaves like a component and can be accessed through `x0`. The node numbers defined in the `x0` line are global, whereas all node numbers within the sub-circuit are local, except for the ground net. Hence, the enumeration of the nodes of the subcircuit is independent from the top-level circuit. It is possible to insert components either into a subcircuit, or directly into the netlist. Furthermore, the command `.include filename.cir` allows to create hierarchical netlists in combination with the possibility of defining subcircuits.

The `.dc` command carries out a DC simulation of the circuit. In this case a set of 13 curves with `vin1`$= 1, 1.25, 1.5 \ldots 4\,\mathrm{V}$ is measured by sweeping `vin0`$= 0 \ldots 5\,\mathrm{V}$ for each `vin1`. The voltages for the global nodes 1, 2 and 3 are saved to the result file.

Finally, it is essential to define the CMOS device models that are used for simulation. In this example, a basic model (SPICE level 3) is used.

Figure 4.2: *Example SPICE netlist, which represents an operational amplifier and carries out an open-loop DC analysis.*

Chapter 5

Evolution Software Environment

This chapter introduces the evolution software environment, denoted as EvoPoly, which has been developed in C++, in order to perform the experiments in this thesis. The operation principle and the implementation of the EA, the genetic representation of the circuit and the testmode based experimental setup are described. Since all components of the software environment are implemented in a modular manner, it is possible to easily extend the evolution system with customized algorithms and genome representations as well as to compose flexible setups. Besides, the measurement (evaluation) of the genomes and the fitness calculation is entirely independent from the EA, thereby offering the possibility to target different evolution platforms, namely configurable hardware and circuit simulators, by implementing according interfaces. Hence, a comprehensive and customizable evolution software framework is achieved, which lends itself for being developed by numerous programmers in parallel, due to the modularity of the implementation. Finally, the user interface is briefly described.

For the experiments in this thesis, the original *darkGAQT* software, described in [49] has been further developed and included into the *HANNEE* software framework, described in [35]. Both software projects have been developed in the *Electronic Vision(s)* group at the Kirchhoff-Institute for Physics in Heidelberg. The EA framework has been developed during this thesis, whereas the hardware access modules are adapted from *darkGAQT* and the GUI is based on the *HANNEE* framework.

5.1 Operation Principle of the Modular Evolution Software Framework

evolutionary algorithm and hardware abstraction layer

The modular evolution software framework is implemented in C++ [39, 80] and consists of three main parts: the evolutionary algorithm engine, the measurement setup and the hardware abstraction layer of the evolution substrate. All three parts are implemented as independent modules which are communicating through generic interfaces. Hence, it is possible to easily operate different genetic algorithms and to extend and customize the experimental setup. Additionally, the developed algorithms and genomes are not exclusively bound to one particular evolvable substrate, since the substrate itself can be replaced with another one. In this thesis, two substrates are used for the evolution of analog circuits, namely the FPTA, which is described in chapter 3, and a SPICE simulator, which is described in chapter 4. Due to the fact that this work is focused on the evolution of analog circuits on FPTA architectures, specialized genome classes are designed to implement circuit components and FPTA cells. Consequently, the derived genome and substrate classes do depend on each other in practice, although the base classes are independent from each other.

targeting different substrates for evolution

5.2 The Algorithmic Side of the Evolution Software

the GALib

The implementation of the evolutionary algorithm engine is based on the GALib of Matthew Wall [95] and entirely independent from the measurement and the hardware modules. Hence, the developed algorithms can be easily adapted to any other optimization problem, which is a nice example of the advantages of object oriented programming. Since evolutionary algorithms operate on at least one population of possible solutions, represented by a set of genomes resp. individuals, the algorithm base class already provides an empty population. The evolution is carried out stepwise by applying variation operators, namely mutation and crossover, to the current genomes. Subsequently, the arisen new candidate solutions are evaluated and finally the fittest individuals are carried to the next generation due to a given selection scheme. The variation operators represent the interface between the evolutionary algorithm and the evolution substrate and are specialized to certain kinds of target substrates, although the evolutionary algorithm itself does not need to know their respective implementation. All operators of the evolutionary algorithm, depicted in figure 5.1, are implemented as independent modules which can be linked to the base algorithm. Thus, the algorithm can be customized by assembling the desired operators and modules. Furthermore, the population, the genomes and the evolutionary algorithm itself can be easily extended or modified by deriving new classes, while the basic functionality is maintained.

modularized implementation

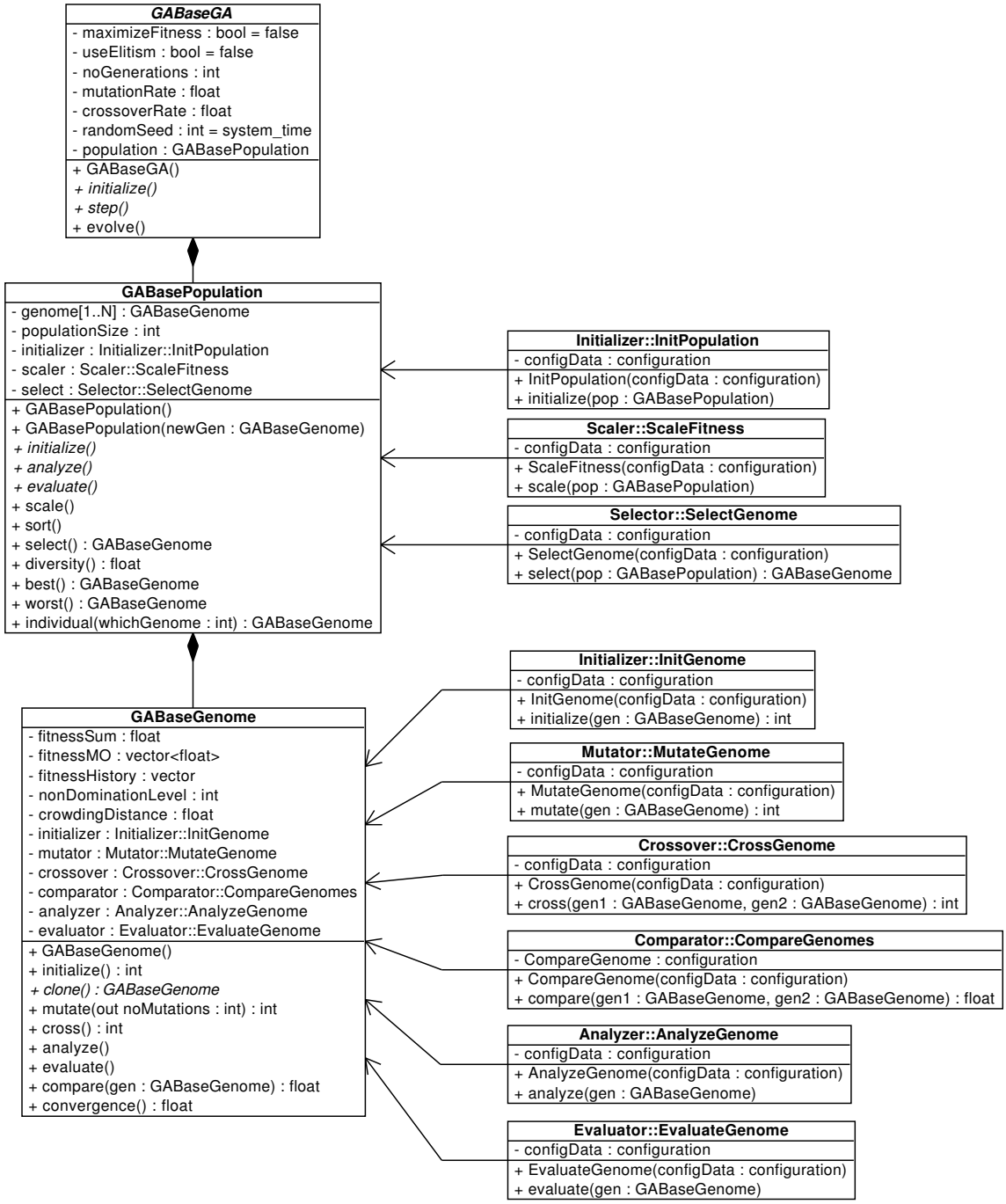


Figure 5.1: *The diagram of the fundamental classes of the evolutionary algorithm is depicted above. As can be seen from the class diagram, the evolutionary algorithm contains one or more populations, which, for their part, consist of a number of genomes. Both, the populations and the genomes, include references to evolutionary operators, which are implemented as independent methods respectively. Hence, the evolutionary operator can be freely combined and can even be swapped at runtime. The base classes of the evolutionary algorithm, the population and the genome already provide methods, that are necessary to operate the algorithm. Consequently, it is easy to derive new evolutionary algorithms by simply overload the initialize() and the step() methods of the* `GABaseGA`*. Additionally, a custom genome has to be derived from* `GABaseGenome` *and according genetic operators have to be implemented. The functionality of the actual evolution process and the book-keeping are provided by the base classes. This structure provides a flexible, modular evolutionary algorithm library.*

5.2.1 Class Structure of the Evolutionary Algorithm

initialization and evolutionary step

As can be seen from the unified modelling language (UML) diagram in figure 5.1, - `GABasePopulation`, `GABaseGA` and `GABaseGenome` are base classes, which provide fundamental methods and data structures for running the evolutionary algorithm. The `GABaseGA` class contains a population, global configuration parameters (variation probabilities, no. of generations, random seed) and the methods that carry out the evolution run. Thereby, important methods are the evolution *step()*, whereas the method *evolve()* is provided for convenience and merely repeatedly calls *step()*, and the *initialize()* method which successively calls the initializer of each population. Further, global settings, as for example whether the fitness is to be maximized or minimized and whether to enable elitism or not, are made in the `GABaseGA` class. Note that the EA is referred to as genetic algorithm (GA) in the names of classes and methods.

population management

The `GABasePopulation` class consists of an array of genomes and provides methods for managing the population. It is for example possible to sort the individuals according to their fitness or to directly access those individuals with the best or worst performance. Three main operators are linked to the population: first, the initializer, which initializes the population and, if necessary, successively carries out the local initializer of each genome. Second, the scaling scheme, which is not mandatory, but can be used to post-process the fitness score before ranking the individuals as e.g. proposed in [27]. Third, the selection scheme, which defines the rules for selecting and abandoning individuals. Additionally, the *diversity()* method, which uses the *compare()* function of the `GABaseGenome` class to calculate a value, which represents a measure for the diversity of the genomes within the current population.

manipulating and evaluating the individuals

Finally, the `GABaseGenome` class is the base class for an individual. Data, which is specific for each genome, is stored in this class and respective genetic operators as well as methods for analyzing and evaluating the genome are linked to this class. Thereby, the variation operators are carried out by the methods *mutate()* and *cross()*, while the methods *compare()*, *analyze()* and *evaluate()* are performing the measurement and the evaluation of the genome. As the initialization process is carried out hierarchically, the *initialize()* method performs the initial setup of the genome. Each individual is assigned a vector of fitness values (*fitnessMO*), depending on the number of objectives in which the individual is to be improved. For further calculation, the *fitnessSum* and the *fitnessHistory* of the past N generations are also recorded in the `GABaseGenome` class. Additionally,—aiming to multi-objective optimization experiments—the value for the *nonDominationLevel* and the *crowdingDistance* of the individual are stored. The latter values are more closely described in chapter 7, section 7.1.2. The `GABaseGenome` class does not yet contain any genetic information or coding, thus, it is mandatory to derive a specialized genome in order to be able to use it. Depending on the desired experiments, the genome has to be extended with a suitable data structure, which contains the genetic information.

5.2.2 Derivation of Custom Evolutionary Algorithms

customizing the evolutionary step

Starting with the base classes of the evolution framework shown in figure 5.1, it is relatively easy to derive new classes and to customize the EA: first, a new GA has to be derived from `GABaseGA`. As an example, the new GA is referred to as `GAExampleGA`.

operator	range (data type)	meaning of return value
initializer	$0 \ldots N$ (int)	number of generated components
scaler	$0 \ldots N$ (int)	error code / success=0
selector	& (GABaseGenome)	a reference to the selected individual
mutator	$0 \ldots N$ (int)	no. of performed mutations
crossover	$0 \ldots N$ (int)	error code / success=0
comparator	$0 \ldots 1$ (float)	measure for equality: 0=equal, 1=tot. different
analyzer	$0 \ldots N$ (int)	error code / success=0
evaluator	$0 \ldots N$ (float)	fitness values

Table 5.1: *The meanings of the return values of the genetic operators are described in the above table. Those values are used by the evolutionary algorithm for calculating statistics and for bookkeeping, and as such have to be properly implemented.*

In order to achieve this, it is merely necessary to (re-)implement (overload) the *initialize()* and the *step()* method of the base class. The *initialize()* function is responsible for properly initializing all populations and all of their individuals, while the *step()* function represents one evolutionary loop. Hence, the user has any freedom to design the course of the evolutionary algorithm. Second, a custom `GAExampleGenome` has to be derived from `GABaseGenome` and must be extended with a datastructure which contains the genetic information. Third, if necessary, the mapping from genotype to phenotype has either to be implemented as an additional method in the `GAExampleGenome` class, or can be sourced out to the *analyzer* operator. For this work, the mapping functions are added to the genome class, since, if different *analyzers* are used, it is advantageous to implement it in only one place. Contrary to that, if a variety of different genomes are analyzed with the same *analyzer*, it will make more sense to put the mapping method into this module. Finally, as can be seen from figure 5.1, appropriate operators have to be created, which are able to process the custom genomes. Unlike the algorithm and the genome, the population base class can be used without further specialization, although, if extra functionality is desired, `GABasePopulation` can be derived as well. An example application is given in algorithm A.2 in appendix A.

5.2.3 Implementation of Modular Genetic Operators

In addition to the customized evolutionary algorithm, it is necessary to implement variation operators, which are compatible with the genome used. That is, in case of a genome encoding an analog circuit, the initialization, mutation and crossover operators have to be designed for varying such circuits and need to know about the available, changeable elements. The operators are implemented in separate classes, which can be linked at runtime to either a population or a genome. Each of those classes has to implement a method with a predetermined name, e.g. the mutator has to provide the method *+mutate(gen:GABaseGenome):int*. The names of the access member function for all genetic operators can be seen from figure 5.1. As a consequence of this, it is possible to create a whole set of independent and reusable genetic operators, which can even be switched at runtime by just linking them to a specific genome. Apart from the latter constraints, all

switchable genetic operators

features of the C++ language can be used to derive or to extend the genetic operators.

Furthermore, the programmer has to assure that correct return values are passed back by the genetic operators, since they are needed by the EA for calculating statistics and for bookkeeping. The meanings of the respective return values are listed in table 5.1. For illustration, an example implementation for an initializer-mutator-crossover set is shown in algorithms A.3 and A.4 in appendix A. The selector, scaler and comparator classes provide the same interface—although the methods are *select(), scale()* and *compare()*—and therefore have to be implemented in the same manner.

5.3 Analysis and Evaluation of the Genomes

separation of analysis (substrate dependent measurement) and evaluation (fitness calculation)

Generally, in evolution experiments, the entire process of mapping the genome to the phenotype, performing any measurement, further analyzing and finally assigning an according fitness value, is referred to as evaluation. Despite this, especially in the case of analog circuit evolution, it makes sense to divide the latter process in two procedures: first, the mapping of the genome to the target substrate and the measurement of the configured substrate. Second, calculating the fitness value based on the measured results and the desired specification. By doing so, the substrate dependent part of evaluation remains separated from the pure fitness calculation, which should actually not depend on the given problem or target technology. Consequently, the first step is referred to as *analysis* of the individual and the second step is referred to as the *evaluation* of the individual for the remainder of this thesis.

test modes

The experiments are organized in test modes, that each represent one circuit analysis. One test mode contains the input and the target voltage pattern, the definition of the inputs and outputs, the type of analysis that shall be carried out and the information of which fitness function shall be used. Therefore it is possible to set up a great variety of complex experiments by combining desired test modes.

5.3.1 Class Structure of the Testmode-Based Experimental Setup

As can be seen from figure 5.2, the `MeasuringSetup` class, which is linked to the genomes, contains on the one hand a vector of `TestModeBase` classes, which represent the actual experiments and, on the other hand, the `CalculationBase` class, which is responsible for the fitness calculation. Thereby, the *analyze()* method of `GAGenomeBase` performs the configuration of the target substrate, the genotype-phenotype mapping and the measurement. The measured results, as well as the input and target patterns, are temporarily stored in a `MeasuringData` object. The latter result object can be accessed by the `GAGenomeBase`'s *evaluate()* method, which calculates separate fitness values for each test mode; it delivers a vector of fitness values for multi-objective optimization in addition to the sum of all fitness values. Besides, in the case of multi-objective optimization, one test-mode can deliver more than only one fitness, although the number of fitness values is constant during the course of the experiment. Additionally, the `MeasuringData` object also provides various IO methods, such as importing data from different file formats, e.g. SPICE, and exporting data to different file formats.

analyze

evaluate

multiple fitness values for multi-objective

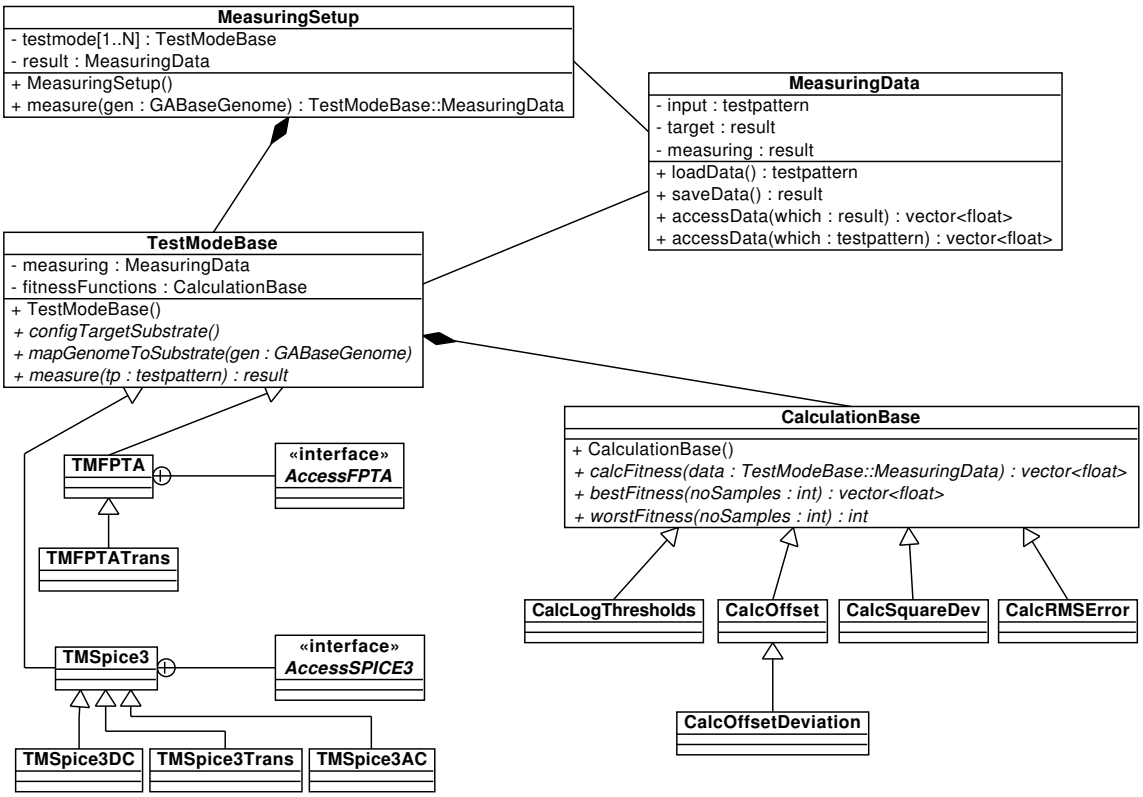


Figure 5.2: *The relevant classes for setting up an experiment are depicted. The* `TestModeBase` *class and the* `CalculationBase` *class are base classes, that need further specialization in order to use them. Thereby. the* `TMFPTA` *class, is extended with an interface to the FPTA chip and is dedicated to perform transient measurements on the hardware, whereas the* `TMSpice3` *provides an interface to the SPICE circuit simulator. Furthermore, the setup of the fitness function and the actual fitness calculation is carried out with the classes, derived from* `CalculationBase`*.*

5.3.2 Targeting Different Evolution Substrates

specializing test modes to substrates

Both, the `TestModeBase` class and the `CalculationBase` class are, as the names suggest, base classes, that need further specialization in order to use them. As depicted in figure 5.2, for the experiments in this thesis, two main branches are derived from `TestModeBase`: first, the `TMFPTA` class, which is extended with an interface to the FPTA chip and is dedicated to perform measurements on the hardware at different sampling frequencies. Second, the `TMSpice3` class, which provides an interface to Berkeley's SPICE3f5 circuit simulator. Due to the fact that, in addition to the transient analysis, other kinds of simulation (e.g. DC, AC, noise) can be carried out with the simulator, the `TMSpice3` class itself needs to be further specialized according to the desired type of analysis. Three example classes are given in figure 5.2, namely `TMSpice3DC` `TMSpice3Trans` and `TMSpice3AC`. Since the `TestModeBase` class can be specialized for other simulators or new hardware substrates in the future, it is easily possible to extend the current evolution system with additional evolution platforms.

5.3.3 Implementation of Fitness Functions and Fitness Calculation

A set of fitness functions, represented by `CalculationBase` objects, is assigned to each test-mode. Thereby, the `CalculationBase` class is the base class for any fitness function. Thus, a new fitness function can be added by deriving it from `CalculationBase` and simply overloading three methods, namely *bestFitness()*, *worstFitness()* and *calcFitness()*. The first two methods have to deliver the best and worst possible fitness value, respectively. Those values are used by the EA, e.g. for fitness scaling or in case the measurement fails. Again, this class structure allows for conveniently adding new fitness functions to the evolution system, while no changes have to be made in other code sections, since the interface classes remain the same.

5.4 Implementation and Customization of the Genotype

The genetic representation of the phenotype—in the case of this thesis an analog circuit—is equally important as accessing different substrates for hardware evolution. Hence, there are important demands on the genetic representation: first, it should describe an analog circuit on a high abstraction level, which allows for mapping it to different evolution substrates (e.g. FPTAs, simulators). Second, it should be possible to easily extend it and to add new components. Third, an expedient trade-off between memory consumption and convenience has to be found.

Basically, there are two slightly different ways of genetic encoding and manipulating the genetic encoding for EAs. In the first case, the genes are represented by a bit string, which is changed by the genetic operators without prior knowledge about the meaning of the respective bits. Contrary to that, in the second case, the variation operators are designed to rather change properties of the phenotype, which implies that, to a certain extend, the structure of the phenotype is known by those operators. It is possible to include either encoding in the presented framework, although the second approach is preferred for the experiments in this thesis.

5.4.1 Class Structure of the Genetic Representation of Analog Circuits

The evolutionary algorithm framework, introduced in section 5.2, provides the genome base class from which the actual genome, which is used for the presented experiments, is derived. Since the focus of this work is set to analog circuit evolution, the first step is to derive a `GACircuitGenome` class from `GABaseGenome`, as depicted in figure 5.3, and include a pointer to the experiment module (`MeasuringSetup`), which is described in section 5.3. Furthermore, the circuit genome is extended with the `CircuitStructure`, which is a flexible genetic representation, that contains the construction plan for the respective analog circuit. Again, the `CircuitStructure` contains a vector of `ComponentBase` objects and each of them represents one configurable component or building block of the whole circuit. As can be seen from figure 5.3, any desired circuit component can be derived from the `ComponentBase` class by using the given data structures together with according implementations of the genotype-phenotype mapping functions *getMySubstrateRepresentation()*. The `ComponentBase` class provides four different data structures for creating a custom, configurable circuit component: first,

customizing the circuit genome

mapping to different target substrates

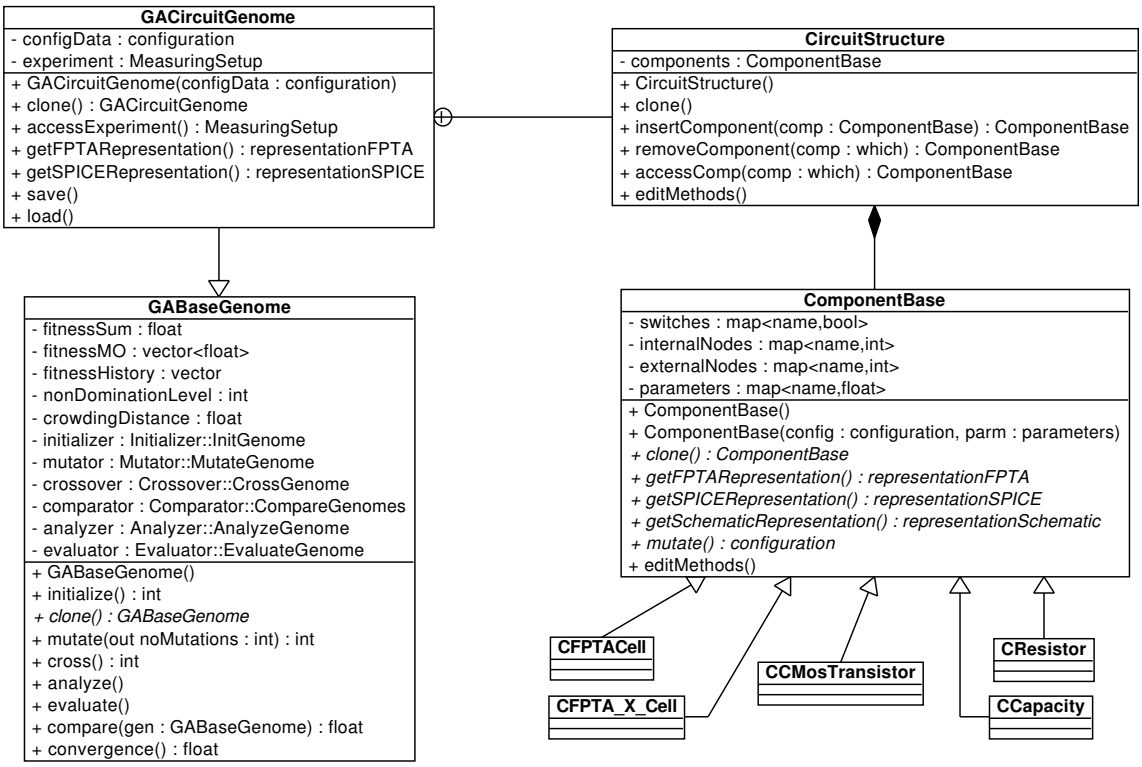


Figure 5.3: *The fundamental classes of the genome as well as their specialization for analog circuits are depicted. Thereby, the* `CircuitStructure`*, contains the actual analog circuit, which consists of various possible components, e.g.* `CFPTACell` *and* `CCMosTransistor`*. The latter classes are, on their part, derived from the* `ComponentBase` *class. The* `GACircuitGenome` *represents the interface to the EA library.*

a map, containing the external nodes, which define the connections to other components. Second, a map, that contains the internal nodes, which can be used to define the internal topology. Third, switches can be defined, in order to interconnect two nodes. Fourth, a map, for storing the parameter set which is necessary to characterize the respective component. All of those four data structures are realized as standard template library (STL) [40] maps, due to the fact that each node, switch and parameter can be assigned to an additional identifier, which improves the manageability of those items and the readability of the code. Since the derived components can be freely combined in the final circuit structure, this class system provides a flexible framework for encoding circuits of any kind and subsequently use them for evolution experiments.

5.4.2 Derivation of Custom Circuit Components

In order to derive a custom circuit component, its particular initial setup and configuration has to be done in the respective constructor, while the interpretation of the stored configuration has to be implemented in the *getFPTARepresentation()* and the *getSPICERepresentation()* functions. The specialization of a `ComponentBase` to a configurable

implementation of circuit components

transistor `CCMosTransistor` is chosen as an example and shown in algorithm A.1 in appendix A. Assuming the final circuit consists of several such transistor components, the information which ones are interconnected, is stored by assigning the same node number to one of the external nodes of both. Consequently, in the resulting netlist, those components will be connected. Since a transistor features three terminals (gate, source, drain), three corresponding internal nodes are created, which can be attached to one of the available external nodes, by simply assigning the respective identifier of the external nodes. In addition to the external nodes, the terminals can be connected to vdd, which is set to a predefined unique number, or gnd, which is always denoted by the node number 0. Two parameters W and L are created and initialized with sensible float values, which define the transistor size. Furthermore, the configurable cell can be switched from PMOS to NMOS, by changing the *CMOSTYPE* switch from *true* to *false*.

5.4.3 Modular Genetic Representation of Custom FPTA Architectures

arranging components in regular patterns

Keeping in mind that the genetic circuit representation shall be mapped to either the current FPTA, or even new FPTA architectures in the future, the first approach is to structure the representation according to such topologies. Presuming that FPTAs are generally composed of a regular pattern of configurable basic cells, custom architectures can be easily created by first, deriving the configurable basic cells from `ComponentBase`, add the desired amount of those cells to the circuit structure and configure the respective external nodes according to the desired topology. In principle, as long as a corresponding mapping function is provided, the genetic representation has not necessarily to be a one-to-one copy of the hardware, although this is the case for the representations in this thesis.

5.4.4 The Genetic Representation of the Current FPTA

The genetic representation of the current FPTA is close to the hardware and is implemented as a specialization of the modular representation of section 5.4.3. It consists of an array of data structures, that contain the configuration data for the cells: the x- and y-coordinates, 6 boolean variables for the routes and two float variables for the W/L ratio. Thus, the design of the basic cell equals the architecture of the configurable transistor cell of the chip, which is described in section 3.1.1. The greatest advantage of a direct genotype-phenotype mapping is that no complex place and route algorithm is needed for the configuration of the transistor array. Contrary to that, it might be disadvantageous to constrain the connectivity of possible circuits in advance, although the smaller search space, that results from the latter constraints, may lead to faster convergence of the EA.

5.5 Control Software and User Interface for the Evolution Software

darkGAQT software

The control software is basically the superstructure of the EA library, the genetic representation and the analysis and evaluation classes, which provides the user interfaces and configures the library modules. For the experiments in this thesis, the original *darkGAQT* software, described in [49] has been further developed and included into the *HANNEE*

software framework, described in [35]. Both software projects are developed in the *Electronic Vision(s)* group at the Kirchhoff-Institute for Physics in Heidelberg. The reasons for adapting the original software to the *HANNEE* framework are: first, the *HANNEE* framework provides a factory system for easily creating modular user interfaces for configuration. Second, the software automatically stores its current state and data in an extensible markup language (XML) configuration file, from which it can be fully restored. Third, it is possible to run the program in command line mode, which is useful for running experiments on remote computers, which has exhaustively been used for the presented experiments. Fourth, the *HANNEE* software provides so called *observer* classes, which make it convenient to display any desired data during the experiment by automatically creating the according plot windows. Finally, it is a great advantage to share implemented functionality with others. An example screenshot of the user interface of the evolution software is depicted in figure 5.4.

HANNEE software framework

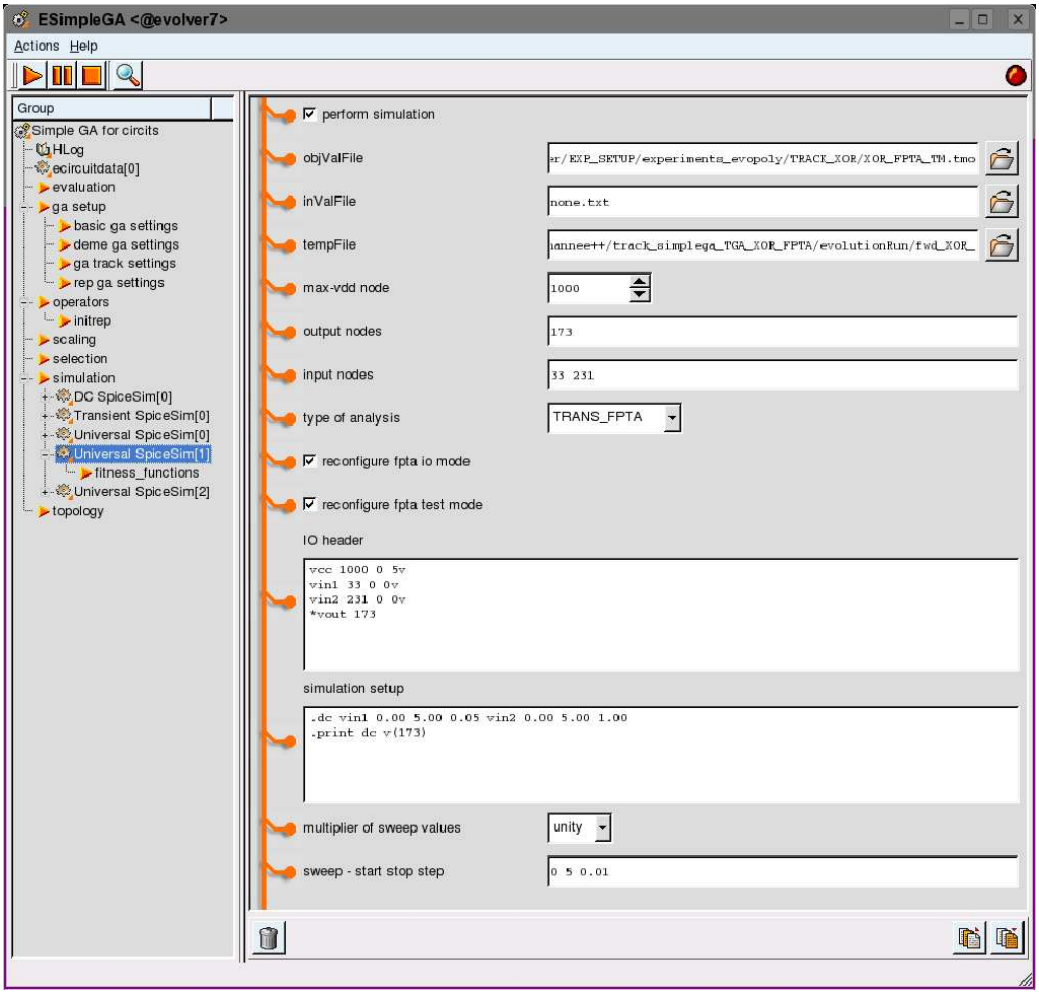


Figure 5.4: *A screenshot of the GUI of the evolution software.*

Part III

Experiments and Results

Chapter 6

Evolution of Transferable Circuits on the FPTA

The aim of this chapter is to develop an evolutionary algorithm, which is able to synthesize circuits on the FPTA, which are transferable to other technologies and can therefore be validated with a simulator outside the chip. In order to achieve this, new variation operators are developed, which make it at least technically possible to derive and reuse new concepts of electronic design from the evolved circuits. It is shown that the developed algorithm performs equally well as a straight forward implementation of the EA (Basic GA) in finding good solutions for logic gates and for comparators. The newly developed algorithm is referred to as the Turtle GA throughout the remainder of this thesis. Furthermore, the circuits, which are evolved with the Turtle GA, are extracted into SPICE netlists and the performance of the on-chip measuring is compared to the simulation results. In addition to this, the circuits are measured on different FPTA substrates, in order to investigate to what extent the circuits are bound to the chip, on which they are evolved. For all experiments, basic manually made designs, taken from [6, 90], are realized on the configurable transistor array and used as references for the evolved circuits. Finally, schematics are automatically generated from the best solutions, using the algorithm from chapter 4, section 4.5.2, and it is shown that in the case of some circuits with a good performance, it is possible to understand how they actually work.

6.1 Development of the Evolutionary Algorithm

off-chip circuit validation

The aim is to develop an EA, which is able to synthesize circuits on an FPTA chip, which are transferable to other technologies. In this case, those other technologies are in the first place either other FPTA substrates or a SPICE simulator. If it is possible to validate circuits, which are evolved on the transistor array, with a simulator outside the chip, it will be at least technically possible to derive and reuse new concepts of electronic design, which are possibly found by means of evolution. Additionally, the synthesized circuits can be more accurately analyzed with a simulator. This is helpful, since most evolved circuits are difficult to unterstand. To achieve this, an algorithm with new variation operators has been developed: the *Turtle GA*. It is desired that the new algorithm performs equally well than the *Basic GA*, which has been successfully applied in previous experiments [49, 55]. Both algorithms are used for the evolution of logic gates and comparators, in order to assess and to compare their performace. Additionally, statistical performance tests are carried out with all variation operators. The variation operators of both algorithms are designed to operate with the genetic representation of the FPTA, that is introduced in section 5.4.3. This representation is used for all experiments in this chapter. Tournament selection is used as selection scheme and in order to preserve arising good solutions, elitism is applied to the two best parents by replacing them unchanged with the two worst individuals of the offspring population. The implementation of the evolutionary step and the selection scheme is given in figure 6.1.

new genetic operators

tournament selection and elitism

6.1.1 The Basic GA

The *Basic GA* is based on the simple genetic algorithm introduced in [27]. A straight forward implementation of this algorithm has previously been used in [49, 55] for analog circuit evolution experiments on the FPTA. Hence, it is—with slight changes—used as a reference for the performance of the *Turtle GA*, which is developed in this thesis. Since adaptations of the simple genetic algorithm are widely used in the field of evolvable hardware [42, 60, 71, 77, 84, 108], it is a suitable benchmark for newly developed algorithms.

Implementation of the variation operators.

mutation operator (Basic GA)

The *mutation operator* randomly changes the properties of the FPTA cells according to the defined mutation probabilities. There are three features of the FPTA cell, that can be modified by the mutation operator, namely the routing, the connection of the transistor terminals and the size of the configurable transistor. Thus, a separate mutation probability can be configured for each of the three features. Mutation is carried out in a straight forward way by consecutively addressing all transistor cells, which are used for the respective evolution experiment. Thereby, due to the given probabilities, the six possible routes are flipped and the three transistor terminals are (re-)connected to another target node. Contrary to that, the W and L is not randomly changed, but increased or decreased about up to 3 steps for each application of the mutation operator. It is randomly decided, with a probability of 0.5, whether to increase or to decrease W and L. The actual number of steps is obtained from a gaussian distribution, which is based on the mutation probability. Therefore W/L remains most likely unchanged, while the probability for one (two, three) step(s) are given by the probabilities represented by 1σ (2σ, 3σ) of the Gaussian

Algorithm 6.1: This algorithm describes the course of the evolutionary step, which is used for both, the *Basic GA* and the *Turtle GA*. The evolutionary step creates the offspring population by selecting individuals from the parent population and by subsequently applying crossover and mutation to those individuals. Tournament selection is used as selection scheme. After the new population is created, all individuals are evaluated and ranked. In the next generation, the current offspring becomes the new parent population. Elitism is used for the two best parents by replacing them unchanged with the two worst individuals of the offspring population, in order to increase the probability of preserving good solutions.

```
procedure GATOURNAMENTSELECTOR::SELECT( )
    select a random champion from the parent population
    for i ← 1 to tournament size do
        select a random competitor from the parent population
        if fitness of competitor < fitness of champion then
            competitor becomes new champion
        end if
    end forreturn current champion
end procedure

procedure GAGENETICALGORITHM::STEP( )
    for i ← 1 to half population size do
        select ind1 from parent population
        select ind2 from parent population
        if random float [0..1] < crossover probability then
            cross ind1 with ind2 → ind1', ind2'
        else
            unchanged ind1, ind2 → ind1', ind2'
        end if
        mutate ind1' according to mutation probability
        mutate ind2' according to mutation probability
        add ind1', ind2' to the offspring population
    end for
    measure and evaluate offspring population
    perform elitism: copy best two parents to worst two offspring
    save statistical data
end procedure
```

distribution.

In all experiments, a rectangular area of $n \times m$ FPTA cells ($m, n < 16$) is provided to the GA for circuit evolution. Thus, the *crossover operator* simply exchanges two randomly sized and randomly positioned rectangular areas of transistor cells of two selected individuals. While it is possible to delimit the maximum size of the exchanged blocks of cells, the position can be freely chosen by the GA within the available area of the transistor array. The operation principle of both variation operators are visualized in figure 6.1 and figure 6.2.

crossover operator (Basic GA)

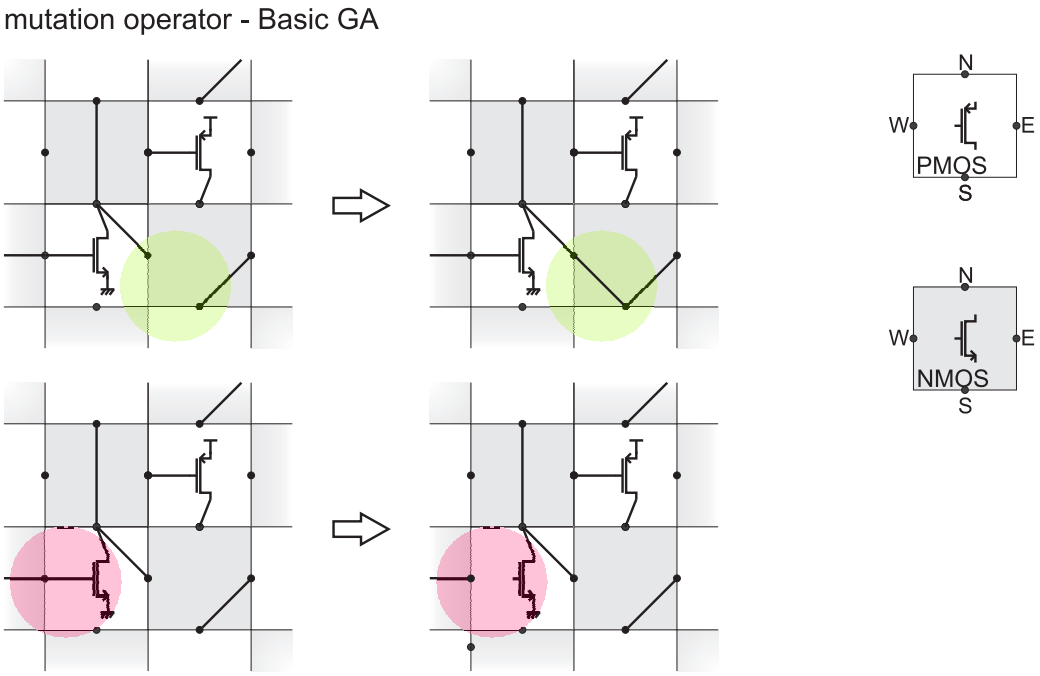


Figure 6.1: *The operation principle of the mutation operator of the Basic GA is depicted above. Mutation is carried out by (re-)connecting randomly selected transistor terminals, switching random routes on or off and randomly changing the W/L ratio of arbitrary transistors. An according crossover operator is shown in figure 6.2*

6.1.2 The Turtle GA

recursive or rule-based circuit creation

The operation principle of the *Turtle GA* is inspired by growing networks like the nervous system or blood vessels. Despite those growing processes are based on randomly created paths, furcations and interconnections, the overall process is directed by developmental rules. In the case of blood vessels, possible developmental rules could be to ensure a minimum volumetric flow rate, a minimum coverage of all parts of the body and the constraint that branches have always to be less in diameter than the main trunk. The circuit creating meachanisms of the *Turtle GA* are inspired by developmental rules of the latter kind. A rule-based developmental stage is supposed to be most efficient in combination with genetic encodings, for which the phenotype is not a one-to-one mapping of the genotype, e.g. in the case of GP [47]. Nevertheless, in this case, the variation operators work on a genetic representation, that is close to the phenotype. The name *Turtle GA* is inspired by the *turtle graphics* language, with which a graphics cursor can be directed to create a plot[1]. Analogous to creating a graph, the variation operators of the *Turtle GA* draw a circuit on the configurable transistor array, based on random decisions for direction and component sizing.

[1] The turtle graphics language has originally been developed for sending commands to a plotter. A basic set of instructions is available and each command (pen up, pen down, move n steps forward, rotate n degrees) is directly carried out by the graphics cursor (turtle).

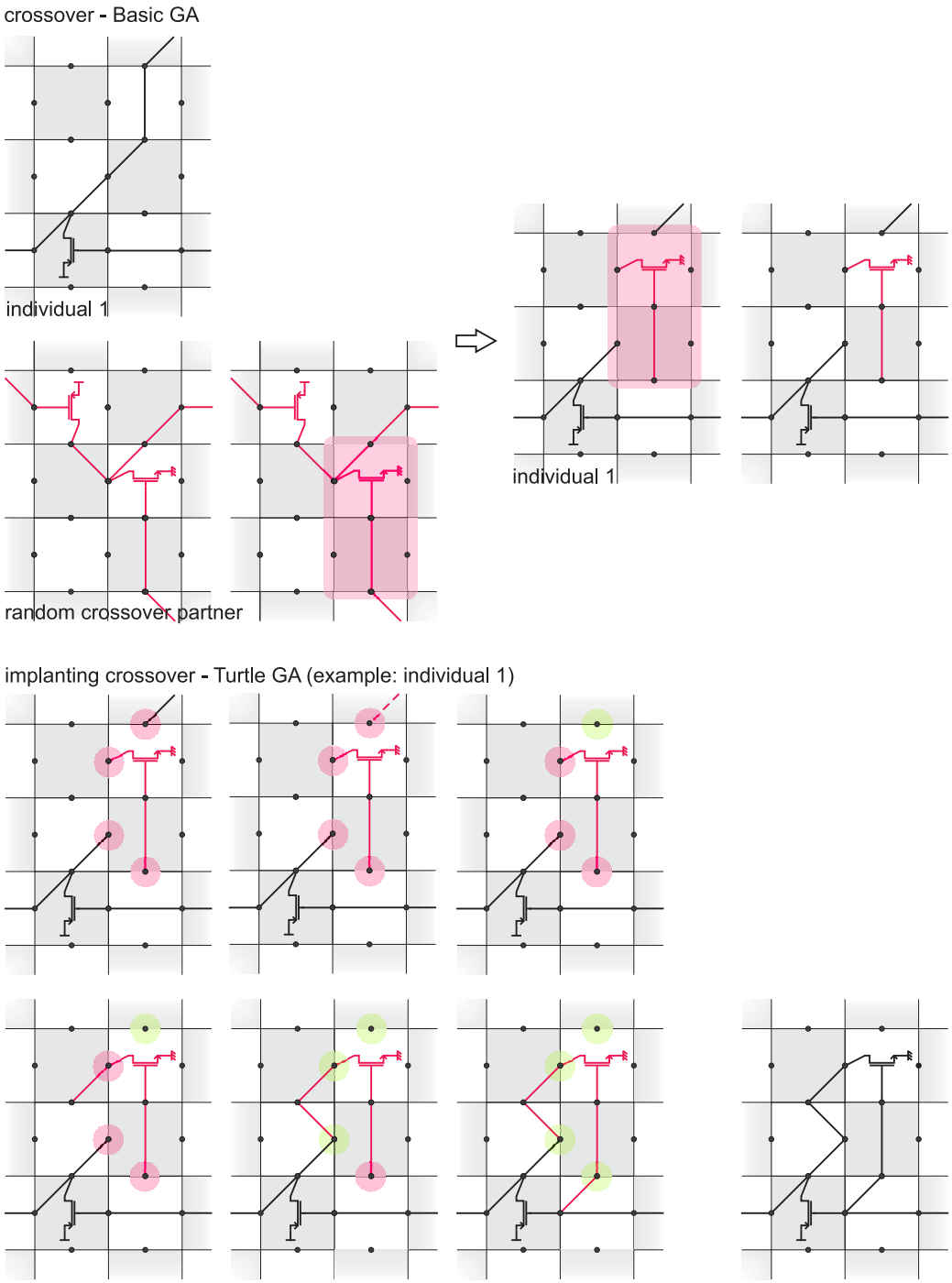


Figure 6.2: *The picture illustrates the operation principle of the crossover operators of the Basic GA (top) and the Turtle GA (bottom) respectively. Principally, the first step (top) of exchanging a randomly sized and positioned block of cells between two selected individuals is the same for both algorithms. While the operation is thereby already completed in the case of the Basic GA, a second stage (bottom) takes care of fixing the occurring floating nodes in the case of the Turtle GA.*

Implementation of the variation operators.

random wires mutation operator (Turtle GA)

The *random wires mutation* operator randomly selects an outside node of an arbitrary FPTA cell as starting point for the algorithm. For such a node, both the current cell and the adjacent neighbor cell provide three possible routing connections to the remaining outside nodes (N, S, W, E) and three connections to the transistor terminals (gate, source, drain). There are two operation modes of the *random wires mutation*, namely the create mode and the erase mode. In case the create mode is active, once the starting point is selected, the algorithm randomly chooses one of the unconnected, possible target nodes, connects it to the current node and subsequently the target node becomes the new starting point. Connections are recursively established until the target node features more than 1 connections, including the new connection. If a transistor terminal is selected as target node, the algorithm is recursively carried out with the two remaining terminals as starting points. In the latter case, the four outside nodes of the current transistor cell are possible target nodes, to which the transistor terminal can be connected. The algorithm works the same way in erase mode, although the stop criterion will be reached, if, after erasing the chosen connection, the target node features either 0 or more than 2 connections. In erase mode, transistors are rewired instead of removed with a probability of 0.5, by switching to create mode after detaching the current terminal. As a consequence of this, the mutation operator inherently avoids floating nodes and terminals, thus, produces valid circuits for simulation.

Thereby, the selection of the target node can be influenced with two configuration parameters: first, it can be configured, whether it is more likely to select a target node of the current cell, or to proceed to the neighbor cell (0.5 means both is equally probable). Second, the probability of connecting a transistor terminal, instead of an outside node, and the maximum number of transistors used can be specified. Furthermore, a maximum number of routing connections per cell can be set, which, if exceeded, will cause the operator to switch more likely to erase mode. Finally, the probability, with which it is decided whether to start in create mode or in erase mode can be configured.

The W and L of a transistor is randomly configured, immediately after the mutation operator has inserted it into the active circuit. During the course of evolution, the sizes of all transistors are varied in the same way as for the *Basic GA*, described in section 6.1.1.

implanting crossover operator (Turtle GA)

Since the relatively complex *random wires mutation* is yet available, the operation principle of the *implanting crossover* operator is simpler. Analogous to the *Basic GA*, two randomly sized and randomly positioned rectangular areas of FPTA cells are exchanged between two different individuals. It is possible to delimit the maximum size of the exchanged blocks of cells, while the position can be freely chosen by the GA within the available area of the transistor array. Since this operation in general breaks the layout of the previously intact circuits, a postprocessing stage takes care of fixing the occurring floating nodes by applying the *random wires mutation* operator to each of them. As a consequence, the validity of the circuit is restored. The operation principle of both variation operators is visualized in figure 6.2 and figure 6.3. Additionally, it is described in pseudocode A.5 and A.6 in appendix A.

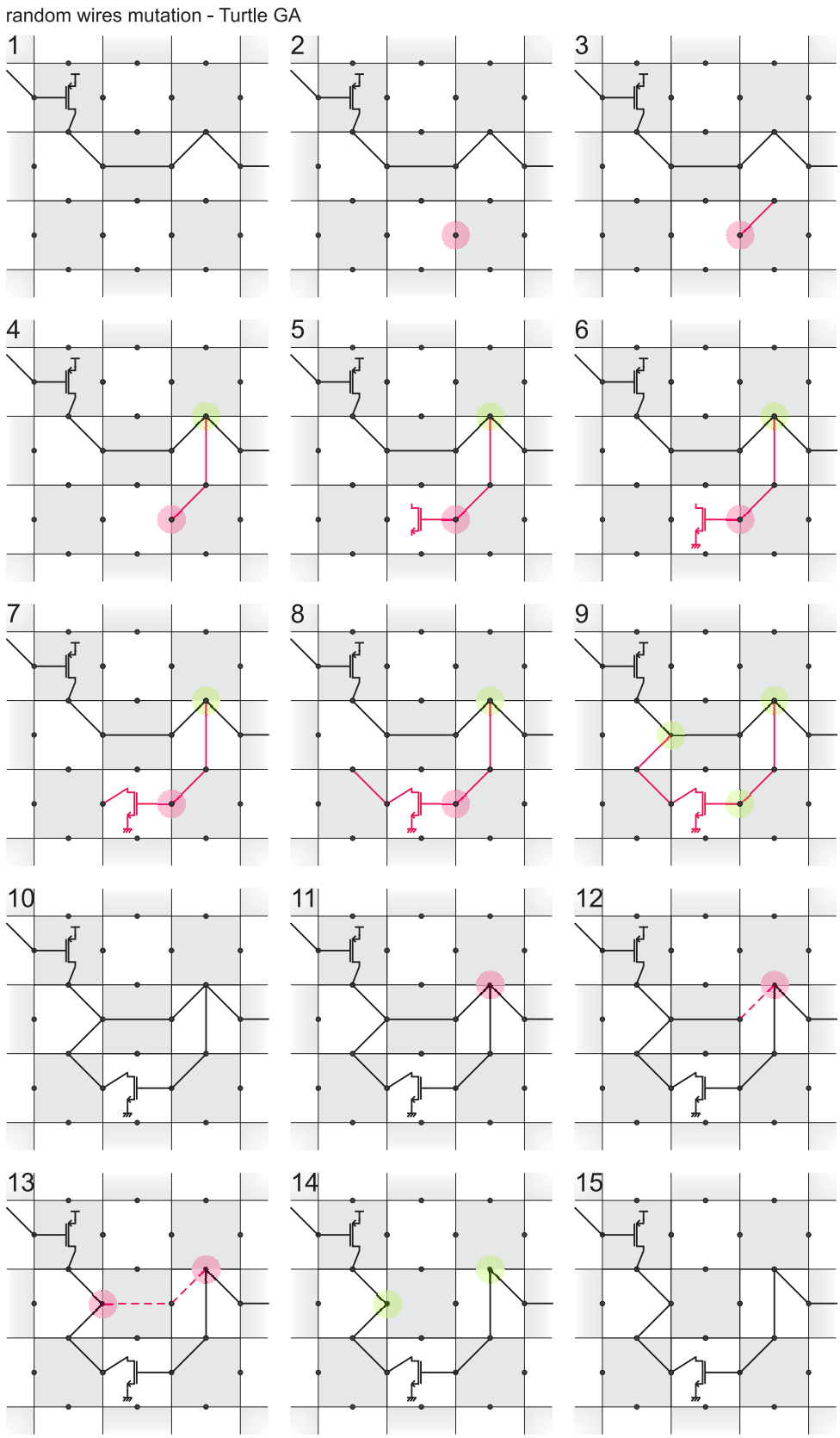


Figure 6.3: *The operation principle of the random wires mutation operator of the Turtle GA is depicted above. Transistors and wires are recursively inserted (or removed) into (from) the active circuit. Pictures 1-10 show an example where a starting point is randomly selected and subsequently a new transistor is attached to the circuit, while in pictures 11-15 the deletion of a wire is illustrated. Further, the W/L ratio of all transistors is randomly changed. The mutation operator is designed in a way that the resulting circuits contain no floating nodes.*

6.1.3 Shortcomings of the Basic GA

unconnected nodes and transistor terminals

no verification outside the FPTA

no transfer to other technologies

sensitive dependency on the evolution substrate

Some important shortcomings of the *Basic GA* can already be seen from figure 6.1, in which a section of a typical genotype is depicted. As can be seen from this figure, the FPTA allows for circuit configurations, that contain unconnected nodes and transistor terminals. Those floating nodes are actually connected to closed transmission gates, which are part of the configuration circuitry of the chip, hence, are not really unconnected. As a consequence of this, if an evolved circuit shall be successfully tested in simulation outside the FPTA, it will be necessary to consider the entire configuration circuitry of the chip. Thus, the simulation time is significantly increased to up to 1 day, in the case of a large circuit. Furthermore, it is—even if they are found—neither feasible to derive new design concepts from the evolved circuits, nor practical to transfer found solutions to other technologies. The reason for this is that the circuits, which are synthesized by the *Basic GA*, may sensitively depend on the influence of the parasitic effects of the configuration circuitry of the FPTA. Thus, in some cases, it is not even guaranteed that an evolved circuit performs equally well on another FPTA chip of the same kind.

6.1.4 Improvements of the Turtle GA Compared with the Basic GA

simulation and verification outside the FPTA

reduced dependency on parasitic effects

The design of the variation operators of the *Turtle GA* aims for overcoming the shortcomings of the *Basic GA*. Hence, the fact that floating nodes and floating terminals are inherently avoided, makes it possible to simulate the evolved circuits outside the FPTA without obligatory considering the influence of the configuration circuitry of the transistor array. Although the parasitic capacitances of the closed transmission gates and the inactive parts of the configurable transistor are still present and influence the evolving circuit, the strength of this dependency is significantly reduced, since there are no longer floating nodes of incompletely connected transistors. In addition to this, it is a great improvement that schematics can be automatically generated from the circuits that are evolved with the *Turtle GA*, by using the SKILL language as described in section 4.5.2. The latter achievement is a great step towards facilitating the understanding of transistor circuits, which are evolved on the FPTA.

6.2 Experimental Setup

testbench with randomly varied capacitive load

quasi-DC measuring

All experiments within this chapter are performed with the *Basic GA* and the *Turtle GA* respectively. For the evolution of the set of logic gates, an area of 7×7 configurable transistor cells is used, whereas for the evolution of comparators an area of 8×8 cells is provided to the EA. As can be seen from figure 6.2, a testbench with randomly varying capacitive load is connected to the circuit's input in both cases, in order to promote circuits, which are able to drive different capacitive loads. The chip-in-the-loop setup inherently provides exclusively the possibility of transient measuring. Therefore, in order to obtain a quasi-DC characteristic of the evolving circuit, relatively long settling times are chosen for each voltage sample in conjunction with randomizing the order of the voltage test pattern.

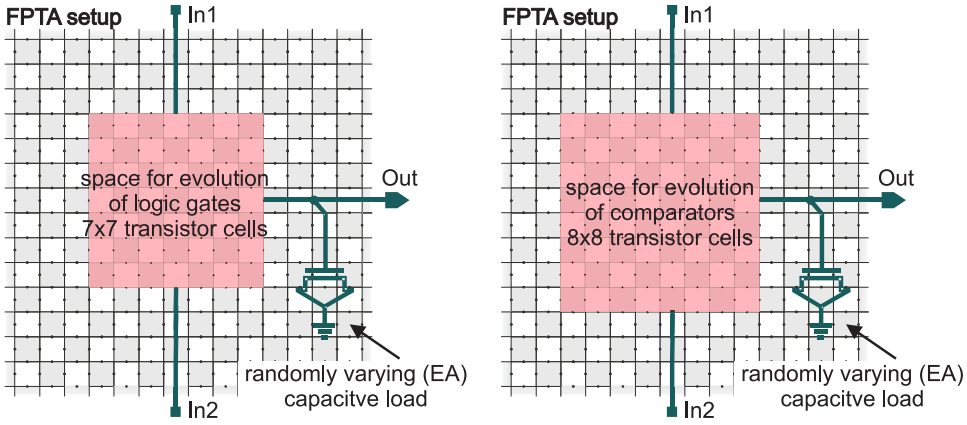


Figure 6.4: *Left: the FPTA setup for the evolution of logic gates is shown. In this case, an area of* 7×7 *transistor cells can be changed by the algorithm. Right: an area of* 8×8 *FPTA cells is provided to the EA for the evolution of comparators. In both cases, a capacitive load is realized with the transistors on the chip and is attached to the circuit's output. Thereby, the advantage is that this capacitive load can be randomly varied by the EA during the course of evolution.*

6.2.1 Test Modes for the Logic Gates

V_{sweep} [V]	V_{set} [V]	AND	NAND	OR	NOR	XOR	XNOR
< 2 (low)	< 2 (low)	0	1	0	1	0	1
< 2 (low)	> 3 (high)	0	1	1	0	1	0
> 3 (high)	< 2 (low)	0	1	1	0	1	0
> 3 (high)	> 3 (high)	1	0	1	0	0	1

Table 6.1: *The truth table for all six symmetric logic gates is depicted above. Providing a better overview, on the right hand side, the logic low and logic high are represented by zeroes and ones. Despite this, in the output voltage pattern, a logic 0 indicates that the target output voltage is* 0 V*, while a logic 1 indicates that the target output voltage is* 5 V*.*

Each of the two test modes for the logic gates features two inputs (V_{sweep}, V_{set}) and one target output (V_{tar}) respectively. The input voltage pattern is given by $V_{\text{sweep}} = 0\ldots2,3\ldots5\,\text{V}$ for each of the 10 values of $V_{\text{set}} = 0, 0.5, 1, 1.5, 2, 3, 3.5, 4, 4.5, 5\,\text{V}$. Thus, each test mode consists of a set of 10 sample curves and a total of 640 voltage samples. Note that the transition regions at $V_{\text{sweep}}, V_{\text{set}} = 2.5 \pm 0.5\,\text{V}$ are not considered, with the aim to facilitate the search for logic gates. The target voltages are either set to 0 V or to 5 V, depending on the result of the desired logic calculation. According to the transistor-transistor logic (TTL) definition, the voltage range of $0\ldots0.8\,\text{V}$ represents a logic *low* and the range of $2\ldots5\,\text{V}$ represents a logic *high*. Thus, from this point of view, the specification of the target voltage pattern is even more restrictive for the presented evolution experiments. The timing scheme of the chip is configured in a way that the sampling frequency is 0.91 MHz. Therefore, randomizing the sample voltage sequence assures a settling-time of at least $1.1\mu s$. Since logic gates shall deliver the same output, if the in-

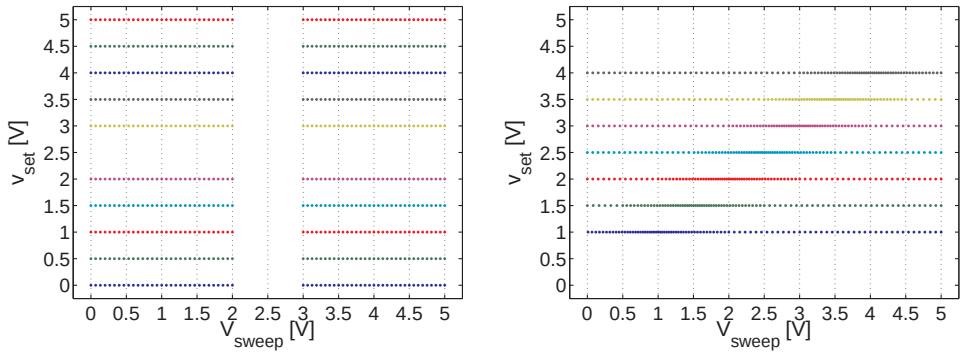


Figure 6.5: *The schemes of the input voltage pattern for the evolution of logic gates (left) and comparators (right) are depicted above. While, in the case of the logic gates, the transition regions of $V_{sweep}, V_{set} = 2 \ldots 3$ are not considered, in order to facilitate the search for good solutions, in the case of the comparators, the density of the voltage samples is even increased towards the switching points, in order to emphasize it.*

puts are swapped, two test modes are necessary to cover both input configurations. A visualization of the input voltage pattern is given in figure 6.5 and the truth table for the logic gates is listed in table 6.1. In the section 6.4, the transition region is included in the measuring ($V_{\text{sweep}} = 0 \ldots 5\,\text{V}$) but not considered for fitness calculation, in order to obtain nicer plots of the output voltage characteristics of the logic gates.

6.2.2 Test Modes for the Comparators

Again, there are two inputs present (V_{sweep}, V_{set}) and one output (V_{tar}). In the case of the comparators, the voltage test pattern features a set of 7 curves with $V_{\text{sweep}} = 0 \ldots 5\,\text{V}$ and $V_{\text{set}} = 1, 1.5, 2, 2.5, 3, 3.5, 4\,\text{V}$. V_{sweep} consists of 100 sample voltages, resulting in a total of 700 samples for each test mode. The values of V_{sweep} are not uniformly distributed between 0 V and 5 V, but with increasing density towards the respective V_{set}, in order to emphasize the switching points for the fitness calculation. Thereby, the smallest difference between two consecutive voltage samples is never smaller that 20 mV, owing to the resolution of the measuring system, discussed in chapter 3, section 3.3.2. The input pattern is illustrated in figure 6.5 (right) and the target voltages are calculated according to

$$V_{\text{tar}} = \begin{cases} 5\,\text{V} & V_{\text{sweep}} \leq V_{\text{set}} \\ 0\,\text{V} & V_{\text{sweep}} > V_{\text{set}} \end{cases}. \tag{6.1}$$

Since the design of a comparator, which performs well within the whole voltage range, is a difficult task, V_{set} takes on values between 1 V and 4 V, in order to help the genetic algorithm in finding good solutions. The order of the voltage samples is randomized, which ensures, alongside with the selected sampling frequency of 0.77 MHz, a minimum settling-time of about $1.3\mu s$. Since it is expected that a comparator will produce the inverse output voltage characteristic, if the inputs are swapped, two test modes—one with swapped inputs and inverse target voltages—are used to assess the evolving circuit's performance.

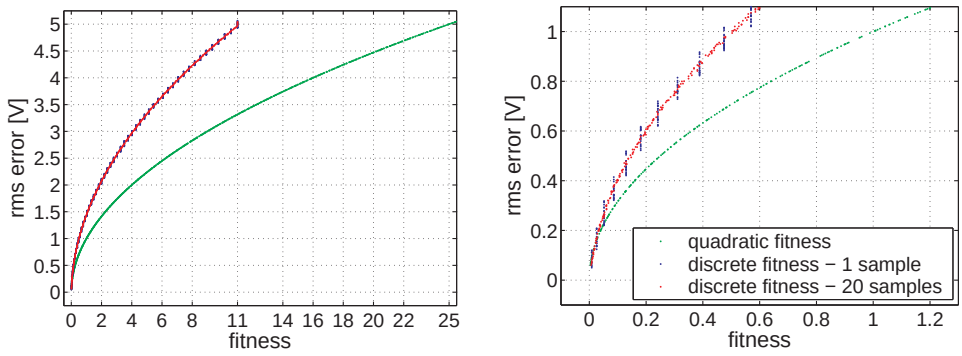


Figure 6.6: *In order to make the fitness values more meaningful in the field of circuit design, the fitness is—for some graphs—translated to the according rms error in volts. The according mapping of fitness ⟷ RMS error is depicted above. Contrary to the squared fitness, the fitness used is discrete, which provides two advantages: first, the measured voltage values are less sensitive to the noisy hardware measurement and the resolution of about* 20 mV *of the hardware evolution system. Second, the resolution of the fitness measure is refined towards small voltage changes, while improvements of large voltage changes are robust.*

6.2.3 Simulator Setup

the fabrication process of the FPTA

The simulations are carried out with the SPICE simulator, described in chapter 4 and in [65]. BSIM3v3 transistor models of the AMS 0.6 μm process, with which the FPTA is designed, are used for simulation. SPICE netlists are extracted from the circuits that have been evolved on the transistor array and the input voltage patterns correspond to those used for the on-chip measurements. In cases where the simulation delivers more output voltage samples than the measuring on the FPTA, only the common subset of samples is considered for fitness calculation. Additionally, the fitness values, calculated from the simulation results, are obtained by using the same fitness functions as are used throughout evolution.

SPICE Netlists of the resulting circuits are extracted as described in chapter 4, section 4.4. Level 2 netlists are used for comparing the results from the chip to those obtained from simulation. Furthermore, a load-capacity of 10 pF is attached to the circuit's output in simulation, owing to the randomly varied capacitive load, which is present at the circuit's output during evolution.

6.2.4 Fitness Measure

accumulated fitness value

The same fitness measure is used for all experiments in this chapter, in order to be able to compare them. Both the setup for the logic gates and the setup for the comparators consist of two test modes respectively and each test mode delivers a separate fitness value. Additionally, a third fitness value is obtained by quantifying the ressource consumption. The latter three fitness values are added up to the total fitness of an individual, which is denoted as *accumulated fitness*. It is a great problem if circuits, which use less ressources, are preferred right in the beginning of the experiment, since evolution will get more often stuck in the local optimum of using no transistors and no routes at all after only a few

generations. To overcome this problem, a widely used approach is to assign the worst fitness for the ressource consumption as long as the fitness, calculated from the output voltage characteristics, is not below a certain threshold. Hence, the latter approach is also applied for the experiments in this chapter. The worst ressource consumption of 2048 is thereby calculated by considering all 256 transistors and 256×6 routes of the FPTA in equation 6.7 and all fitness values are calculated as follows:

$$\Delta V_i = |V_{\text{target}_i} - V_{\text{measured}_i}| \tag{6.2}$$

$$N = \text{no. of voltage samples} \tag{6.3}$$

$$\text{discrete fitness} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{50}\begin{cases} j \cdot \frac{11}{1275} & \Delta V_i - 20\,\text{mV} > 0.1 \cdot (j-1) \\ 0 & \Delta V_i - 20\,\text{mV} \leq 0.1 \cdot (j-1) \end{cases} \tag{6.4}$$

$$\text{squared fitness} = \frac{1}{N}\sum_{i=1}^{N}\Delta V_i^2 \tag{6.5}$$

$$\text{rms error} = \sqrt{\text{squared fitness}} \tag{6.6}$$

$$\text{ressource consumption} = 2 \times \text{transistors used} + 1 \times \text{routes used}. \tag{6.7}$$

$$\tag{6.8}$$

mapping the fitness to the corresponding rms error

Furthermore, for the purpose of making the fitness values more meaningful in the field of circuit design, the fitness is—for some graphs—translated to the according rms error in volts. The mapping of fitness $\longleftrightarrow$ RMS error is depicted in figure 6.6. For illustration, rms error and fitness are calculated for one single voltage sample and for 20 randomized voltage samples respectively. Contrary to the squared fitness, the fitness used is discrete, which can be seen from the curve for 1 voltage sample in figure 6.6 and is referred to as *discrete fitness*. In the case of hardware evolution, the discrete fitness features some advantages compared with the squared fitness: first, the measured voltage values are less sensitive to the noisy hardware measuring and the resolution of about 20 mV of the hardware evolution system (chapter 3, section 3.3.2). Second, the resolution of the fitness measure can be refined towards small voltage changes by decreasing the distances of the fitness steps, while improvements of large voltage changes are more robust and therefore preserved.

6.2.5 Estimation and Setup of the EA Parameters

As can be learned from publications about scanning the parameter space and finding optimal parameters for EAs ([36]) and from test runs of evolution experiments, carried out for this thesis, optimally tweaking the EA is a difficult and time-consuming task. Actually, there are two approaches for finding the optimal EA parameters for a certain a certain task: the first possibility, which surely finds the global optimum, is simply to sample the whole parameter space by performing according experiments and subsequently pick the parameters of the setups, that have lead to the best results. However, a great disadvantage of the latter approach is the immense time consumption, since the EA's parameter set consist of at least 4 independent variables, which have to be optimized, namely mutation rate, crossover rate, population size and selection pressure. The influence of custom fitness functions and genetic encodings is thereby not yet even taken into account. In

practice, for a customized implementation of an EA, about 20-30 independent parameters, rather than only 4, have to be optimized. As a consequence of this, assuming the simplest case, 10 samples for each parameter (which is quite coarse) and a statistics of only 10 runs for each parameter set, a total of $10 \times 10^4 = 100000$ experiment runs would have to be performed, in order to find a good set of parameters for one specific problem. The second possibility is to make an educated guess for finding a good parameter set, although this will possibly not lead to the global optimum.

Since the focus of this work is rather set on improving the EA and finding solutions for various problems, than to exhaustively sampling the parameter space, the parameter set used is found by an educated guess. Important parameters of the *Basic GA* and the *Turtle GA* are listed in table 6.2 conjoined with their numerical value and the considerations, on which the respective value is based.

an educated guess for the EA parameters

In order to substantiate the chosen set of parameters, 20 evolution experiments—for which the task is to evolve a logic AND–are carried out with those parameters. Furthermore, another 20 evolution runs are carried out for certain variations of the chosen parameter set respectively. Variations of the parameter set are thereby obtained by slightly increasing or decreasing certain parameters. Only one parameter is changed at the same time, hence, the results obtained with the varied parameter sets deliver information about the vicinity of the respective variable and are listed in table 6.2. Concluding those results, it can be stated that a parameter set, with which a good performance of the EA is achieved, is found, although it cannot be claimed that the global optimum is found.

parameter	which GA	smaller < used < greater value	success rate $rms < 200\,\mathrm{mV}$	success rate $rms < 500\,\mathrm{mV}$	educated guess, considerations
mutation rates					The mutation rates used are picked by dividing 1 by the number of configurable routes (6×256), terminals (3×256) and W/L (256). Thus, the rates are set to values, with which, on average, one of each kind of cell features is changed per individual and generation.
routing	Basic	$0.15\,\% < 0.3\,\% < 0.6\,\%$	$10\,\% > 5\,\% > 0\,\%$	$30\,\% < 60\,\% > 55\,\%$	
terminals	Basic	$0.3\,\% < 0.6\,\% < 1.2\,\%$			
W/L	both	$0.8\,\% < 1.6\,\% < 3.2\,\%$			
turtle rate	Turtle	$0.5 < 1 < 2$			
erase/create	Turtle	$40\,\%/60\,\%$	$10\,\% < 25\,\% > 20\,\%$	$45\,\% = 45\,\% = 45\,\%$	
reconnect	Turtle	$50\,\%$			
to old/new cell	Turtle	$30\,\%/70\,\%$			
crossover rate	Basic	$10\,\%$			About 10 % of the individuals are subject to a crossover operation.
	Turtle	$5\,\% < 50\,\% < 20\,\%$	$5\,\% < 25\,\% < 30\,\%$	$40\,\% < 45\,\% > 65\,\%$	
cross. block size	both	$0 \ldots 4 \times 0 \ldots 4$			
elitism	both	keep 2 best ind.			The significant value is #evaluations = pop. size × no. generations. An adequate selection pressure is achieved by setting the tournament size to about 5 % of the pop. size.
tournament size	Basic	3			
	Turtle	$2 < 3 < 4$	$20\,\% < 25\,\% > 10\,\%$	$40\,\% < 45\,\% > 25\,\%$	
population size	Basic	50			
	Turtle	$25 < 50 < 100$	$20\,\% < 25\,\% > 15\,\%$	$30\,\% < 45\,\% < 55\,\%$	
no. generations	both	10000			

Table 6.2: *The GA parameters for the Basic GA and the Turtle GA, which are used for most of the experiments in this thesis, are listed in the table above. The FPTA consists of* $16 \times 16 = 256$ *configurable transistor cells, each of them featuring six possible routes, three transistor terminals and one W/L ratio. Since it is desired that the individuals are varied without destructive impact on useful structures, it makes sense to set the basic mutation rates to values that, on average, only one of the individual's features is changed per generation. These conservative rates do not allow for effective exploration of the search space in the beginning of evolution. Therefore, the basic mutation and crossover rates are multiplied with a factor* $1 \leq multiplier \leq 5$*,* $multiplier \in \mathbb{N} \,|\, current\ best\ fitness < \frac{worst\ fitness}{2 \cdot \tau^{multiplier-1}}$*,* $1 < \tau < 2$*, that depends on the current best fitness.*

6.3 Measuring the Performance of the Variation Operators of Both GAs

Although the *Turtle GA* features many eligible improvements compared with the *Basic GA*, it has to be assured that the new variation operators perform equally well as the basic operators. Thus, in order to measure the performance of the variation operators of both GAs, they are applied to the same individual for one evolutionary step and the differences between the previous fitness and the respective new fitness is recorded. Thereby, owing to the noisy output of the hardware, only those fitness differences are considered, that exceed a certain threshold. This threshold, in turn, is obtained by performing multiple measurements of the same circuit without changing it and subsequently calculating the gaussian width of the distribution of the obtained results. Due to the fact that measuring only one single individual is not representative, a set of different individuals with similar fitness is measured.

creating a repository of individuals

For carrying out this measuring, different individuals with different fitness values are collected during the evolution of logic gates (AND, XOR) and are stored in a repository. This repository is divided into 100 equally sized fitness intervals, that cover the range of resulting fitness values between $0.5 \ldots 5.5$, which corresponds to rms errors between $0.8 \ldots 3.4\,\mathrm{V}$. The fully populated repository features 100 different genomes for each fitness interval and therefore contains a total of 10000 individuals. This repository makes it possible to reproducible generate predefined initial populations for evolution experiments. In the case of measuring the performance of variation operators, two fitness intervals are

which task	which GA	rms error interval	success rate mutation [%]	failure rate mutation [%]	success rate crossover [%]	failure rate crossover [%]
AND	Basic	1.5 V	25.6	11.7	27.8	6.2
		2.5 V	25.7	6.0	27.2	4.7
	Turtle	1.5 V	25.7	12.1	26.5	5.5
		2.5 V	30.8	9.7	30.2	6.3
XOR	Basic	1.5 V	7.2	39.0	7.9	8.2
		2.5 V	19.8	23.4	6.0	11.4
	Turtle	1.5 V	15.7	49.9	4.2	47.7
		2.5 V	7.5	27.2	5.1	17.0

Table 6.3: *The success rate and the failure rate of the mutation and crossover operator of the Basic GA and the Turtle GA are opposed in the table above. Thereby, the success rate (failure rate) is calculated from* $\frac{\text{no. of runs with fitness below (above) threshold}}{no.of total runs=100000}$*. It can be seen that the variation operators of both EAs perform equally well for both, populations of individuals with a relatively good fitness and populations of individuals with a bad fitness respectively. In the case of the XOR, it is surprising that the mutation success rates and the crossover failure rates show an inverse behavior. A possible reason for the latter behavior could be the fact that, first, the XOR turns out to be hard to evolve on the FPTA and second, contrary to the AND, the output voltage characteristic of the XOR is symmetric. Thus, the performance of the operators for corresponding fitness intervals is not necessarily the same. Besides, greater success rates suggest faster convergence of the individuals in the case of the AND.*

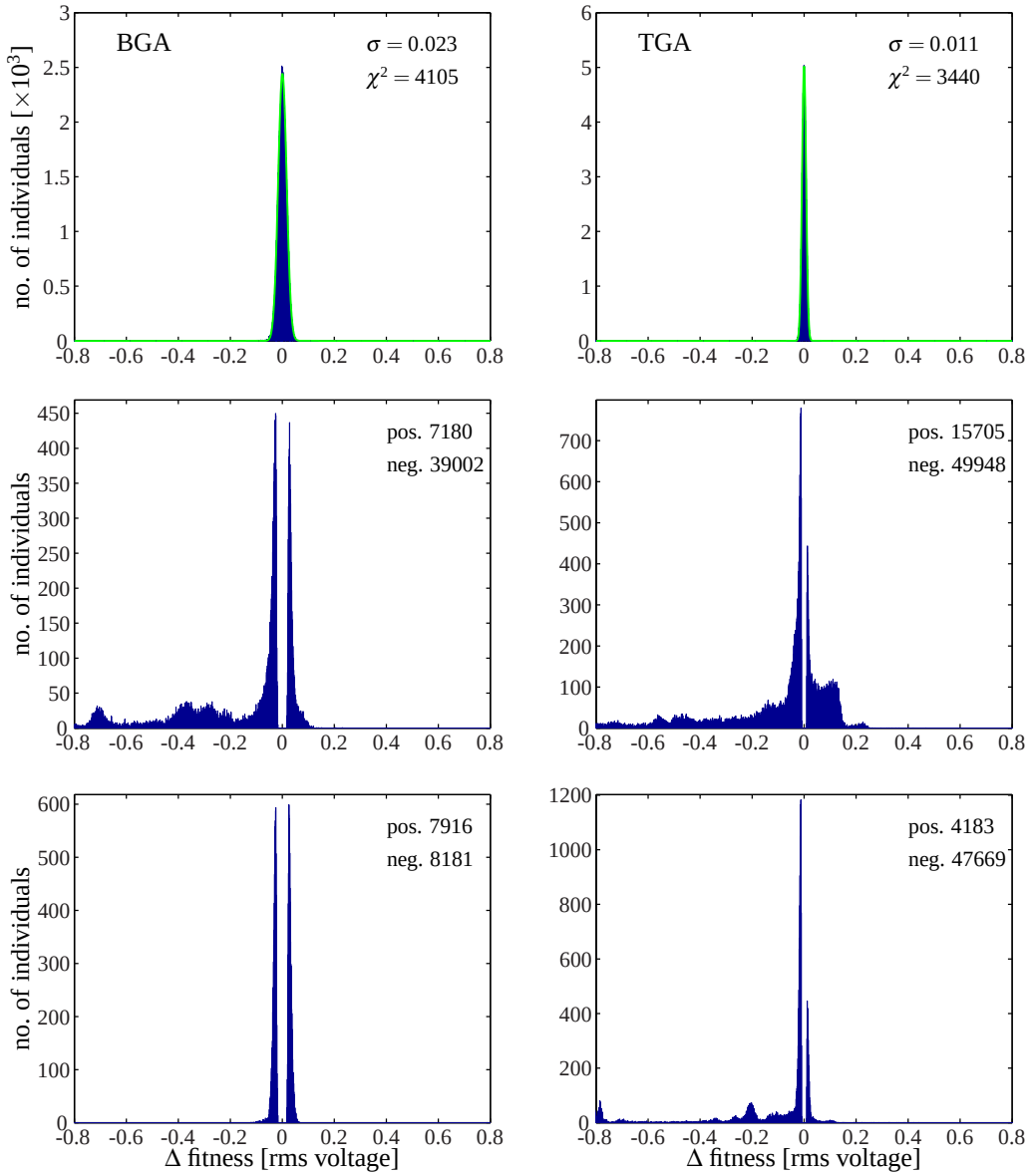


Figure 6.7: *For illustration, the distributions of the results from the XOR individuals with a fitness of 1 are depicted above. The results for the Basic GA (BGA) are depicted on the left hand side, while those for the Turtle GA (TGA) are shown on the right hand side. Thereby, the graphs on top show the measuring of the unchanged individuals, those in the middle show the influence of the mutation operator and those at the bottom show the influence of the crossover operator. Further, a positive Δfitness means that the individual is improved, while a negative Δfitness means that the individual is degraded.*

repeatedly applying the genetic operators

selected for the setup: one fitness interval of individuals with a relatively good fitness of 1 (resp. an rms error of 1.5 V) and another interval of individuals with a bad fitness of 3 (resp. an rms error of 2.5 V). Subsequently, each individual of a selected interval is tested 1,000 times, resulting in 100,000 for the mutation operator, the crossover opera-

tor and the unchanged genome respectively. The Gaussian Width of the distribution of the latter measurements is used as a threshold, that has to be exceeded when the genome is changed, in order to distinguish fluctuations of the output from real improvement or degradation of the individuals. Finally, the ratio of measurements, where the application of a genetic operator lead to better fitness, resp. worse fitness, can be used to express the efficiency of the applied variation operator. The results for the success rates of mutation and crossover of the *Basic GA* and the *Turtle GA* are compared in table 6.3 and, for illustration, the distribution of the results from the XOR individuals with a fitness of 1 is depicted in figure 6.7.

success rates of the genetic operators

As can be seen from table 6.3, the variation operators of both GAs perform equally well for populations of individuals with a relatively good fitness and populations of individuals with a bad fitness respectively. In the case of the XOR, it is surprising that the mutation success rates and the crossover failure rates show an inverse behavior. Possible reasons for the latter behavior are: first, the XOR turns out to be hard to evolve on the FPTA. Second, contrary to the AND, the output voltage characteristic of the XOR is symmetric. Thus, the performance of the operators for corresponding fitness intervals is not necessarily the same. Concluding, the results of the performance measuring suggest that both GAs will show comparable performance for the following evolution experiments. Besides, as long as the success rate is $> 0\,\%$, the individuals are further improved by means of selection. Thereby, the greater the success rate, the faster converge the individuals.

6.4 Results for the Evolution of Logic Gates and Comparators Using Both EAs

experimental setup

A total of 50 evolution runs is carried out with the *Basic GA* and the *Turtle GA* respectively, in order to find solutions for all six logic gates (AND, NAND, NOR, OR, XOR, XNOR) and the comparators. The population size is always 50 individuals, which are processed for 10.000 generations in the case of the gates and for 20.000 generations in the case of the comparators. A complete list of the EA parameters is given in table 6.2. The output fitness value range is $0 \ldots 11$. Thereby, 0 is the best fitness which would only be obtained by the target voltage pattern itself and 11 is the worst fitness value, which would be produced by a circuit with the inverse target voltage characteristic. The individuals of all evolution runs are randomly initialized and it is observed that the best individual of the initial generation in most cases produces a straight line as output voltage characteristic. Hence, the best initial rms error is about $2.5\,\mathrm{V}$ resulting in a best initial fitness in the range of 3..4. Thus, for a better overview, the x-axis of the fitness histograms of the resulting populations ranges between 0 and 3, instead of between 0 and 11. As a short remainder, the main questions, that shall be answered by looking at the following results are:

- How do the *Basic GA* and the *Turtle GA* perform in finding solutions for logic gates and comparators?
- Are both EAs performing equally well?
- Do the circuits work on different substrates?
- How well does the *Turtle GA* perform its new features: transfer to other technologies, reduced ressource consumption and schematic generation?

6.4.1 Comparison of the Results of Both Algorithms

On-chip Results for the Logic Gates

logic gates: equal on-chip performance of both EAs

For the *Basic GA* and the *Turtle GA*, the best fitness of the resulting population of each run is depicted in histogram 6.9. It can be seen that both EAs end up in similar regions of fitness for the same logic gate respectively. Thus, in the case of the logic gates, both algorithms perform equally well. Furthermore, histogram 6.9 shows that there are three difficulty levels for the evolution of logic gates on the FPTA: first, the NAND and the NOR are the easiest gates to evolve, since the resulting best fitness of almost all runs is below 0.25 (rms error $< 700\,\text{mV}$). Second, the AND and the OR gates, which are more difficult to evolve, due to the fact that, in this case, only $\frac{2}{3}$ of the runs feature a fitness below 0.25, while the remaining part features fitness values between 0.25 and 1. Additionally, there are $2\ldots3$ runs with a fitness between 1.5 and 2.5 ($1.3\,\text{V} <$ rms error $< 2.5\,\text{V}$). Third, the XOR and the XNOR are the hardest to evolve, since the resulting best fitness values of the experiment runs are distributed between 0 and 3. Nevertheless, for the XOR and the XNOR, at least 2 out of 50 runs achieved a fitness below 0.25. It can be seen that, with increasing complexity (NAND,NOR=4 transistors, AND,OR=6 transistors, XOR,XNOR=14 transistors), the circuits get harder to evolve. Concluding this so far, both algorithms are able to reliably find good solutions for all six logic gates on the FPTA. Measured output voltage characteristics of the best solutions are shown in figure 6.14.

more complex circuits are harder to evolve

On-chip Results for the Comparators

The fitness of the best individual of each evolution run is shown in histogram 6.10 for both algorithms. In the case of the comparators, the resulting fitness values of both EAs are ranging between 0.1 and 1 ($50\,\text{mV} <$ rms error $< 1.5\,\text{V}$), although the distribution of the fitness values indicates that the *Basic GA* performs better in finding good solutions: about half of the runs achieved a best fitness below 0.25 (rms error $< 700\,\text{mV}$). Contrary to that, only 25% of the runs of the *Turtle GA* feature a best individual below a fitness of 0.25. Despite this, four runs reached the best fitness bin in both cases. It is interesting to see that, considering the complexity of 7 transistors of a basic comparator circuit, the distribution of the fitness values is similar to those of the AND and the OR, which are equally complex. As a consequence of this, it is suggested that the probability, with which a good solution is found, rather depends on the number of transistors and their connectivity, than on the particular problem or the type of circuit. To conclude, it can be stated that both algorithms are able to reproducible synthesize comparators with a good performance on the chip. According output voltage characteristics of the best circuits found are depicted in figure 6.15.

comparators: on-chip performance of Basic GA is slightly better

finding good solutions rather depends on no. of transistors, than on problem

Considering the Ressource Consumption

The first significant advantage of the *Turtle GA*, compared with the *Basic GA*, can be seen from figure 6.8 and figure 6.11, where the rms error of the best individual of all experiments is graphed over the resource consumption (no. of transistors used) of the respective circuit. For drawing the figures 6.8 and 6.11 the rms error is chosen, instead of the fitness, since this makes it easier to qualify the electrical performance of the re-

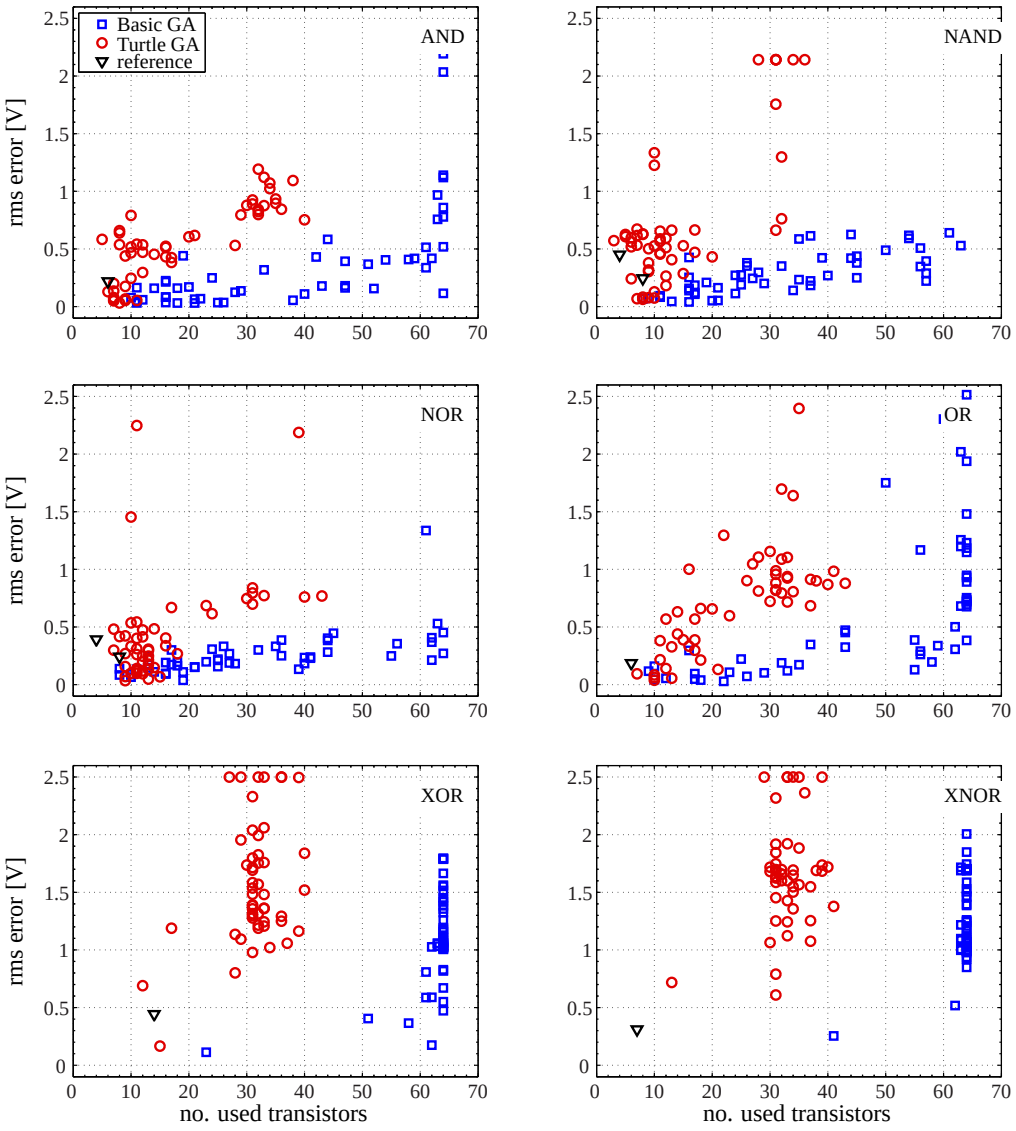


Figure 6.8: *The fitness from the test modes is plotted over the no. of transistors used. As can be seen from the graphs, the Turtle GA features reduced ressource consumption, compared with the Basic GA. Furthermore, both EAs achieved to find solutions for the logic AND, NAND, NOR, OR and XOR, which perform equally well as the respective manually made reference designs, which are realized on the FPTA.*

sulting circuits. In both experiments, the evolution of logic gates and the evolution of comparators, the *Turtle GA* produces circuits, which use less transistors than the *Basic GA*. Reducing ressource consumption is desireable insofar that, on the one hand, freed ressources can be (re-)used by the EA for the development of additional features and, on the other hand, a smaller number of transistors increases the probability of understanding evolved circuits.

the Turtle GA significantly reduces ressource consumption

advantage: (re-)using ressources

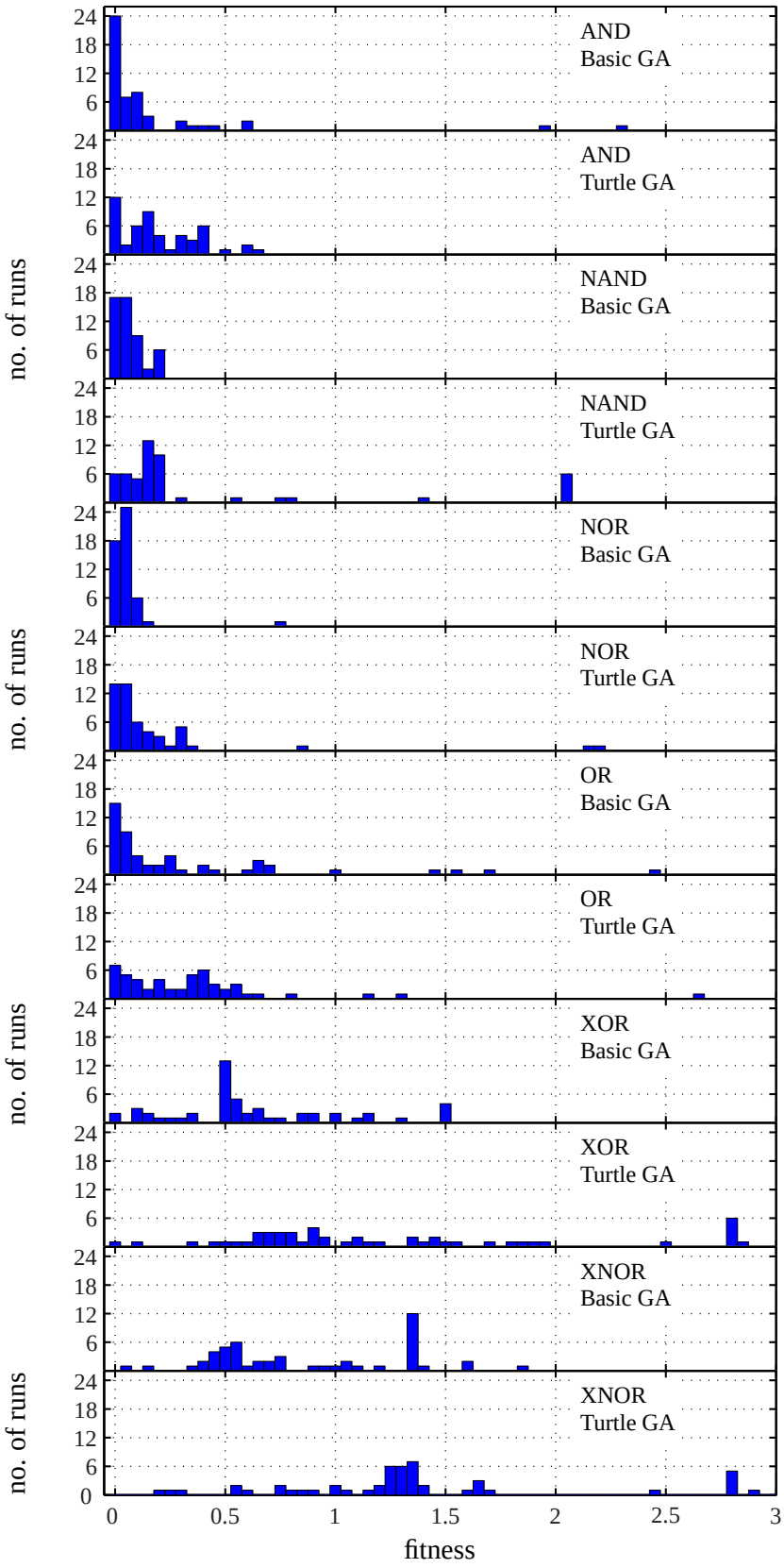


Figure 6.9: *The distribution of the fitness of the best resulting individual of all 50 evolution runs is graphed. The outcome of both EAs, namely the Basic GA and the Turtle GA are depicted for all 6 logic gates. It can be seen that the number of good solutions decreases with increasing complexity of the circuits.*

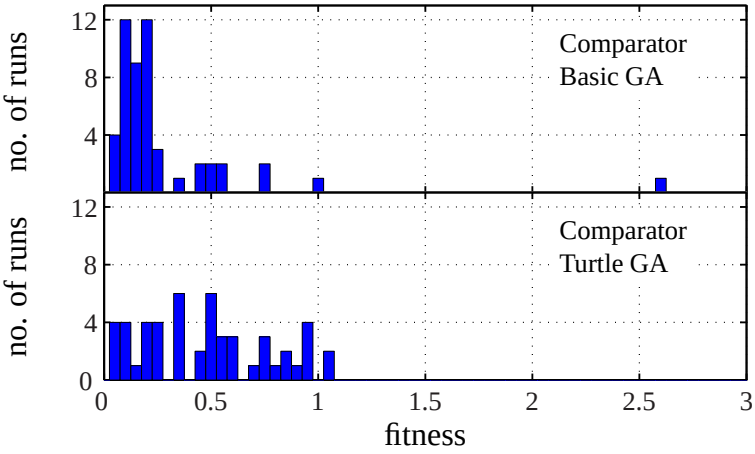


Figure 6.10: *The distribution of the best fitness for comparators of each of the 50 evolution runs is shown in the above histograms. Thereby, both the Basic GA and the Turtle GA achieve to reliably find good solutions for comparators, although the over-all performance of the Basic GA is slightly better in this case.*

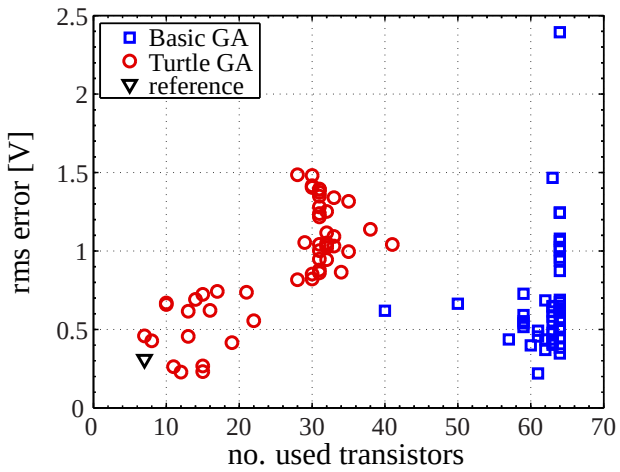


Figure 6.11: *Once more, for the comparators, the fitness from the test modes is plotted over the no. of transistors used. Thereby, it can be seen that in the case of the comparators, the Turtle GA achieved to significantly reduce the ressource consumption, compared with the Basic GA. For comparison, the performance of a manually made reference design on the FPTA is marked with a triangle.*

6.4.2 Verifying the Evolved Circuits in Simulation

The second important feature of the *Turtle GA* is the possibility of extracting the resulting circuits into SPICE netlists, as described in section 4.4, and therefore it is possible to validate them with a simulator outside of the chip. Since the aim is to proof that good solutions, which are evolved on the FPTA, are working outside the chip without including the whole configuration and controlling circuitry of the transistor array into simulation (level 3), the simulations of the circuits are carried out with netlists of level 2, in which only plain transistors and the mean parasitic on-resistance of the switches are included. Contrary to that, level 1 simulations, which represent each transistor cell as plain transistor, are not used, due to the fact that it has been observed in preliminary experiments that the level 1 simulations generally fail. One individual of each run, which features the best performance on the chip, is measured on the FPTA, in a DC SPICE simulation and in a transient SPICE simulation respectively.

Simulation of the Logic Gates

the Turtle GA finds logic gates, which perform well in simulation

The results for the logic gates are compared in histogram 6.12. It can be seen that, although the performance of many solutions decreases in the DC simulation, at least 1 out of 50 individuals with a good fitness performs equally well as on the chip. DC voltage characteristics of the on-chip measuring of the best solutions of both GAs and the DC voltage characteristic of the simulation of the best *Turtle GA* solution are depicted in figure 6.14. Contrary to the DC simulation, the performance of the circuits is getting significantly worse in the transient simulation. Nevertheless, this is expected, since parasitic capacitances are not included in the netlist. Despite the worse rms error, the transient voltage characteristics, graphed in figure 6.16, show the expected behavior of the respective logic gate, which is a positively surprising result.

transient performance is worse than DC performance

Simulation of the Comparators

the Turtle GA finds comparators, which perform equally well in simulation

In the case of the comparators, the rms errors obtained from simulation and from the measurement on the chip are similar. The results are shown in figure 6.13. Once more, 1 out of 50 evolution runs features an individual with a good performance, which performs equally well in the DC simulation and on the FPTA. The output voltage characteristics are depicted in figure 6.15. For the comparators, the output of both test modes is shown, since, contrary to the logic gates, the curves will be inverted, if the inputs are exchanged. Furthermore, although the rms error of the majority of solutions gets significantly worse, the over-all best individual performs well in a transient simulation, as can be seen from figure 6.17. Thus, it is possible to synthesize circuits on the FPTA that are not bound to the architecture of the configurable transistor array.

good solutions are not bound to the FPTA

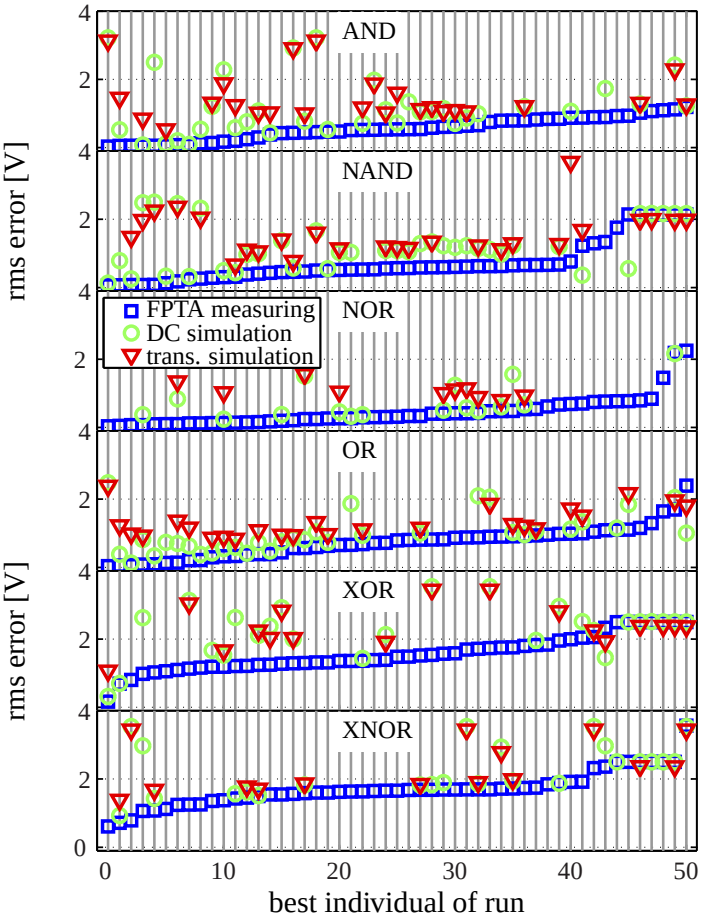


Figure 6.12: *The performance of the best solutions for the logic gates on the FPTA are compared with their performance in a DC and a transient SPICE simulation for all 50 evolution runs respectively. For a better overview, the results are sorted by their fitness, obtained with the FPTA, in ascending order. It can be seen that about 50…70 % of the circuits are failing in simulation. Despite this, in all cases, at least one good solution is found, which performs equally well on the chip and in simulation.*

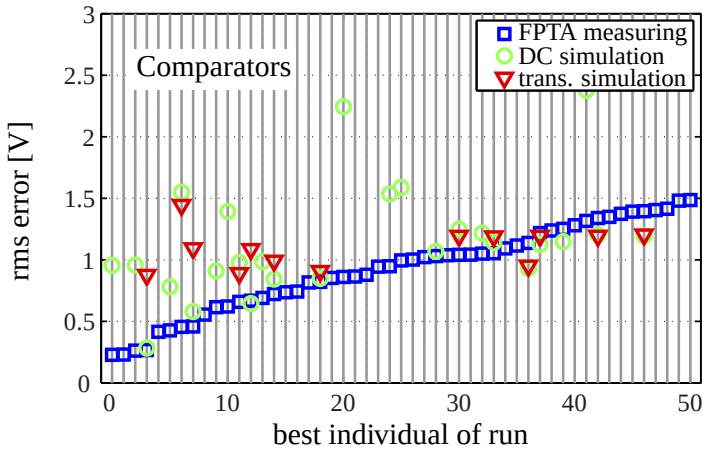


Figure 6.13: *The on-chip performance of the best comparators of all 50 evolution runs is compared with their performance in a DC and a transient SPICE simulation. Again, for a better overview, the results are ascending sorted by their fitness, obtained with the FPTA. Thereby, about half of the circuits are not working in a DC simulation and about 3/4 of them fail in a transient simulation. Despite this, at lest 4 solutions are found, which achieve a similar fitness in all three cases.*

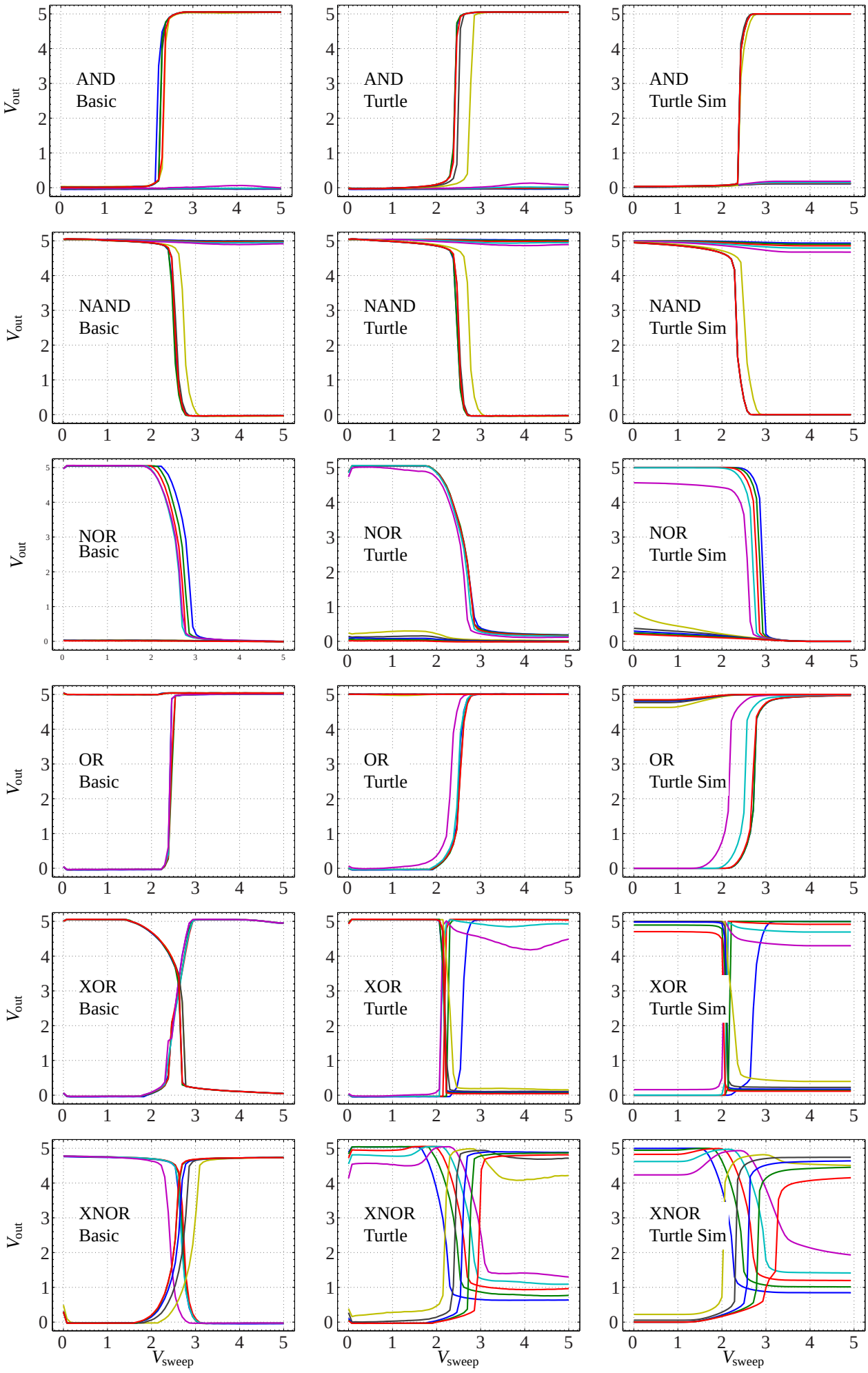


Figure 6.14: *The output characteristics, obtained from the Basic GA, the Turtle GA and the DC simulation for all 6 logic gates are depicted above.*

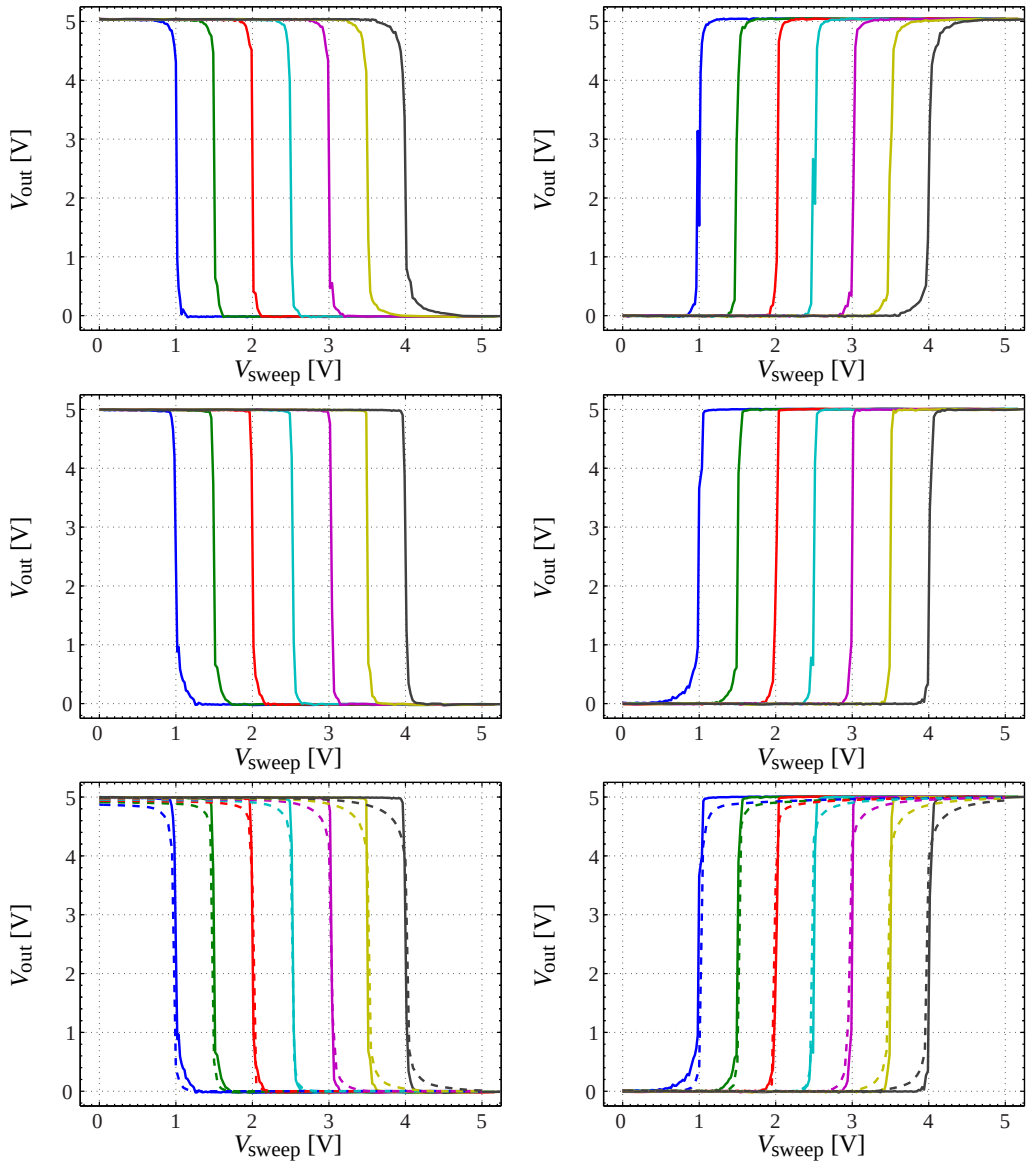


Figure 6.15: *The output voltage characteristics of the best comparators, evolved with the Basic GA (top) and the Turtle GA (middle and bottom) are depicted. In the case of the Turtle GA the output voltage characteristic from the FPTA is additionally compared to the DC simulation result (bottom).*

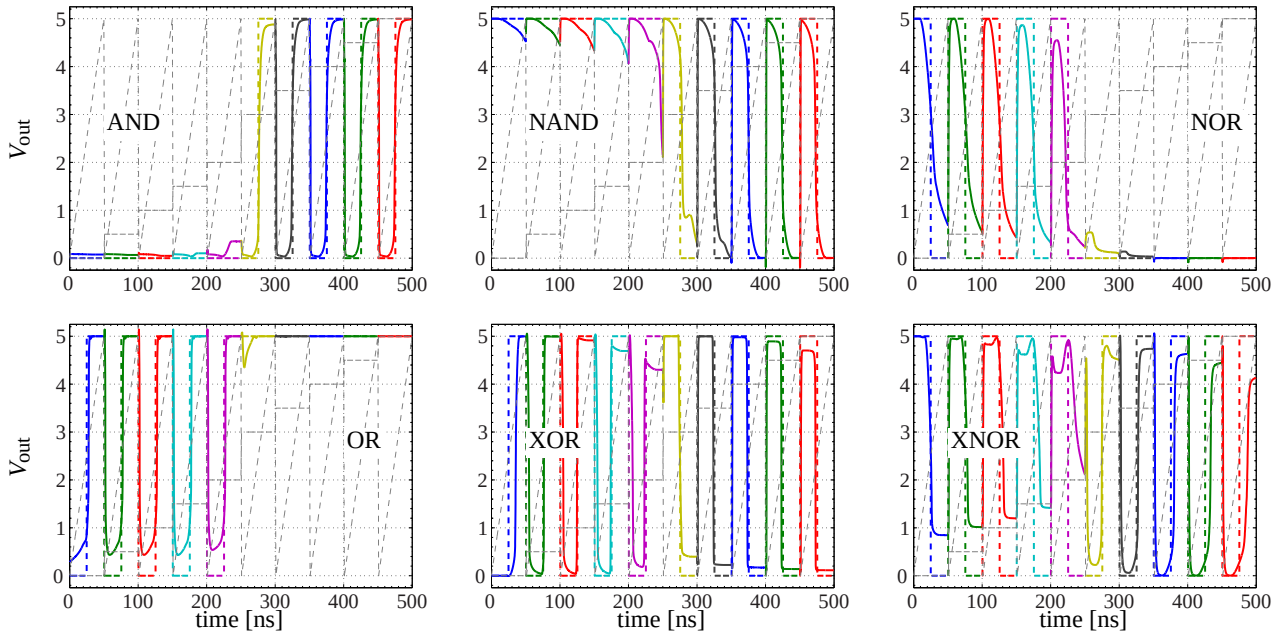


Figure 6.16: *In addition to the DC voltage characteristic of the logic gates, shown in figure 6.14, it is interesting to see that the transient output voltage characteristic of the over-all best solutions, depicted above, shows the desired behavior.*

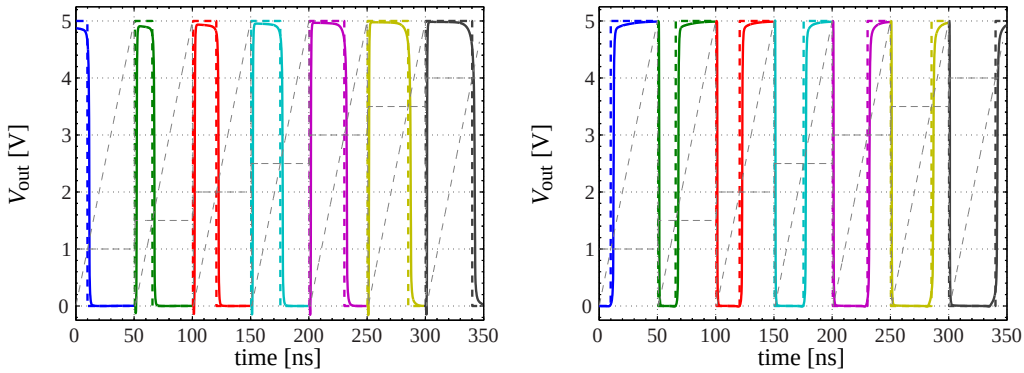


Figure 6.17: *In addition to the voltage characteristic of the best comparator, obtained with a DC simulation, the measuring of an according transient simulation is depicted above. It can be seen that the best coparator, which is found by the Turtle GA, performs equally well on the FPTA, in a DC simulation and in a transient simulation.*

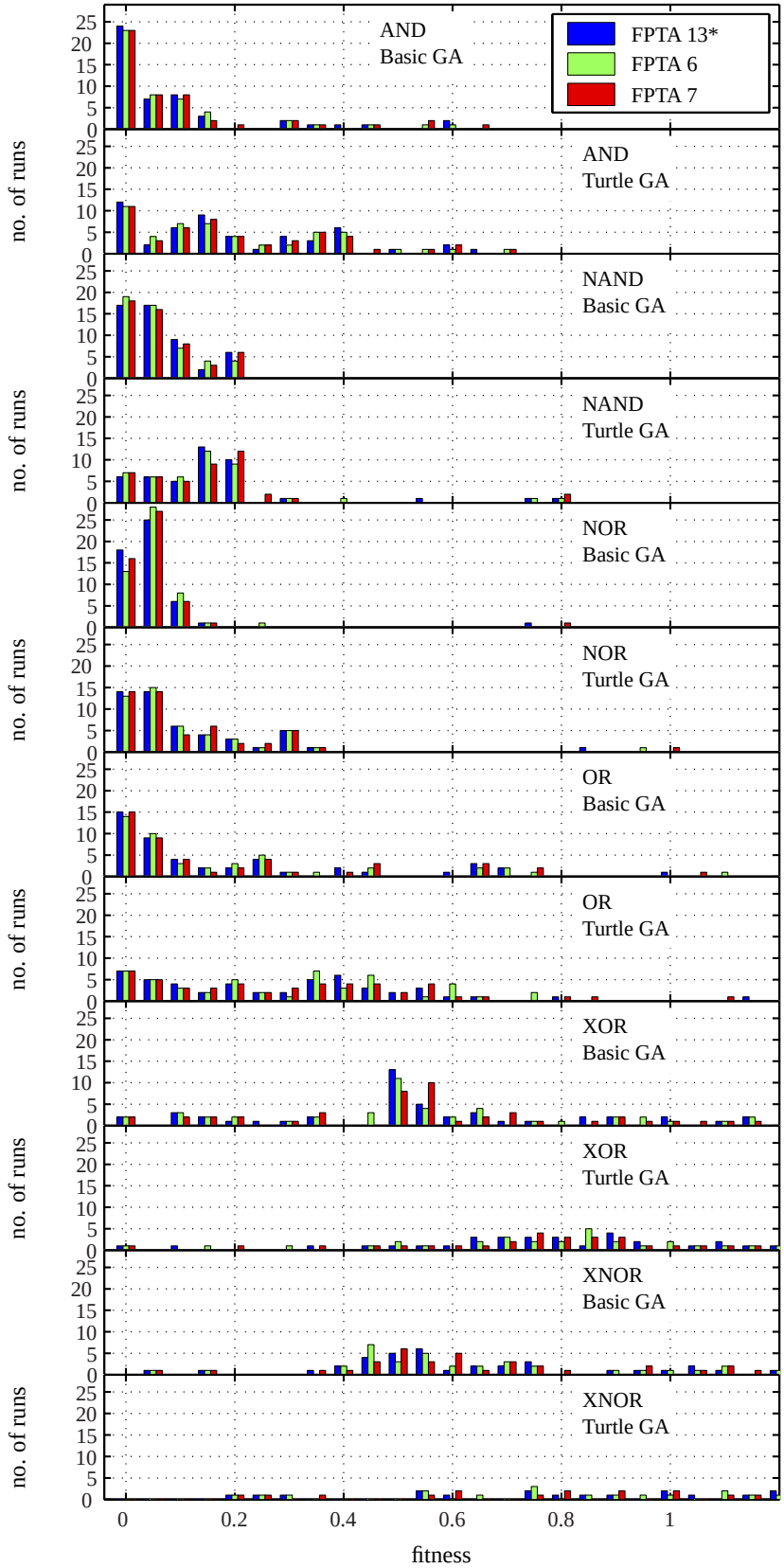


Figure 6.18: *The best individual of all 50 evolution runs is measured on three different FPTAs. Thereby, the 'home' chip is marked with an asterisk. As can be seen from the above histograms, all solutions for the logic gates perform equally well on all three FPTA chips.*

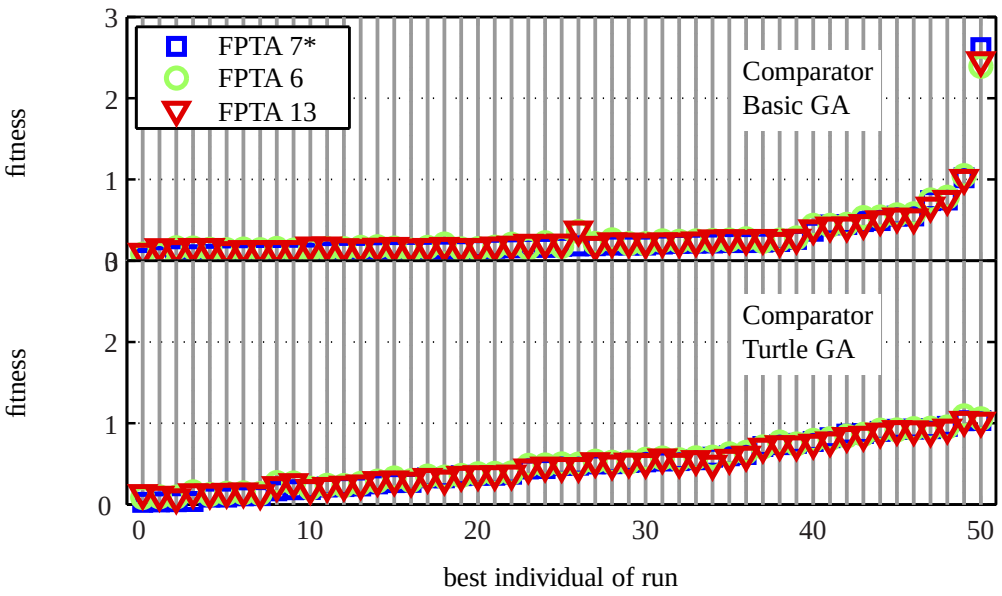


Figure 6.19: *As can be seen from the graphs above, the best solutions for the comparators perform equally well on all three FPTA substrates. Again, the chip, on which the comparators are originally evolved, is marked with an asterisk.*

6.4.3 Performance of the Evolved Circuits on Different FPTAs

testing on three different FPTAs

A further question is whether the circuits, which are evolved on a certain substrate, can be successfully operated on different FPTAs. For this thesis, 3 identical, independent FPTA-in-the-loop evolution systems are available for carrying out experiments. The three different chips are named after the number of the respective host PC: FPTA6, FPTA7 and FPTA13.

Measurement of the Logic Gates

logic gates perform equally well on all FPTAs

The evolution results for the logic gates are obtained with FPTA13. Thus, for comparison, the best circuits of each run are measured on FPTA6 and FPTA7. It is seen from figure 6.18 that the circuits of all 50 evolution runs perform equal on all three transistor arrays. In the case of the logic gates, the results obtained from different chips are depicted in a histogram, instead of a scatter plot, in order to provide a better overview of the numerous plots.

Measurement of the Comparators

comparators perform slightly worse on other FPTAs

The comparators are evolved with FPTA7 and the best individuals of each run are tested on FPTA6 and FPTA13 for comparison. The results for the comparators are depicted in figure 6.19. Unlike in the case of the logic gates, the fitness distribution is shifted to slightly worse values ($+0.05 \ldots 0.1$), corresponding to an additional rms error of about $150 \ldots 250\,\mathrm{mV}$. The reason for this is the importance of the switching point of the comparator. While the transition region at $V_{\mathrm{sweep}} = 2 \ldots 3\,\mathrm{V}$ is not considered for the logic

gates, the no. of samples is even increased near the switching points of the comparators, as described in section6.2.

Based on the latter results, it is suggested that the fabrication variations between different chips are not exploited by the EA for optimizing evolving circuits. On the one hand, this is a positive result, since the solutions feature a certain robustness by not depending on a particular substrate. On the other hand, the hope that the EA could be able to create and optimize circuits by using some kind of 'hidden' features of the hardware is not corroborated.

no hidden parasitic features are exploited

6.4.4 How the Algorithm Does FPTA Tricks

There are two different views on the parasitic effects of evolvable hardware substrates in the research field. On the one hand, some groups try to evolve circuits beyond usual application by exploiting the parasitic effects of a particular substrate as much as possible [30,60,85]. On the other hand, the aim is to evolve robust circuits that are independent from the substrate on which they are evolved, resistant to environmental influences like temperature or pressure or even transferable to other technologies. In the first case, the aim is to maximize the performance for one particular device by using the parasitic effects for tweaking the circuit, although the circuit generally performs poor on other substrates, even of the same kind. In the second case, it is desired that the synthesized circuits work on various platforms and that new design principles can possibly be learned from evolution.

exploiting parasitic effects: yes or no?

Since this thesis is rather following the second approach, it is desired to reduce the

this thesis: rather reducing the influence of parasitic effects

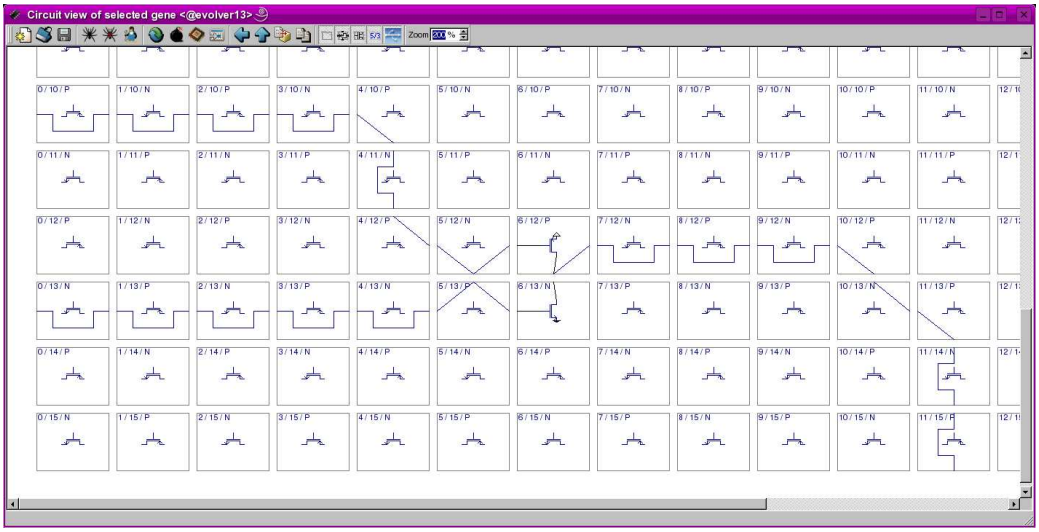


Figure 6.20: *An example circuit (screenshot of the circuit editor of the evolution software), for which the EA mixed the input voltages without previously connecting them to a transistor is depicted. This kind of solution is not wanted, since it does not fail in simulation, but performs different in simulation and on the chip. There are examples of the latter effect for any signals within the circuit, which should—from a designers point of view—rather be used to control the gates of different transistors, than being interconnected.*

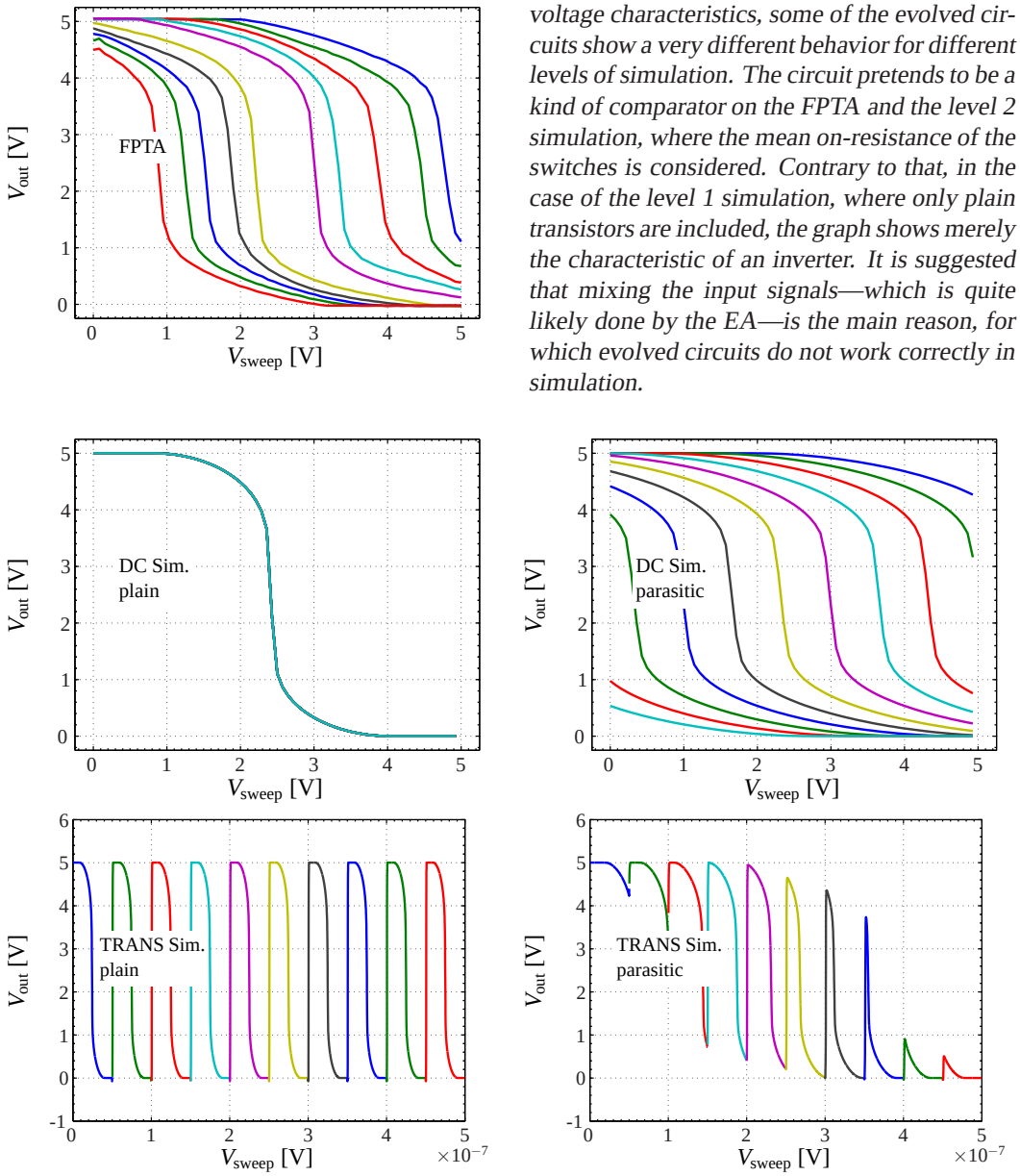


Figure 6.21: *As can be seen from the depicted voltage characteristics, some of the evolved circuits show a very different behavior for different levels of simulation. The circuit pretends to be a kind of comparator on the FPTA and the level 2 simulation, where the mean on-resistance of the switches is considered. Contrary to that, in the case of the level 1 simulation, where only plain transistors are included, the graph shows merely the characteristic of an inverter. It is suggested that mixing the input signals—which is quite likely done by the EA—is the main reason, for which evolved circuits do not work correctly in simulation.*

influence of parasitic effects and to avoid circuits, which are bound to their host substrate. The first step to achieve this, was the development of the *Turtle GA*, with which it is possible to evolve circuits, that can be validated in simulation. Despite this, while carrying out the experiments of this chapter, two important 'features' of the *Turtle GA* are observed, which still lead to circuits, that strongly depend on the architecture—i.e. the parasitics—of the FPTA: first, two gates are interconnected, resulting in a 'floating gate', which leads to the failure of simulation. Although those circuits can be simulated

despite this, parasitic 'features' are observed

by considering all configuration switches, their behavior is different in simulation than on the chip. Most often, such a 'floating gate' configuration depends on the previous state of the FPTA, from which random charges are left. Thus, such circuits are most likely unstable an therefore automatically dismissed by the selection mechanism. Second, the EA connects the circuits inputs without previously routing those signals through a transistor. This is bad, since such circuits do not fail in simulation, but nevertheless perform different in simulation and on the chip. The latter effect is not restricted to the input voltages, but occurs also for different signals within the circuit, which should—from a designers point of view—rather be used to control the gates of different transistors, than being interconnected. An example circuit for such input violation is shown in the screenshot of the transistor array 6.20. As can be seen from the measuring respectively the simulation results of this circuit, shown in figure 6.21, the circuit pretends to be a kind of comparator on the FPTA and the level 2 simulation, while the level 1 simulation (see section 4.4) shows only the characteristic of an inverter. It is suggested that this signal mixing is the main reason, for which evolved circuits do not work in simulation. Additional examples of the discussed effects can be found in appendix B. However, it is interesting to see that, independent from the EA used, the evolved circuits perform equally well on different substrates.

different characteristics in simulation and on the chip

unusual configurations work equally well on different FPTAs

6.4.5 Understanding Schematics of the Evolved Circuits

The developed software framework offers the possibility to automatically generate schematics from circuits, which are evolved with the *Turtle GA*. The according procedure is introduced in chapter 4, section 4.5.2. It is intended to transfer good solutions into a more human readable format and to possibly understand the operation principle of evolved circuits. Furthermore, the task is to find solutions, that are, as far as possible, independent from parasitic effects of the substrate, i.e. the level 1 simulation, where only plain transistors are considered, has to result in a similar voltage characteristic as the measuring on the FPTA. Schematics are generated from the best logic gates and comparators of this chapter and it is found that, indeed, the NAND, AND and the OR gate are working outside the chip in a simulation with only plain transistors. The schematics are shown and discussed in figures 6.22, 6.23 and 6.24. In the case of the NOR, XOR, XNOR and the comparators, the examined examples did not work in a level 1 simulation and therefore, the according schematics are shown in appendix B.

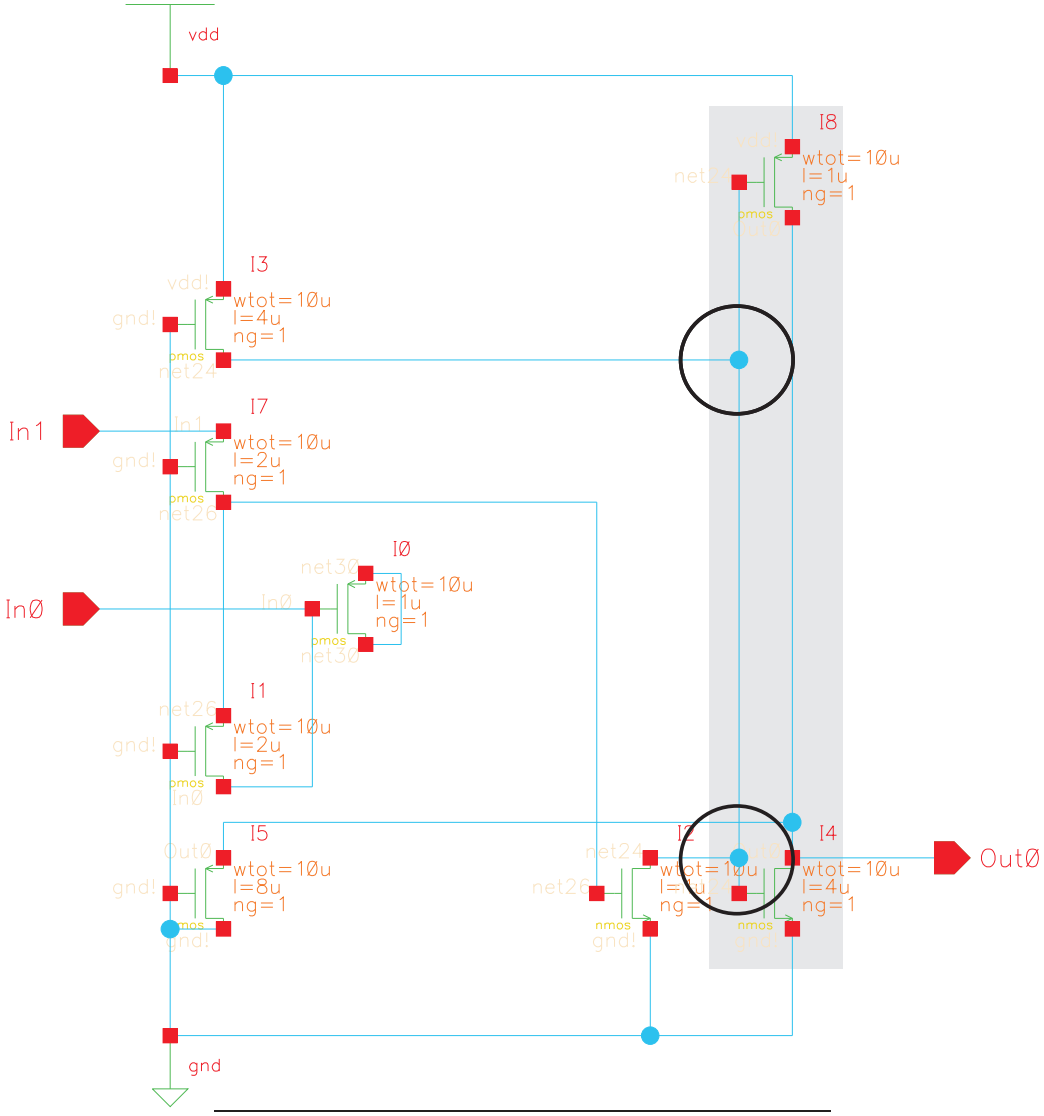
creating schematics from good solutions

It is interesting to see that in all three cases, where the circuits are actually working outside the chip, evolution came up with a similar kind of solution: there is one critical node present in all circuits, which is pulled towards vdd (5 V) by attached PMOS transistors or towards gnd (0 V) by attached NMOS transistors. The balance of strength between those pulling transistors is depending on the input voltages. Subsequently, the logic decision is taken by an inverter, which is always connected to the circuit's output and tuned in a way that it fits the states of the critical node and the correct result is computed. Example voltage outputs for the critical node, the inverter and the final output of the NAND gate are graphed in figure 6.25. Despite the presented circuits are working well in simulation, they might cause problems, if they were embedded into a real-world circuit, since the inputs are often mixed or influenced themselves by the critical node. This is no problem, as long as the inputs are strong enough to drive the input voltage, which is always the case in

NMOS and PMOS in tug of war for the critical node

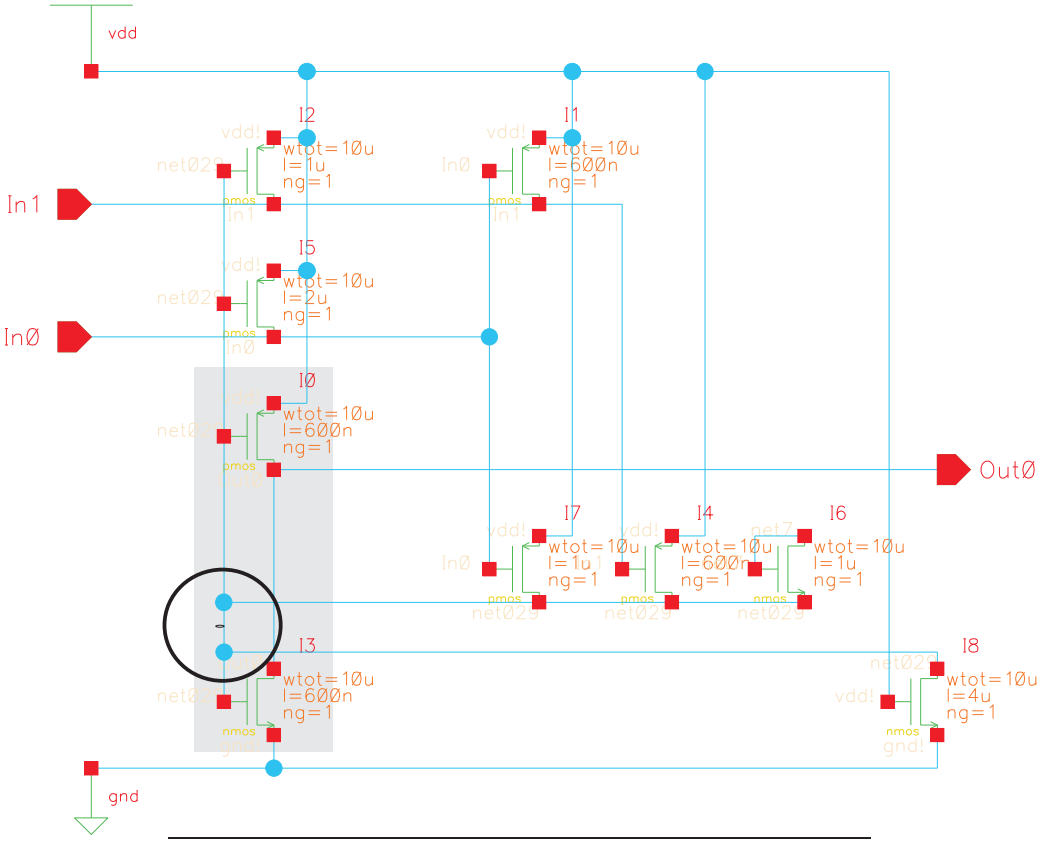
decision of the inverter

issues in real-world circuits



	I0	I1	I2	I3	I4	I5	I7	I8
type	P	P	N	P	N	P	P	P
W/L [μm]	$\frac{10}{1}$	$\frac{4}{2}$	$\frac{9}{1}$	$\frac{15}{4}$	$\frac{14}{4}$	$\frac{1}{8}$	$\frac{5}{2}$	$\frac{10}{1}$

Figure 6.22: *Once more, for the logic OR, the output inverter (I4 and I8) is underlayed with a grey box and the critical nodes, which actually represent the same net, are marked with a circle. In this case, I1, I3, I5 and I7 are always open. Thus, I3 is pulling the critical net towards vdd, while I5 (which is comparably weak) is pulling the output towards gnd. Furthermore, I1 and I7 are mixing the input voltages and are controlling the gate of I2, which is pulling the critical net towards gnd. The higher the mixed input voltage is, the stronger is I2 pulling the critical net towards gnd, resulting in a high output voltage. Contrary to that, if both inputs are low, I2 will become weaker than I3, causing the critical net to be pulled towards vdd and therefore the output to be low. The resulting output voltage characteristic is that of a logic OR.*

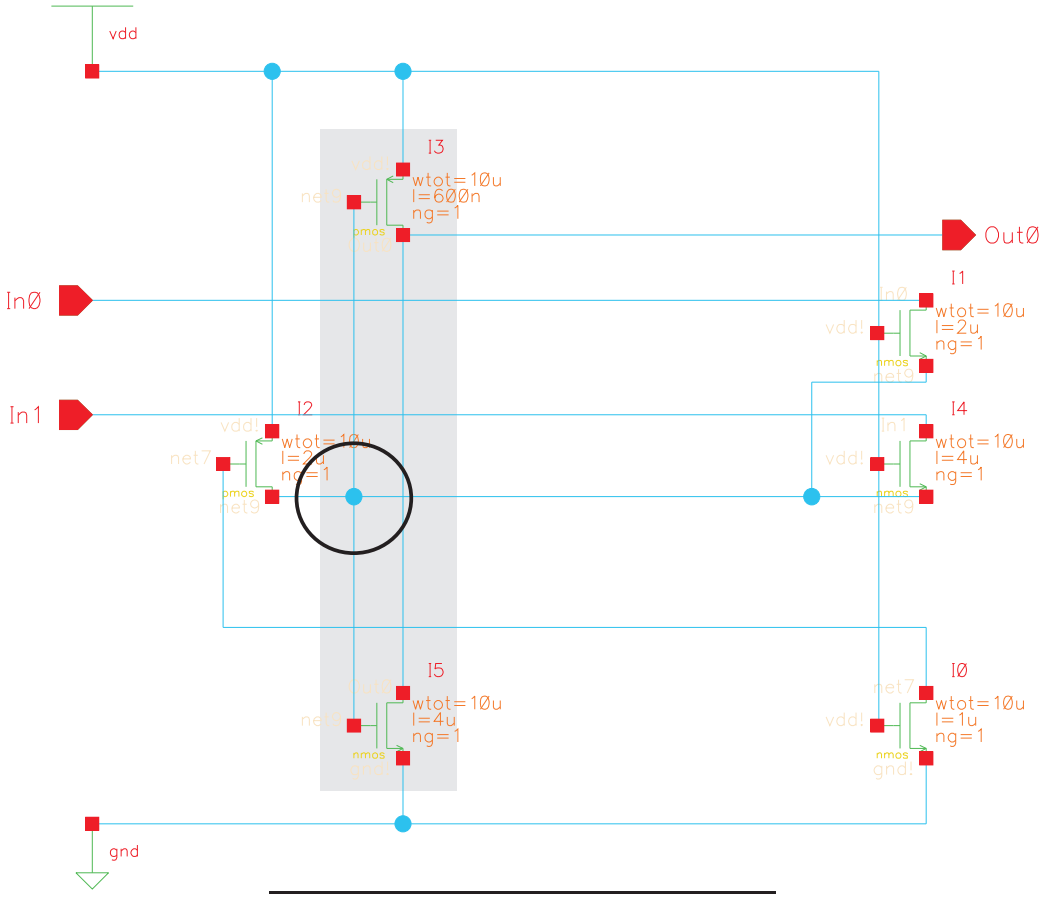


	I0	I1	I2	I3	I4	I5	I6	I7	I8
type	P	P	P	N	P	P	N	P	N
W/L [μm]	$\frac{4}{0.6}$	$\frac{15}{0.6}$	$\frac{14}{1}$	$\frac{15}{0.6}$	$\frac{15}{0.6}$	$\frac{3}{2}$	$\frac{5}{1}$	$\frac{15}{1}$	$\frac{4}{4}$

Figure 6.23: *Again, in the case of the AND gate, the output inverter (I0 and I3) is underlayed with a grey box and the critical node is marked with a circle. In this case, the critical node is, on the one hand, pulled towards vdd by the transistors I1, I4 and I7 with variable strength, depending on both input voltages. On the other hand, I8 is always open and pulls the critical node towards gnd. Thereby, as long as one of the two inputs has a low voltage, the voltage of the critical node is rather high, I2 and I5 are closed and the output is low. Contrary to that, if the inputs are high, I1, I4 and I7 will be closed and the critical node voltage is pulled to gnd. Additionally, the latter effect is emphasized, since I2 and I5 are opened and are pulling the inputs further towards vdd (which should be rather deprecated from a designers point of view), causing the output to be high. Hence, the resulting output voltage characteristic is that of an AND gate.*

simulation and also on the FPTA, where strong enough input OPs are applying the input voltage pattern. However, the latter dependency is not wanted in practice, where such circuits are usually designed with a high input resistance, i.e. the inputs are connected only to gates.

To conclude, it can be learned from the schematics that the EA seems always to follow the same strategy for finding solutions: first, a lot of transistors with various sizes are inserted into the circuit until the output characteristic is changing in a desired way. Second,



	I0	I1	I2	I3	I4	I5
type	N	N	P	P	N	N
W/L [μm]	$\frac{8}{1}$	$\frac{14}{2}$	$\frac{3}{2}$	$\frac{15}{0.6}$	$\frac{15}{4}$	$\frac{3}{4}$

Figure 6.24: *In the case of the NAND gate, the transistors I1 and I4 are always open and used for mixing both input voltages. The inverter (I3 and I5) at the output is underlayed with a grey box and the critical node is marked with a circle. Depending on both input voltages, the critical node is, with a certain strength, pulled towards vdd by I2 (which is opened by I0), thus, as long as one of the inputs has a low voltage, the critical node is also below the switching voltage of the inverter and therefore the output Out0 is vdd. If both inputs have a sufficiently high voltage, Out0 becomes gnd, resulting in the output characteristic of a NAND gate.*

it is very likely that a NMOS and a PMOS transistor are combined to an inverter, which pulls the output towards vdd or gnd, depending on the voltage characteristic of a critical node (a critical net). Finally, the circuit is tweaked by balancing the W/L ratio (strength) of the transistors, which define the voltage of the critical node.

6.5 Concluding Remarks

comparing the Basic GA and the Turtle GA

The experiments in this chapter show that the *Basic GA* and the *Turtle GA* are able to

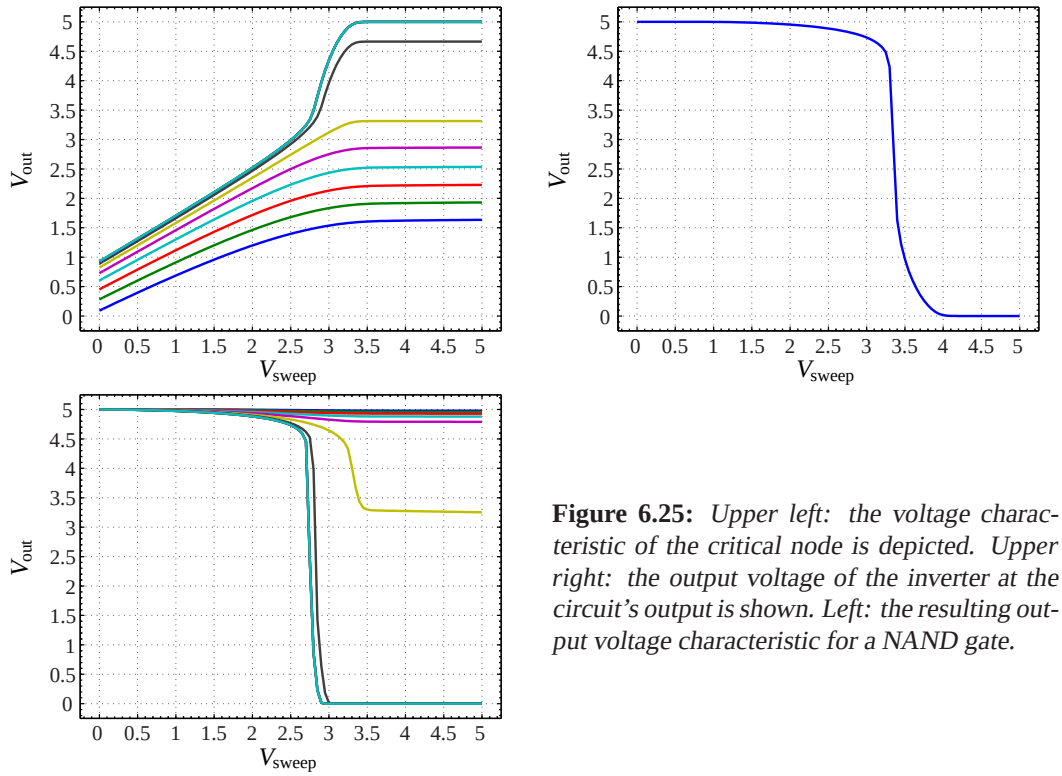


Figure 6.25: *Upper left: the voltage characteristic of the critical node is depicted. Upper right: the output voltage of the inverter at the circuit's output is shown. Left: the resulting output voltage characteristic for a NAND gate.*

successfully synthesize transistor circuits—in this case logic gates and comparators—on the FPTA. Furthermore, the best solutions perform equally well (or at least similar in the case of the XNOR) and use a comparable amount of transistors as manually made circuits, that are realized on the configurable transistor array. The quality of the output voltage characteristics of the found solutions will not get worse, if they are measured on other substrates, than the one on which they are evolved.

achievements of the Turtle GA

Additionally, it is possible to extract SPICE netlists from the evolved circuits and validate them in simulation, by using the *Turtle GA*. It is also possible to create schematics from the best results, in order to facilitate understanding of the found solutions.

general observations

Concluding, the *Basic GA* is slightly more successful in tweaking the performance of certain circuits on the chip than the *Turtle GA*, although it lacks the ability of creating circuits that can be transferred to other technologies or which are easy to understand. Furthermore, as the example in section 6.4.4 shows, it is relatively easy to generate a 'good looking' output on the FPTA, based on an elsewhere useless circuit structure. Due to these results, further experiments of this thesis are carried out with the *Turtle GA*.

An additional interesting perception of this chapter is that the difficulty level of finding good solutions for a given problem is only depending on the complexity (no. of necessary transistors and connectivity) of the desired circuit (section 6.4.1), which is not the case for a human designer, who creates solutions with the knowledge about the behavior of the transistors and how they are supposed to be connected. Thus, it is suggested that the routing architecture of the current chip is the main limiting factor for further improvement of the circuits, since either two paths are quite likely shorted or a huge number of routing switches is necessary to connect two distant transistor terminals.

Chapter 7

Multi-Objective Optimization of the Transistor Circuits

In this chapter, a multi-objective (MO) EA, based on the work, presented in the previous chapter and in [91], is developed and successfully applied for the synthesis of comparators, oscillators and OPs on the Heidelberg FPTA [50]. Hence, it is referred to as the MO-Turtle GA, throughout the remainder of this thesis. A multi-objective approach is chosen, in order to be able to include the various specifications of e.g. an operational amplifier into the process of circuit synthesis. Moreover, the presented algorithm is designed to preserve the diversity within the population troughout the course of evolution and is therefore able to efficiently explore the design space. In the case of the comparators and the OPs, the evolved circuits are proven to work on the chip as well as in simulation outside the FPTA. Additionally, the results for the comparators are compared with the non-MO results from the previous chapter. Automatically generated schematics of good solutions are presented and their characteristics are compared to those of basic manually created OPs. Furthermore, oscillators are evolved with the multi-objective approach, which was previously not achieved. The latter oscillators are an example of a truly multi-objective result, since it is possible to harvest solutions with different frequencies from successfull evolution runs. Nevertheless, the synthesis of OPs is the most challenging task in this chapter.

To date, as to the authors knowledge, only a few analytic solutions for analog design automation are available. Examples, in which previously known topologies are tested while the sizing of the components is done by an optimization algorithm, are given in [10, 33]. In a great number of approaches, the topology is also to be found automatically, therefore, developmental strategies [45, 46, 74, 94] or heuristic interconnection of building blocks [48] are applied, in order to deal with the high complexity of amplifiers. An alternative possibility is to choose a multi-objective evolutionary algorithm [17, 18], in

order to face the fact that, for the solution of almost every complex problem, numerous variables have to be taken into account for optimization. Operational amplifiers, as well as other transistor circuits, found to this point by means of hardware evolution in conjunction with multi-objective optimization (MO), are reported in [32, 92, 103, 109]. Other results, obtained with the FPTA, can be found in [25]. Furthermore, a multi-objective approach provides the designer with a variety of choices instead of only one more or less good solution. This is a great advantage, especially in cases where trade-offs have to be made, e.g. between gain and speed of an amplifier.

7.1 The Multi-Objective Evolutionary Algorithm

Since the evolution of transistor circuits is a challenging task, where numerous variables have to be taken into account for optimization, the *Turtle GA* is extended with a multi-objective strategy, first proposed in [27], for the experiments in this chapter. The *Turtle GA* itself is introduced in chapter 6, section 6.1.2. This allows for a separate evaluation and optimization of different properties of the evolving circuits, which would not be possible with a single objective algorithm. The multi-objective *Turtle GA*, referred to as the *MO-Turtle GA* throughout the remainder of this thesis, consists of a non-dominated sorting algorithm and a crowding distance measure, which are described in the following and are based on those from the non-dominated sorting genetic algorithm, presented in [18, 19]. Using an MO approach offers two important advantages: first, the population is of great diversity during the whole course of evolution, for the reason that individuals with a bad over-all performance survive as long as they are superior in at least one objective. Thus, on the one hand, crossover gains importance by combining differently specialized individuals and, on the other hand, premature convergence of the population is widely avoided. Second, due to the presence of 'specialist' individuals for each objective in the resulting population, numerous results—representing trade-off solutions for the different objectives—can be harvested from the non-dominated front (NDF) instead of only one.

simultaneous optimization of different circuit properties

maintaining diversity of the population

numerous trade-off results

7.1.1 Variation Operators of the MO-Turtle GA

The variation operators of the *MO-Turtle GA*, namely the *Random Wires* mutation and the *Implanting Block of Cells* crossover, are those from the *Turtle GA* and are reported in the previous chapter, section 6.1.2 and in [91]. The implementation of both operators is adapted to the FPTA's architecture and, as a remainder, briefly described in the following. Note that the most important feature is that the resulting circuits contain no floating nodes, thus, can be validated off-chip.

random wires mutation

Random Wires (Mutation). The mutation operator consists of the create mode and the erase mode. The create mode connects random nodes within the FPTAs transistor array and thereby randomly inserts components into the active circuit. Contrary to that, the erase mode randomly disconnects nodes and removes transistors. The mutation operator is carried out recursively until the resulting circuit contains no dangling nodes and no floating transistor terminals. The width and length of all active transistors is mutated due to a configurable probability.

implanting cells crossover

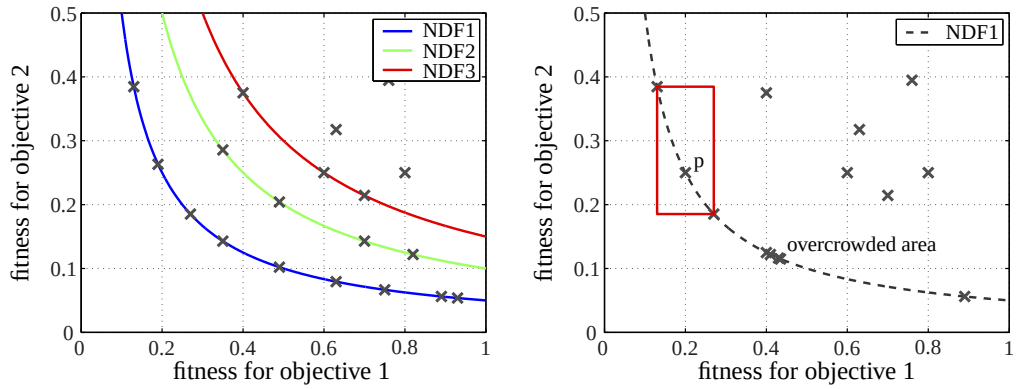


Figure 7.1: *Left: An example set of individuals—which shall be optimized for two objectives—is depicted. The first three NDFs, obtained by evaluating equations 7.1 and 7.2, are drawn in. It is expected that the NDFs propagate towards better fitness values throughout evolution. Additionally, the rank of the NDF is equal to the level of non-domination for each individual of the respective NDF. Right: In this example, the individuals are not distributed uniformly over the NDF. Therefore, in order to be able to drive evolution towards such a uniform distribution, a partial order of the individuals within an NDF is defined by the crowding-distance c_{dist}. The value of c_{dist} for an example individual p is derived from the distance to the next neighbors of p.*

Implanting a Foreign Block of Cells (Crossover). The *implanting* crossover operator is carried out in two stages. The first stage exchanges randomly sized and positioned rectangular blocks of transistor cells between two randomly selected individuals. While the size of both blocks has to be the same for each individual, the positions of the blocks may differ. Since this operation in general breaks the layout of both previously intact circuits, the second stage fixes the occurring floating nodes by executing the *random wires mutation* operator for each of them. Thus, again, the resulting circuits contain no floating nodes.

7.1.2 Non-Dominated Sorting and Crowding Distance

MO extensions

In order to include multi-objective optimization into the current evolutionary algorithm, a new evaluation and selection scheme has to be implemented. In the case of MO, separate fitness values are assigned to the individuals for their performance in different tasks, resulting in a vector of fitness values, instead of only one aggregated fitness value. Note that in the MO language, the different tasks are referred to as *objectives*, thus, this is also the case throughout the remainder of this thesis. As a consequence of this, it is no longer possible to decide whether an individual is better or worse than another, by simply comparing their single fitness value. Therefore, before selection, a ranking of the individuals is achieved by applying the *non-dominated sorting* algorithm. A fast implementation of the non-dominated sorting algorithm, is shown in pseudocode 7.1. Note, that for simplicity in some cases the fitness value for a certain objective is referred to as objective value, which means the same.

ranking and selection by level of non-domination

Non-Dominated Sorting All individuals are classified by calculating their level of non-domination, as shown in figure 7.1, due to their objective values p_i. An individual p is said to dominate q, denoted by $p \preceq q$, if and only if p is partially less than q (equation. 7.1).

$$\forall i \in (1,\ldots,n),\, p_\mathrm{i} \leq q_\mathrm{i} \quad \wedge \quad \exists i \in (1,\ldots,n) : p_\mathrm{i} < q_\mathrm{i} \tag{7.1}$$

$$\mathrm{NDF} := \{p \in P \mid \nexists p' \in P : p' \preceq p\} \tag{7.2}$$

All p satisfying equations 7.1 and 7.2 provide the first non-dominated front NDF_1. The succeeding NDFs are found by removing the individuals of NDF_k from the population $P' = P \setminus \mathrm{NDF}_\mathrm{k}$ and by recalculating equations 7.1 and 7.2 for the new population P' until $\mathrm{NDF}_{\mathrm{k+1}}$ is empty. Consequently, the level of non-domination replaces the aggregated fitness value, which is used in the previous chapter, for selection. The main advantage of the non-domination measure is the fact that also partly good solutions survive, thus, the search space is more efficiently sampled and premature convergence is avoided.

As yet, there are possibly a great number of solutions within the same NDF, thus, it cannot easily be decided which one should get a higher probability to survive. Due to this fact, a second measure is introduced: the *crowding distance*. The algorithm for calculating this value is also given in pseudocode 7.2.

maintaining the diversity of the population

Crowding Distance The crowding distance (c_dist) is a measure for the density of solutions within the vicinity of a particular individual p within the fitness landscape (figure 7.1) and is calculated for the members of each NDF respectively. All objective values are considered for calculating the quantity c_dist which represents an average distance to the nearest neighbors of p and is assigned to each individual of the respective NDF. Therefore, since the aims are to provide a great diversity within the population and to steer the evolution towards a uniform distribution of the individuals over the NDF, c_dist is used as an additional ranking criterion for the individuals within the respective NDF.

Lexical Order of the Objectives

There is some work in the field, where the objectives themselves are ranked due to their suggested importance. This ranking is considered during the calculation of the NDFs, by omitting objectives of less importance in case there is no solution with a sufficiently good performance for the objectives of higher importance. Such lexical ranking is considered to be useful for saving computation time of the non-dominated sorting algorithm and for emphasizing good results in major objectives. Despite this, such explicit lexical order is not employed in this thesis due to the following reasons: first, an importance ranking of the objectives is nevertheless inherently present, since, it is not possible to assign another than the worst fitness for the objective of minimizing e.g. settling-time as long as the objective, that aims at producing a voltage step, is not fullfilled. Second, to the authors opinion, the idea of multi-objective optimization will be violated, if it is necessary to define problem specific thresholds for the activation of additional objectives. Finally, one cannot be sure to not exclude possible pathways within the search space, that lead to a good solution.

inherent ranking of the objectives

Algorithm 7.1: The non-domination sorting algorithm, which is used to classify the individuals into non-dominated fronts, according to their level of non-domination. This level depends on how many other individuals are dominated by an individual p and on how many other individuals dominate p. If no other individual dominates p, p belongs to the first NDF.

```
procedure NONDOMINATEDSORTING( )
    for p ← 1 to population size do
        n_p = 0                                   // no. of individuals, which dominate p
        S_p = 0                                   // list of individuals, which are dominated by p
        for q ← 1 to population size do
            assume p dominates q
            assume q dominates p
            for obj ← 1 to no.of objectives do
                if p_obj > q_obj then
                    p does not dominate q!
                else if p_obj < q_obj then
                    q does not dominate p!
                end if
            end for
            if p still dominates q and q still dominates p then
                neither p dominates q, nor q dominates p!
            end if
            if p dominates q then
                add q to S_p = 0
            else if q dominates p then
                increment n_p = 0 by 1
            end if
        end for
        if n_p equals 0 then
            add p to NDF_0
        end if
    end for                                       // first NDF is found!

    i=0
    while NDF_i is not empty do
        for all p in NDF_i do
            for all q in S_p do
                decrement n_q by 1
                if n_q equals 0 then
                    add q to NDF_i+1
                end if
            end for
        end for
        increment i by 1
    end while                                     // all NDFs are found!
end procedure
```

Algorithm 7.2: The algorithms which calculates the ranking of the individuals within the NDFs, by assigning a value for the *crowding distance* (c_{dist}) to each of them. c_{dist} is a measure for the density of solutions within the vicinity of a particular individual p within the fitness landscape.

```
procedure ASSIGNCROWDINGDISTANCES( )
    initialize c_dist of all individuals with 0
    for all non-dominated fronts NDF_i do
        for obj ← 1 to no. of objectives do
            sort individuals of NDF_i by objective obj
            normalize objective values
            assign c_dist(ind_0 = #objectives
            assign c_dist(ind_#individuals = #objectives        // extrema are preserved
            for i ← 1 to no. of individuals − 1 do
                c_dist(ind_i) += c_dist(ind_i-1) + c_dist(ind_i+1)
            end for
        end for
    end for
end procedure
```

Algorithm 7.3: The tournament selection scheme, which is used for creating the new population from the current repository generation, is described in pseudocode. The randomly picked competitors have to compete in two disciplines: the level of non-domination and, in case they belong to the same NDF, the convergence of their fitness, which is calculated according to equation 7.3. Thereby, the tournament size is 3.

```
function SELECTINDIVIDUAL( )
    randomly select champion from repository population
    for i ← 1 to tournament size − 1 do
        randomly select competitor from repository population
        if NDF of competitor < NDF of champion then
            champion = competitor
        end if
        if NDF of competitor = NDF of champion then
            if fitness convergence of competitor > convergence of champion then
                champion = competitor
            end if
        end if
    end for
end functionreturn champion
```

7.1.3 Selection Scheme

Two equally sized populations are used in the case of MO: first, the repository population, which is directly created from the unchanged individuals of the intermediate population in the first step of the MO algorithm. In this case, as described in the previous section (7.1.4), the decision, which individuals survive is based on their level of non-domination and their crowding distance c_{dist}. Second, the new population, which is generated by applying mutation and crossover to selected individuals from the repository generation. In the latter case, tournament selection with a tournament size of 3 is used as selection mechanism. Thereby, champion and challengers are randomly picked from the repository population and the one with the higher non-domination level wins. If both competitors feature the same level of non-domination, hence, belong to the same NDF, the convergence of their fitness

tournament selection

$$\text{conv.} = \sum_{\text{objectives}} \frac{\text{fitness}(\text{gen}_{i-5})}{\text{fitness}(\text{gen}_i)}. \tag{7.3}$$

is additionally taken into account. The convergence is calculated from their previous fitness values using equation 7.3. Furthermore, the selection mechanism for creating the new generation is described in pseudocode 7.3.

7.1.4 The Evolutionary Step

As can bee seen from figure 7.2, three populations are used in the evolution loop: a repository population *RP*, a new population *NP* of size *N* and an intermediate population *IP* of size 2*N*. The first step is to initialize the algorithm by randomly generating individuals

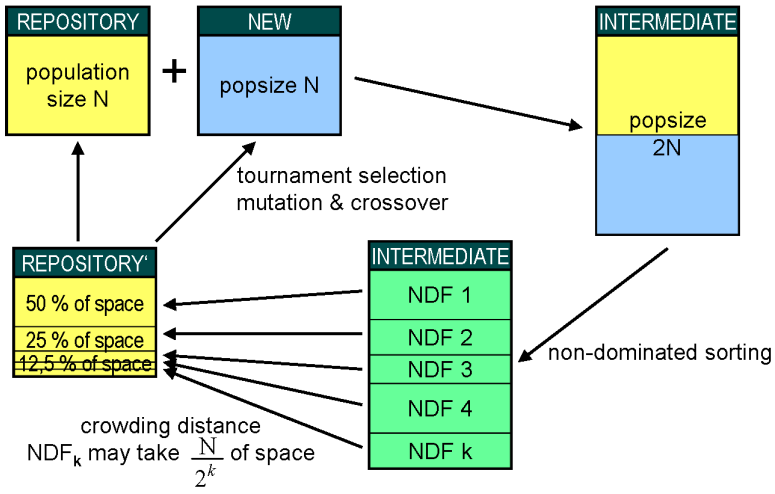


Figure 7.2: *As can be seen from the figure, three populations are used in the evolution loop. NDFs and crowding distance are calculated for the intermediate population. Subsequently, the repository population is created and thereby, NDF_k is allowed to occupy at most $\frac{1}{2^k}$ of the available space. Finally, the new population is created from the repository by applying selection and the genetic operators.*

for *IP*, to perform measuring and to calculate fitness values for all objectives of the latter individuals. Subsequently, the evolutionary loop is started by performing non-dominated sorting and by calculating crowding distances (c_{dist}) for all individuals of *IP*. Thus, a non-domination level (membership in the according NDF) and a value for c_{dist} is assigned to each individual of *IP*. The next step is to fill *RP* with the best individuals of *IP*. Thereby, NDF_k is allowed to occupy at most $\frac{1}{2^\text{k}}$ of the available space in *RP*. In case the size of NDF_k is less than or equal to the available space, the whole NDF_k is copied to *RP* and the remaining free space is additionally provided to the successing NDF. If an NDF contains more individuals than space provided in *RP*, only those individuals with the best c_{dist} ranking are copied. Consequently, since the first NDF is allowed to occupy 50 % of the space in *RP* and, due to the c_{dist} ranking, the superior individuals of each objective survive—as long as the size of *RP* is at least twice the number of objectives—it is guaranteed that the best solutions always survive. Once the repository population is created, the new population is generated from this repository by using tournament selection, described in the following subsection 7.1.3, and by applying mutation and crossover to the selected individuals. Subsequently, *NP* is measured and evaluated and the next evolutionary step (generation) is prepared by combining *RP* and *NP* to a new intermediate population $IP = RP \cup NP$.

MO ranking and selection scheme

keeping the best for each objective

7.2 A First Benchmark: the Comparators

experimental setup

As a first benchmark, the evolution of comparators with the *MO-Turtle GA* is tackled, in order to compare the results with those from the previous chapter 6. The consequences of employing the multi-objective approach are investigated and compared with the *Basic GA* and the *Turtle GA*, where an aggregated fitness value is used. Again, a total of 50 evolution runs is carried out, each with a population of 50 individuals, which are processed for 20.000 generations.

7.2.1 Experimental Setup

aggregated fitness is replaced with a fitness vector

The setup for the test modes, the simulations and the fitness calculation is identical to the experimental setup for the comparators, described in the previous chapter, section 6.2. Although, in this case, the aggregated fitness value is replaced with a vector of 3 fitness values. Thereby, both test modes deliver one fitness value respectively, which assesses the deviation from the respective target voltage pattern and the third fitness value represents the ressource consumption (no. of transistors used).

TM	objective	fitness	description
TM_1	dev. from V_{tar}	min.	calc. with equation 6.4
TM_2	dev. from V_{tar}	min.	inverse inputs, calc. with equation 6.4
—	resource consumption	min.	sum of transistors used

Table 7.1: *An overview of all TMs and their corresponding objectives. In TM_2, the inputs are exchanged, thus, in the case of the comparators, the output voltage pattern is inverted. The aim is to minimize the fitness.*

Furthermore, the same EA parameters are used as in the previous comparator experiments, whereas the new selection scheme (subsection 7.1.3), the non-dominated sorting algorithm and the c_{dist} measure (subsection 7.1.2) of this chapter is used. A short overview over the test modes is given in table 7.1.

7.2.2 Results and Conclusions

On-chip measuring and simulation

In the first histogram 7.3, the individuals of each run with the best sum of MO fitness values are depicted and the axes have the equal scaling as in histogram 6.10, in order to be able to compare the results to the experiments from the previous chapter, where only one aggregated fitness value is used. As can be seen from histogram 7.3, the distribution is flatter, than in the non-multi-objective experiments, i.e. the solutions are almost equally distributed between fitness values of $0.1 \ldots 1.5$ (rms error $= 50\,\text{mV} \ldots 1.75\,\text{V}$) and there are some solutions present with a fitness of up to 3 (rms error $= 2.5\,\text{V}$). Thus, from a single fitness value's point of view, the over-all performance of the MO approach seems to be worse, than in the non-MO case, at least if the algorithms are run for the same number of generations. Nevertheless, the best runs ended up in the same regions of fitness for both approaches. Therefore, it is supposed that the multi-objective approach converges slower than the non-MO approach, but will produce as many good solutions if it is run for a greater number of generations.

flatter distribution of resulting runs

slower convergence in the case of MO

Despite the fact that, as shown in figure 7.4, the rms errors for the measuring on the chip are shifted towards worse values, the results for the DC and transient simulations are not worse than those obtained with the *Turtle GA* and the *Basic GA*. Once again, it can be seen that a good performance on the FPTA is not necessarily correlated with a good performance in simulation. Output voltage characteristics of the best individual are depicted in figure 7.5.

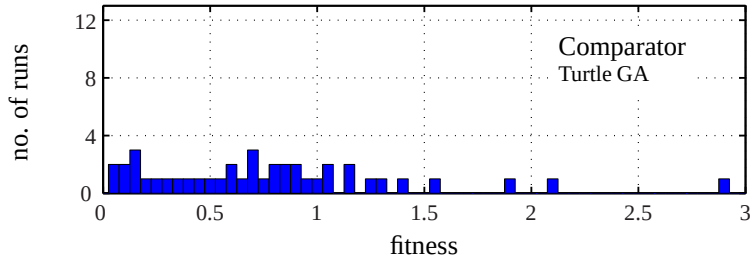


Figure 7.3: *The individuals of each run with the best sum of MO fitness values are depicted in the histogram and the axes have the equal scaling as in histogram 6.10. As can be seen from the graph, the distribution is flatter , than for the non-multi-objective experiments and ranges up to a fitness of 3. Thus, from a single fitness value's point of view, the over-all performance of the MO approach seems to be worse, than in the non-MO case. Nevertheless, since the best runs ended up in the same regions of fitness for both approaches, it is supposed that the multi-objective approach converges slower than the non-MO approach, but will produce as many good solutions if run for a greater number of generations.*

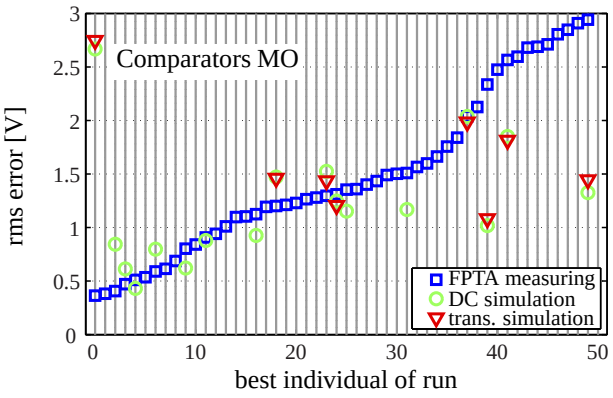


Figure 7.4: *It can be seen from the graph, that the performance of the circuits is again worse in simulation than for the measuring on the chip. Nevertheless, the results for the DC and transient simulations are not worse than those obtained with the Turtle GA and the Basic GA. Thus, it can be seen once again that a good performance on the FPTA is not necessarily correlated with a good performance in simulation. Finally, there are at least 6 solutions, that perform similar in simulation and on the chip.*

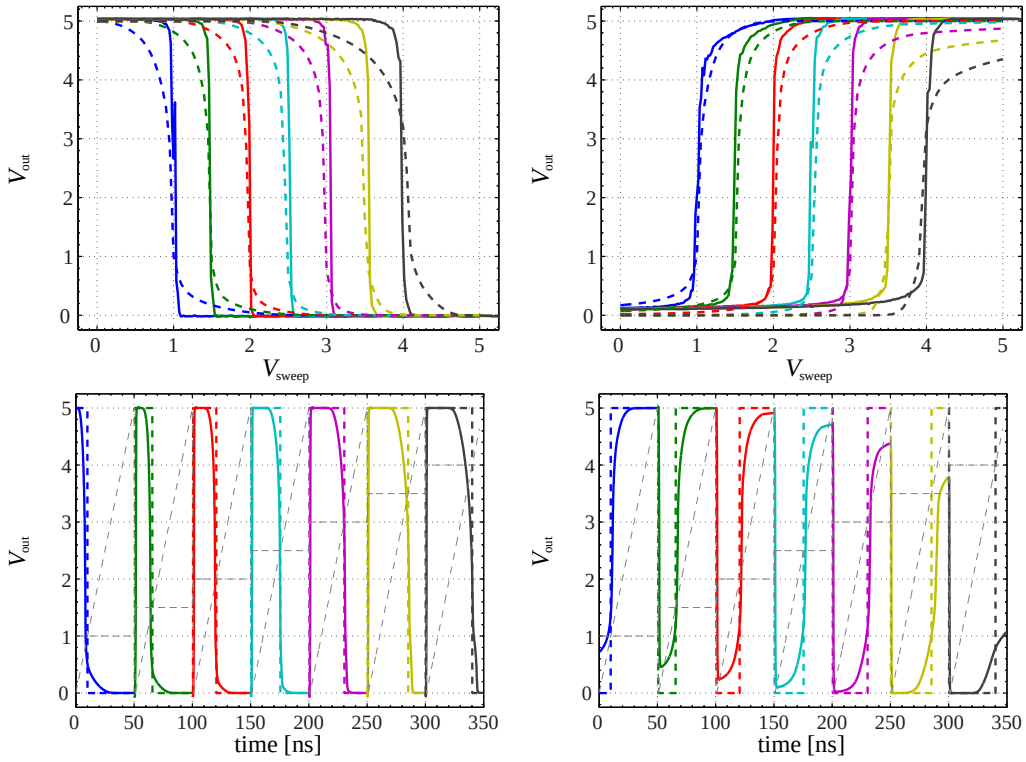


Figure 7.5: *Output voltage characteristics for the MO comparators are shown. Top: DC simulation and on-chip measurement. Bottom: transient simulation.*

Pointing out the advantages of the multi-objective approach

The strength of the multi-objective approach is to make it possible to successfully evolve circuits, where numerous—maybe even contradicting—objectives need to be considered. In order to achieve this, it is necessary that, on the one hand, a population of individuals of great diversity in each objective is maintained by the algorithm and, on the other hand, the individuals are continuously improved in all objectives. The great diversity of the population is, mainly in those objectives, where the circuit improves comparably slow, important for avoiding premature convergence. As can be seen from figure 7.6, in the case of the *MO-Turtle GA*, the fitness for all objectives is uniformly distributed over the feasible range, while, in the case of the *Turtle GA*, most solutions are clustered either near the best solution, or near the worst solution. Further, it can be seen from the depicted example experiment that, in the non-MO case, there are no individuals in the branch 'better fitness for TM_2, worse fitness for TM_1', which indicates premature convergence in TM_1 (objective 1). The scenario is similar, in case the number of transistors used is plotted over the fitness of TM_1 and the fitness of TM_2, respectively. Concluding, the *MO-Turtle GA* successfully found solutions for the comparator and, at the same time, achieved to maintain great diversity for all objectives within the population. This is a promising result, since the aim is to tackle problems with higher dimensional objective spaces in the following experiments.

uniformly distributed fitness values: great diversity

avoiding premature convergence

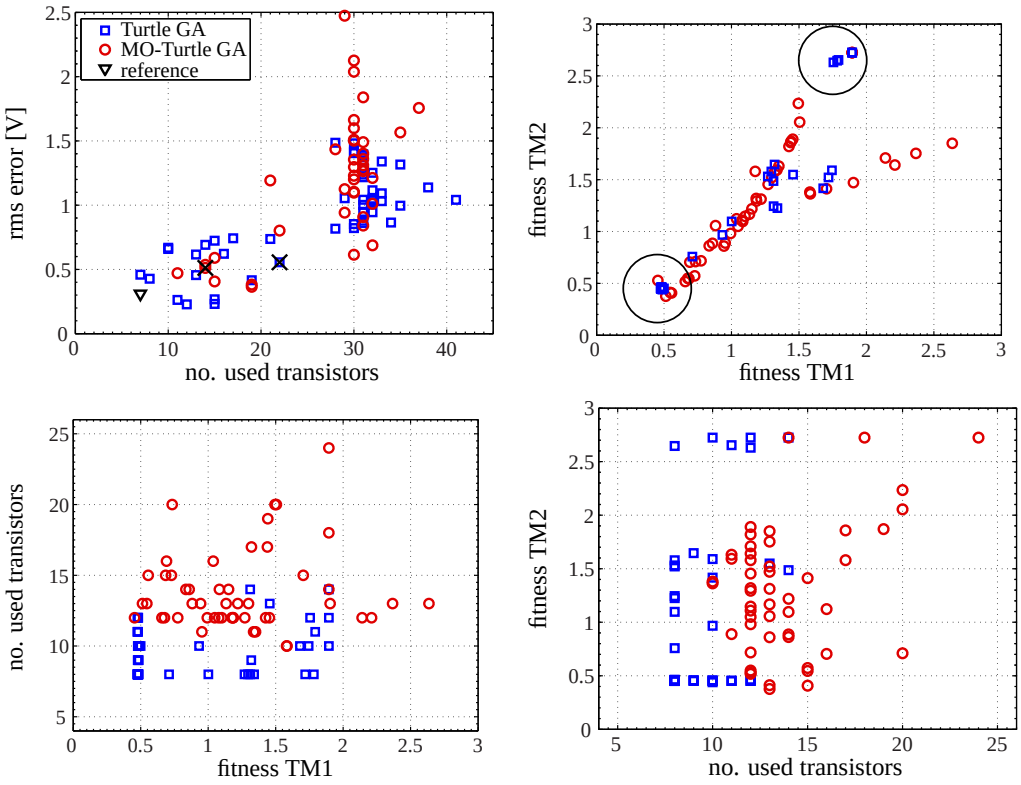


Figure 7.6: *The first picture shows the over-all rms over the resource consumption, while in the other three pictures the marked with a cross experiments are depicted as an example: each possible objective over each possible objective.*

7.3 A Truly Multi-Objective Result: Oscillators from Scratch

no predefined gain stages

The successfull evolution of oscillator circuits from scratch has not yet been achieved with any single-objective approach on the FPTA. To the author's knowledge, there is even no other example—except for the work in [3, 93]—for successfull evolution of oscillators in the research field, where no predefined inverting or gain stages are provided to the algorithm. Since there are no such predefined structures used on the FPTA, the term *'evolution from scratch'* is emphasized. Furthermore, it is a challenging task to evolve oscillating circuits, due to the fact that, at some point, the EA has to modify the circuit in a way that it in fact starts to oscillate. Rendering things more difficult, in the case of an oscillating output, there are some important consequences for the fitness calculation: first, a possibly present transient effect has to be considered. Second, it is usually neither the case that the circuit immediately starts oscillating at the desired frequency, nor that the oscillation is stable. Third, the phase is shifted for each measuring, which would, in the case of $\phi = \pi$, result in the worst fitness for the desired target output voltage, if not taken into account. Finally, amplitude and frequency have to be considered independently. Concluding, if the performance of the candidate circuits in all those objectives are to be considered in solely one aggregated fitness, this value will not decrease monotonically with an improving solution. Hence, oscillators are a predestinated problem for multi-objective optimization.

implications of oscillation

7.3.1 Experimental Setup

neither input present, nor fixed constraints for the output

An area of 10×10 transistor cells is provided to the evolving circuit and the population size is 100 individuals, which are evolved for 5000 generations. A total of 100 evolution runs is carried out. Crucial for this setup is the fact that there is no input and only one output present in the evolving circuit and further, the output is not bound to fixed constraints, in terms of a target voltage pattern. Rather the output voltage behavior, than a fixed output voltage pattern is assessed with the test modes and their respective fitness functions, as described in the following subsection. The FPTA setup is shown in figure 7.7.

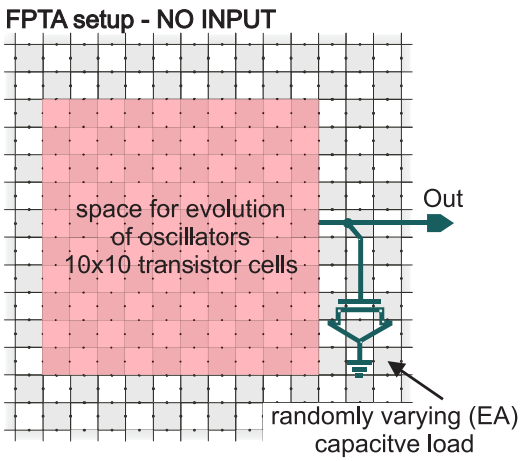


Figure 7.7: *Left: the FPTA test bench for the evolution of oscillators is graphed. An area of* 10×10 *transistor cells are provided to the algorithm. A capacitive load is realized with the transistors on the chip and attached to the circuit's output. Thereby, the advantage is that this capacitive load can be randomly varied during evolution by the EA.*

7.3.2 Test Modes and Fitness Calculation

One testmode, containing 480 samples with a sampling frequency of 10 MHz, is used for the synthesis of oscillators, hence, possible output frequencies of the evolving circuits are ranging within 21 kHz..2.5 MHz. In this setup, rather than comparing the measured output voltage samples to a predefined target voltage pattern, more general characteristics of the output are evaluated by the fitness functions. The fitness measure does not constrain the frequency, phase and signal shape of the circuit's output to fixed values, since it is supposed that this would implicitly exclude useful pathways for evolution towards oscillating circuits. Consequently, a total of 6 more open and phenomenological fitness criteria are used for these experiments and are listed in table 7.2. The equations, that are used for the fitness calculation are listed below:

pathways for evolution through open fitness criteria

$$N = \text{no. of voltage samples} \tag{7.4}$$

$$V_{\text{mean}} = \frac{1}{N}\sum_{i=1}^{N} V_i \tag{7.5}$$

$$\text{DC offset} = \begin{cases} (2.5\,\text{V} - V_{\text{mean}})^2 & V_{mean} \leq 3\,\text{V} \vee V_{mean} \geq 3\,\text{V} \\ 0 & 2\,\text{V} < V_{mean} < 3\,\text{V} \end{cases} \tag{7.6}$$

$$\text{dev. from } V_{\text{mean}} = \sum_{out=1}^{N} (V_{\text{out}} - V_{\text{mean}})^2 \tag{7.7}$$

$$\text{amplitude span} = \max\{V_{out}\} - \min\{V_{out}\} \tag{7.8}$$

$$\text{zero cross.: } \mathscr{Z} = \{\forall\, \text{out} \in \mathbb{N} < N \,|\, V_{\text{out}} = V_{\text{mean}}\} \tag{7.9}$$

$$\text{no. of zero cross.} = \#\{\mathscr{Z}\} \tag{7.10}$$

$$\text{no. of periods} = \lfloor \#\{\mathscr{Z}\}/2 \rfloor \tag{7.11}$$

$$\text{period}_{\text{mean}} = \begin{cases} \frac{2}{\#\text{periods}} \sum_{i=2}^{\#\text{periods}} \mathscr{Z}_i - \mathscr{Z}_{i-1} & \#periods > 0 \\ N & \#periods = 0 \end{cases} \tag{7.12}$$

$$\text{period}_{\text{deviation}} = \begin{cases} \sum_{i=2}^{\#\text{periods}} ((\mathscr{Z}_i - \mathscr{Z}_{i-1}) - \text{period}_{\text{mean}})^2 & \#periods > 0 \\ N & \#periods = 0 \end{cases} \tag{7.13}$$

$$\tag{7.14}$$

7.3.3 Implications of Multi-Objective Optimization

The presented approach for multi-objective optimization inherently maintains great diversity of the individuals in all objectives. Due to this fact, it is expected that, in addition to finding over-all good solutions for oscillators, it will be possible to harvest multiple different results from one single evolution run: e.g. individuals with different *no. of zero crossings* represent oscillators with different frequencies. Thus, it will be a great benefit, if such solutions can be found, which feature an equally well performance in the other objectives.

expecting oscillators with different frequencies

Furthermore, it is neither expected to obtain solutions with a specific curve shape, nor to obtain solutions with specific frequencies, since both parameters are as yet not constraint. More precisely, there is a technical limit for the minimum and maximum

no constraint curve shape and phase

TM	objective	fitness	description
TM_1	DC offset	min.	dev. of V_{mean}, according to eq. 7.6
TM_1	dev. from V_{mean}	max.	penalize straight lines, eq. 7.7
TM_1	amplitude span	max.	the whole voltage range shall be used, eq. 7.8
TM_1	no. of zero crossings	max.	rewarding oscillation, eq. 7.10
TM_1	period deviation	min.	dev. of the detected period lengths, eq. 7.13
—	ressources used	min.	sum of transistors used

Table 7.2: *A list of all objectives, that are used for the multi-objective evolution of oscillators from scratch. Zero crossings and amplitudes are always measured relative to the mean output voltage (V_{mean}). Occurring periods are calculated from those zero crossings and shall feature the same period lengths.*

frequency, given by the number of samples and the sampling frequency, as described in section 7.3.2 (480 samples, $f_{sample} = 10\,\text{MHz}$, thus, freq. range $= 21\,\text{kHz}..2.5\,\text{MHz}$). Besides, it is observed in example experiments that a certain frequency and curve shape can be relatively easy achieved by postprocessing suitable found solutions by means of evolution, although this is not systematically done in this thesis. The test mode and fitness function, which are used to evaluate the phase shift of the sine wave in section 7.4.2, are a suitable setup for this task.

partly good solutions survive: crossover gains importance

Last but not least, an important consequence of the multi-objective approach is that—on purpose—also numerous only partly good, or even bad solutions can be found in the resulting populations. This is, on the one hand, again a consequence of the extensive diversity preserving behavior of the algorithm, which is, in the case of keeping bad solutions, not obviously useful. On the other hand, there are approaches in the field of EAs, where, inspired from natural genome structures, *'genetic garbage'* with no explicit meaning is added to the genomes on purpose, in order to find hidden pathways (tunnels) towards good solutions within the fitness landscape [89]. The inactive garbage genes can thereby be repeatedly mutated without affecting the fitness of an individual and, with a certain probability, will become meaningful, if the crossover copies them into an active region of the genome. Despite the fact that such *'genetic garbage'* is not voluntarily present in the genomes used in this thesis, maintaining whole *'garbage individuals'* probably has the same effects. As a consequence of this, the crossover operator gains importance, since it is able to create a greater variety of combinations.

7.3.4 Results and Conclusion

The distribution of the fitness values for all 6 objectives of three resulting populations are depicted in figure 7.8, in order to give examples of possible results. In principle, three different scenarios are observed: it can be seen from the graph on the top, which depicts the outcome of a successfull run, that the fitness values for all objectives are spread over wide ranges within the population. This indicates that the MO algorithm succeeded in converging towards good fitness and in maintaining diversity. Contrary to that, the graph in the middle and the graph at the bottom illustrate the results of failed runs, where the fitness for all objectives is stuck at bad values.

convergence in only some objectives

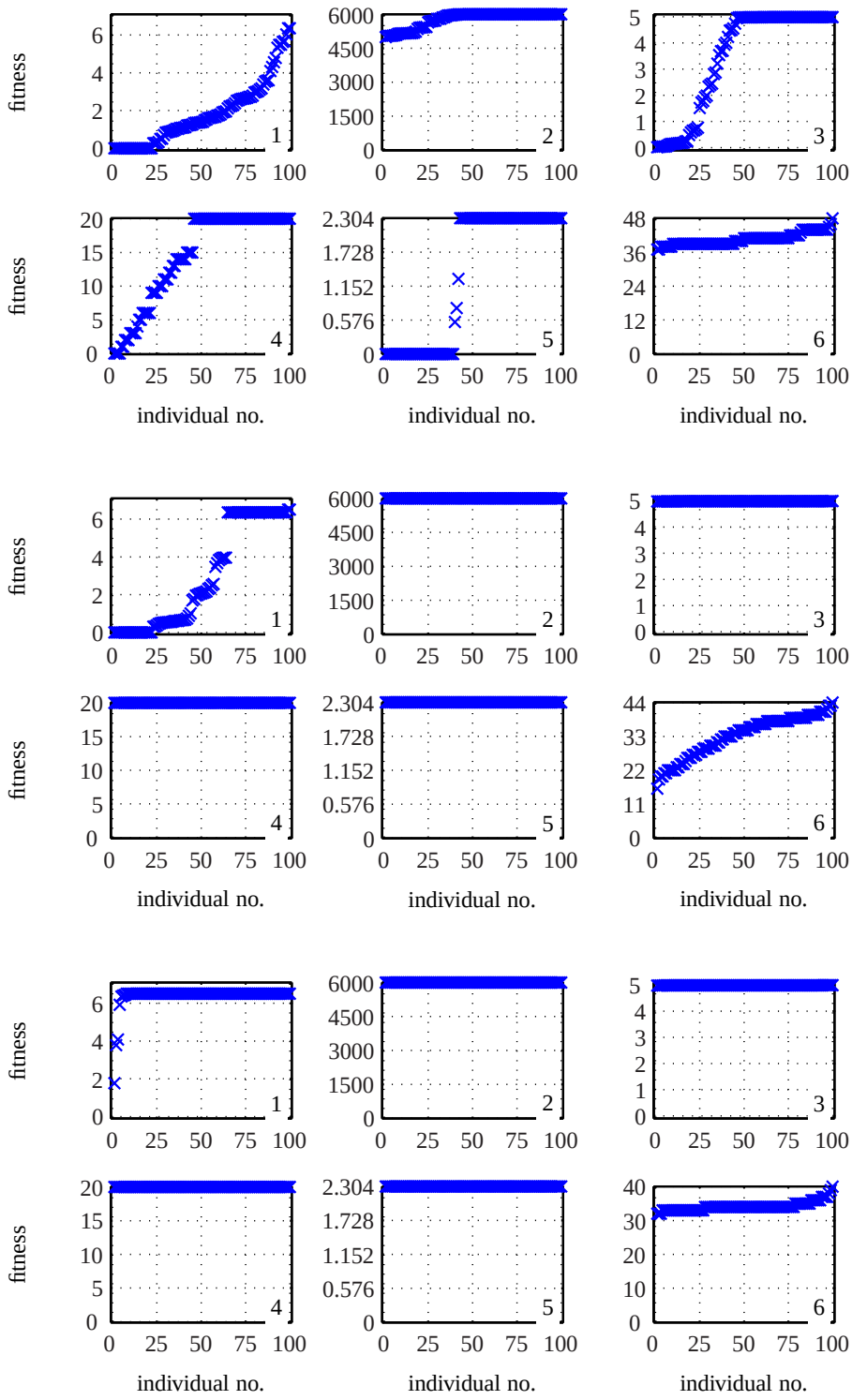


Figure 7.8: *Distribution of fitness of all individuals for runs 4,14,11. Fitness values are sorted for plotting.*

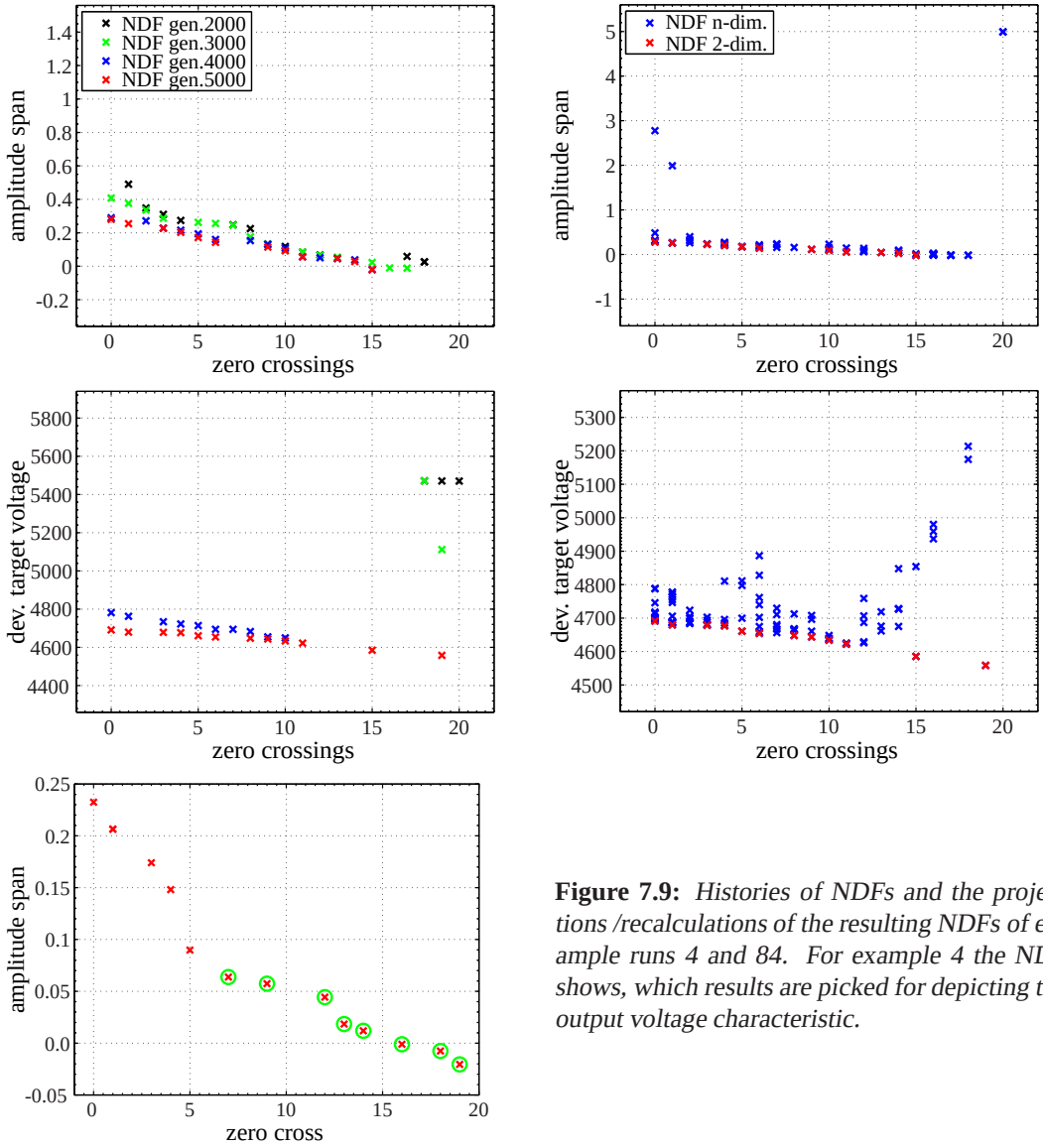


Figure 7.9: *Histories of NDFs and the projections /recalculations of the resulting NDFs of example runs 4 and 84. For example 4 the NDF shows, which results are picked for depicting the output voltage characteristic.*

Despite the fact that the fitness of two objectives—the *DC offset and the ressource consumption*—converges in the case of the run, which is depicted in the middle, this run is not to be considered partly successfull, since none of the resulting circuits actually oscillates. The reason for this is that achieving a sufficiently good fitness in the objectives *amplitude span* and *no. of zero crossings* is crucial for an oscillating circuit: a good fitness for *amplitude span* indicates that there are at least two extrema in the output voltage and a good fitness for the *no. of zero crossings* stands for the presence of a periodical zero crossing.

For these experiments, only 6 out of 100 resulting evolution runs feature oscillating circuits, while the other 94 runs did not converge. Nevertheless, those populations that did converge show a truly MO behavior, as can be seen from figure 7.9 and 7.10, in

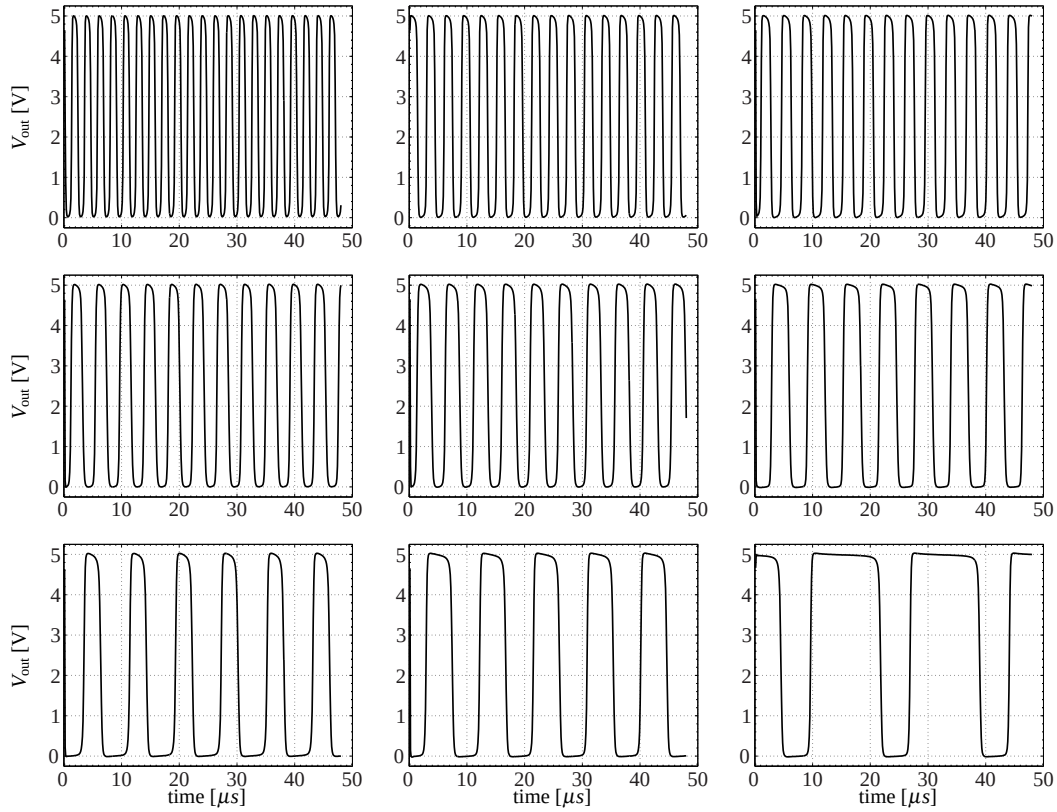


Figure 7.10: *Output voltage characteristics for the MO comparators: Basic GA, Turtle GA and DC simulation.*

which typical results of a successfull run are depicted. As an example, for the objectives *amplitude span* and *period deviation* over *no. of zero crossings*, the improvement of the respective non-dominated front over time (generation 2k, 3k, 4k and 5k) is depicted on the left hand side, while the resulting NDF (generation 5k) is shown on the right hand side. Note that, for illustration, the resulting NDFs are recalculated by considering only the two actually plotted objectives. Thus, on the right hand side, a projection of the respective multi-dimensional NDF, as seen by the algorithm, is additionally shown. Finally, one of the main achievements of the MO approach is that the successfull runs feature numerous solutions for an oscillator instead of only one. Thereby, resulting circuits are found that oscillate at various different frequencies 57 kHz, 107 kHz, 127 kHz, 161 kHz, 200 kHz, 233 kHz, 281 kHz, 328 kHz and 450 kHz, which can be nicely seen from figure 7.10.

harvesting different solutions from one successfull run

Concluding, oscillators with different frequencies are successfully evolved with the *MO-Turtle GA*, although it seems to be hard for evolution to get any initial oscillation started. Thus, the yield of good solutions is comparably low. Additionally, none of the resulting oscillators is working in simulation. It is observed that, as soon as any kind of oscillation is achieved, the populations converge further very quickly. Moreover, it is supposed that the non-restrictive formulation of the objectives is crucial for the evolution of certain properties of analog circuits, e.g. phase-shift, frequency or curve shape.

it is hard to get an oscillation started

7.4 A Circuit with Numerous Demands: an Operational Amplifier

what can be measured on-chip?

the challenge of opposing properties

In this chapter, the aim is to evolve an operational amplifier, since it is a challenging task in the regime of analog circuit design and it moreover combines various properties of the circuits from the previous experiments of this thesis. Further, some important characteristics of an amplifier (gain, common-mode rejection ratio (CMRR)) cannot be measured directly on the chip, due to the lack of configurable constant resistors, lossless feedback and the possibility of performing an AC analysis of the candidate circuits. Thus, it will be interesting to investigate, if those properties, described in the setup for the test modes (subsection 7.4.2) that actually can be measured on the FPTA, provide a sufficiently suitable test environment for the successfull evolution of amplifying circuits. Lastly, since an amplifier is a circuit with various, sometimes even opposing properties (e.g. gain and slew-rate), it represents once more a typically multi-objective problem for evolution.

7.4.1 Setup and On-chip Test Bench

experimental setup

predefined input configuration

The experiments are conducted with a population size of 200 for the intermediate population and a number of 10.000 generations per evolution run. Individuals are mutated according to the probabilities for the *MO-Turtle GA*, given in table 6.2 (mean mutation rate: 1 turtle/individual, erase/create: 40 %/60 %, reconnect 50 %) and crossover is carried out with a probability of 10 % and a maximum block-size of 4×4 transistor cells. An area of 9×9 transistor cells is provided to the evolving circuit. Due to the lessons learned from the previous experiments (section 6.4.4), both, the non-inverting (I_+) and the inverting (I_-) input of the circuit are statically connected to the gates of transistors of the same flavor, in order to avoid meaningless amplifiers. The W and L of those input transistors are always equal, although, during evolution, their size can be varied by the EA. Thereby, it is intended to provide some kind of differential pair as input to the evolving circuit and to avoid 'misuse' of the input voltages at the same time.

two testbenches: open loop and full feedback

Two series of experiments, each of 50 evolution runs, are carried out using PMOS input transistors in the first case and NMOS input transistors in the second case. Further, free resources of the transistor array are used to attach a randomly (by mutation) variable capacitive load to the circuits output and to implement two test benches for the circuit under test: one for open loop testing and another one with full feedback to the inverting input, for which a resulting gain of 1 is assumed. The on-chip test benches for the test modes of the measuring, described in the following subsection 7.4.2, are graphed in figure 7.11. Since the feedback is realized using only the configuration capabilities of the transistor array—where no constant resistors, capacities or current sources are available—it is not feasible to measure properties like gain or common-mode rejection ratio CMRR directly on the chip. Despite this, it is possible to measure and evaluate important properties of an amplifier, namely open-loop behavior, slew-rate, settling-time, DC offset, harmonic distortion and phase-shift, directly on the FPTA.

manually made reference designs

The FPTA is configured with manually created circuits, one with PMOS and one with NMOS input respectively, in order to be able to assess the quality of the synthesized circuits compared with human-made solutions. The reference designs are taken from [6, 90] and consist of a differential input stage and a simple inverter-output stage. The

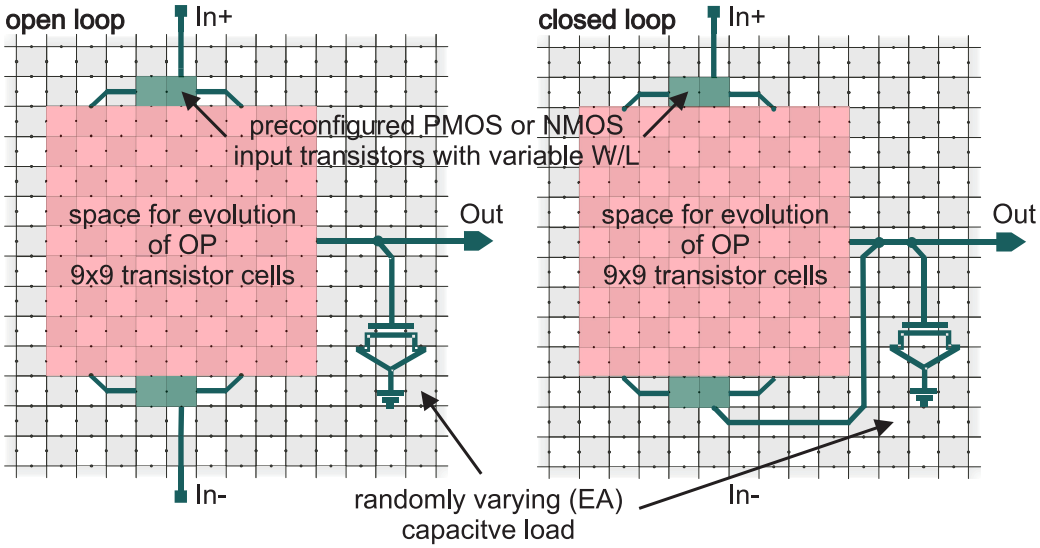


Figure 7.11: *The FPTA test benches for the evolution of operational amplifiers are graphed. Two series of experiments are carried out using preconfigured PMOS input transistors in the first case and NMOS input transistors in the second case. In both cases, an area of* 9×9 *transistor cells and the W/L of the fixed input transistors can be changed by the algorithm. Like in the previous experiments, a variable capacitive load is attached to the circuit's output. This capacitive load is randomly varied by the EA during the course of evolution.*

fitness values are measured for both reference designs, using exactly the same setup as throughout evolution, and are compared to those of the evolved circuits.

7.4.2 Test Modes for the Measurements on the FPTA

five testmodes deliver 12 fitness values

Five different test modes (TM_i) are used for the synthesis of operational amplifiers, delivering fitness values for a total of 12 objectives, including ressource consumption. Thereby, the measuring of TM_1 and TM_5 is performed with the open-loop test bench, while the full feedback test bench is used in the case of $TM_{2\text{-}4}$. An overview over the objectives is given in table 7.3 and the test modes are more closely described in the following:

open-loop behavior, offset

TM_1 and TM_5: open-loop behavior, offset. The task is to pull V_{out} to $V_{tar} = 5\,V$ if $V_{I+} > V_{I\text{-}}$ and to $V_{tar} = 0\,V$ if $V_{I+} < V_{I\text{-}}$ and to keep the offset voltage V_{os} low or at least constant. A set of nine curves at $V_{I+} = 1.5, 1.75, \ldots, 3.5\,V$, each consisting of 100 sample voltages for $V_{I\text{-}} = 0 \ldots 5\,V$, is used as test pattern for both test modes. Contrary to TM_5, the test pattern of TM_1 is randomized. Thus, TM_1 delivers one fitness value (*pull to rails (rand)*), which characterizes the quasi-DC behavior of the circuit, while TM_5 provides a corresponding fitness value (*pull to rails*) for the non-randomized case. Both values are calculated according to equation 7.17 and are actually representing the rms error of the measured output voltages. Additionally, the *DC offset* and *deviation of DC offset* are

calculated from the measuring results of TM_5 using equations 7.19 and 7.20.

$$N = \text{no. of voltage samples} \tag{7.15}$$
$$S = \text{no. of curves} \tag{7.16}$$
$$\text{pull to rails} = \sqrt{\frac{1}{N}\sum_{out=1}^{N}(V_{\text{out}} - V_{\text{target}})^2} \tag{7.17}$$
$$\text{offset pos.: } \mathscr{O} = \{V_{i_{\text{sweep i}}} \in \mathbb{N} < N \,|\, V_{i_{\text{out}}} \leq 2.5\,\text{V} \wedge V_{i+1_{\text{out}}} > 2.5\,\text{V}\} \tag{7.18}$$
$$\text{abs. offset} = \frac{1}{S}\sum_{set=1}^{S}|\mathscr{O}_i - V_{i_{\text{set}}}| \tag{7.19}$$
$$\Delta\text{offset} = \sqrt{\frac{1}{S}\sum_{set=1}^{S}(\mathscr{O}_i - V_{i_{\text{set}}})^2} \tag{7.20}$$
$$\tag{7.21}$$

slew-rate, settling-time

TM_2: Slew-Rate, Settling-Time. The challenge for the output is to follow four voltage-steps from $V_{\text{I+}} = 0\,\text{V}$ to $5\,\text{V}$, from $V_{\text{I+}} = 5\,\text{V}$ to $0\,\text{V}$, from $V_{\text{I+}} = 0\,\text{V}$ to $3\,\text{V}$ and from $V_{\text{I+}} = 3\,\text{V}$ to $2\,\text{V}$ in $t_{\text{step}} = 0.25\,\mu\text{s}$, respectively. Fitness values for the *slew-rate* and the *settling-time* are calculated from the period of time between the step and the point of time when V_{out} has settled at the new target voltage $V_{\text{tar}} \equiv V_{\text{I+}}$. An additional objective is given by the *deviation of V_{tar} from V_{out}*, which is once more calculated from the rms error equation (7.17). The equations that are used for the fitness calculation of *slew-rate* and *settling-time* are listed below:

$$V_{\text{low}} = V_{\text{min}} + 10\,\%\,\text{of}\,(V_{\text{max}} - V_{\text{min}}) \tag{7.22}$$
$$t_{\text{low}} = \{\text{t} \in \mathbb{R} < T \,|\, V_{t_{\text{out}}} \leq V_{\text{low}} \wedge V_{t+1_{\text{out}}} > V_{\text{low}}\} \tag{7.23}$$
$$V_{\text{high}} = V_{\text{min}} + 90\,\%\,\text{of}\,(V_{\text{max}} - V_{\text{min}}) \tag{7.24}$$
$$t_{\text{high}} = \{\text{t} \in \mathbb{R} < T \,|\, V_{t_{\text{out}}} < V_{t_{\text{high}}} \wedge V_{t+1_{\text{out}}} \geq V_{\text{high}}\} \tag{7.25}$$
$$\text{slew-rate} = \frac{V_{s_{\text{high}}} - V_{s_{\text{low}}}}{t_{s_{\text{high}}} - t_{s_{\text{low}}}} \tag{7.26}$$
$$V_{\text{tar}} = \begin{cases} V_{\text{min}} & \text{for step down} \\ V_{\text{max}} & \text{for step up} \end{cases} \tag{7.27}$$
$$\text{settling-time}_1 = \{t \in \mathbb{R} < T \,|\, V_{\text{tar}} - 0.05\,\text{V} < V_{t_{\text{out}}} < V_{\text{tar}} + 0.05\,\text{V}\} \tag{7.28}$$
$$\text{settling-time}_2 = \{t \in \mathbb{R} < T \,|\, V_{\text{tar}} - 0.25\,\text{V} < V_{t_{\text{out}}} < V_{\text{tar}} + 0.25\,\text{V}\} \tag{7.29}$$
$$\text{settling-time}_3 = \{t \in \mathbb{R} < T \,|\, V_{\text{tar}} - 0.5\,\text{V} < V_{t_{\text{out}}} < V_{\text{tar}} + 0.5\,\text{V}\} \tag{7.30}$$
$$\text{offset} = |\frac{t_{\text{low}} + t_{high}}{2} - t_{\text{step}}| \tag{7.31}$$
$$\tag{7.32}$$

magnitude, phase-shift and THD+N

TM_3 & TM_4: Magnitude, Phase-Shift, Harmonic Distortion. A further demand on an OP is to distort and damp the input signal as less as possible and to keep the phase-shift constant below at least 180°, in order to cause the amplifier to remain stable. These properties are measured in TM_4 by applying three different sinusoidal signals with $f = 5, 50\,\text{and}\,500\,\text{kHz}$ to the input and comparing them to the circuits output $V_{\text{tar}} \equiv V_{\text{I+}}$. A

TM	objective	fitness	description
TM_1	pull to rails (random)	min.	quasi-DC behavior, according to eq. 7.17
TM_2	slew-rate	max.	mean slew-rate of all steps (eq. 7.26)
TM_2	settling-time	min.	mean t_{settle} after trans. step (eq. 7.28-7.30)
TM_2	deviation from V_{tar}	min.	rms error, according to eq. 7.17
TM_3	phase-shift	min.	phase-shift of sin between V_{out} and $V_{\text{I+}}$
TM_3	sin-curve deviation	min.	rms error, according to eq. 7.17
TM_4	magnitude	max.	damping of the fund. freq. at unity gain
TM_4	harmonic distortion	min.	sum of ampl. of harmonics if above $-60dB$
TM_5	pull to rails	min.	rms error, according to eq. 7.17
TM_5	DC offset	min.	DC offset of the set of 9 curves (eq. 7.19)
TM_5	dev. of DC offset	min.	std. deviation of the DC offset (eq. 7.20)
—	resource consumption	min.	sum of transistors used

Table 7.3: *An overview of all test modes $TM_{1\text{-}5}$ and their corresponding objectives is given above. Generally, the aim is to minimize the fitness. Thus, in the cases where the objective value is to be maximized, the reciprocal (slew-rate) or absolute value (magnitude) is used as fitness.*

discrete fourier transform is used to calculate the power spectrum of the output signal for each frequency. Subsequently, fitness values for *magnitude* and *THD* are calculated from the resulting power spectrum. A description of THD+N is given in chapter3, section 3.3.2. Additionally, the output of a sinusoidal input signal of $f = 20\,\text{kHz}$ is used in TM_3 to obtain values for the *phase-shift* and the *deviation of V_{I+} from V_{out}*. Thereby, the *phase-shift* is calculated from the difference of the zero crossings of the target voltage pattern and the zero crossings of the measured voltage pattern, while the *deviation of V_{I+} from V_{out}* is, again, simply the rms error of the measured voltage pattern, with respect to the phase-shift.

7.4.3 Performance of the Multi-Objective Approach

convergence in all objectives

A first indicator, whether an evolution run is successfull or not, is the shape of the resulting fitness distribution for the different objectives. Thus, the fitness distributions of two typical evolution runs—one with NMOS and one with PMOS input—are depicted in figure 7.12. Contrary to the oscillators, the fitness values for all objectives of all 50 evolution runs are spread over the wide ranges. Thus, in the case of the OPs, the MO algorithm achieved that each resulting population contains individuals, which converged in at least one objective and to maintain great diversity of the individuals at the same time.

multi-dimensional NDFs

Due to the numerous objectives, the NDFs, which are calculated and optimized by the EA, are multi-dimensional. Thus, projections of example NDFs into the plane spanned by the respective objectives are plotted in figure 7.13. Thereby, all objectives are taken into account for computation. Additionally, the subsets, which are obtained by recalculating the NDF considering only the two graphed objectives, are shown, in order to illustrate, on the one hand, the relation to the example NDF in figure 7.1 and, on the other hand, to show the origin of the NDFs, depicted in figure 7.14. It is interesting to see that there are two different signatures of the distribution of the solutions within the search space: either

towards better fitness

towards greater diversity

the depicted objectives can be independently optimized as e.g. in the case of *magnitude*, *DC offset* and *settling-time*, where the individuals are spread over large areas of the fitness landscape, or the objectives are more closely correlated as e.g. *magnitude* and *distortion*, where only trade-off solutions are possible. According to the latter observation, as can be seen from figure 7.14, the respective NDF rather improves towards better fitness values over time in the first case, whereas in the second case, rather the diversity is improved by spreading the solutions over wide ranges of fitness.

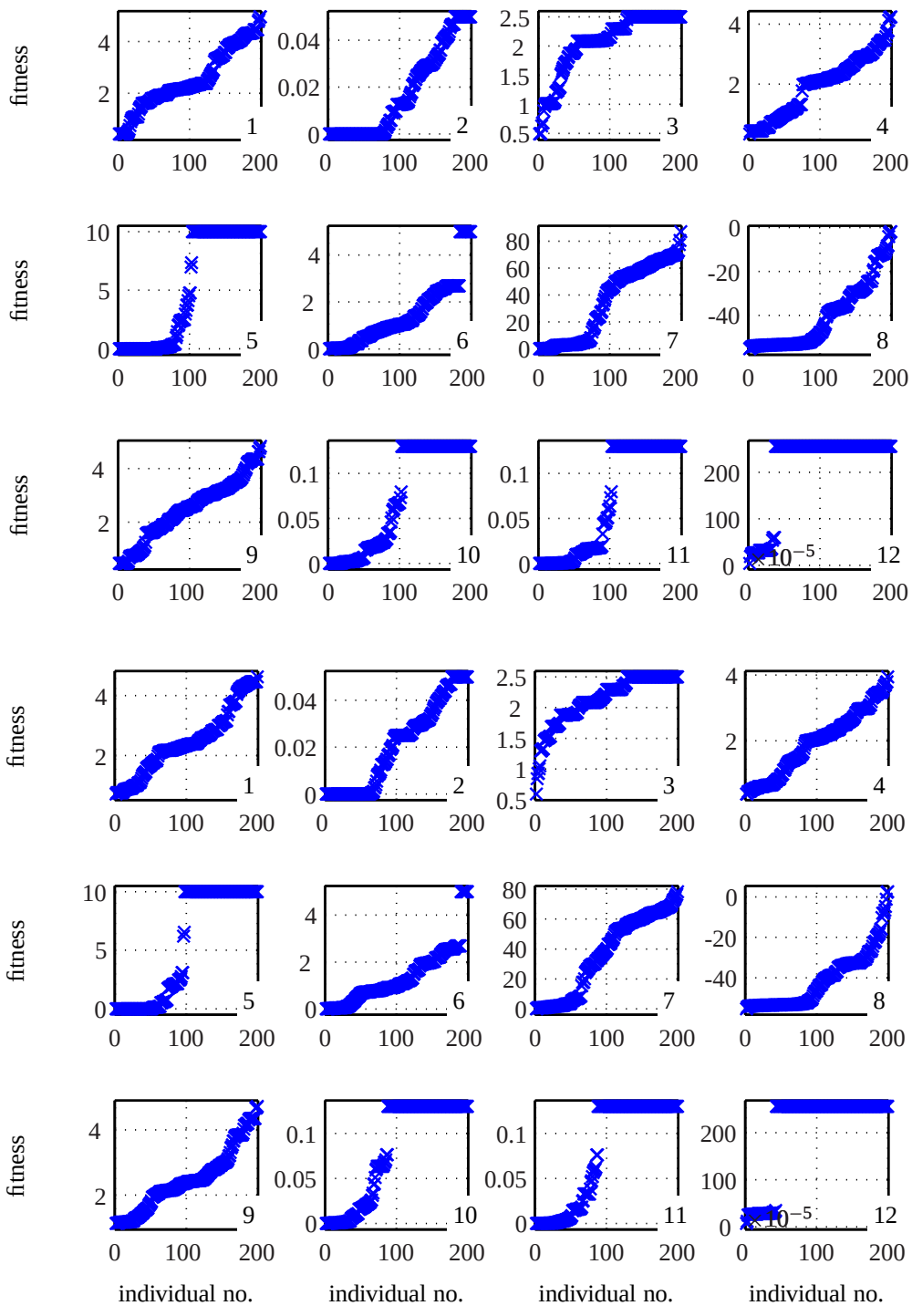


Figure 7.12: *The distribution of the fitness values for all objectives of two example evolution runs are depicted above. Thereby, a typical run with PMOS input (bottom) and a typical run with NMOS input (top) is shown. Furthermore, the fitness values are sorted in ascending order for illustration. For all 50 evolution runs, the fitness values cover wide ranges of the output range. The position of a manually made OP (reference) is marked by a triangle.*

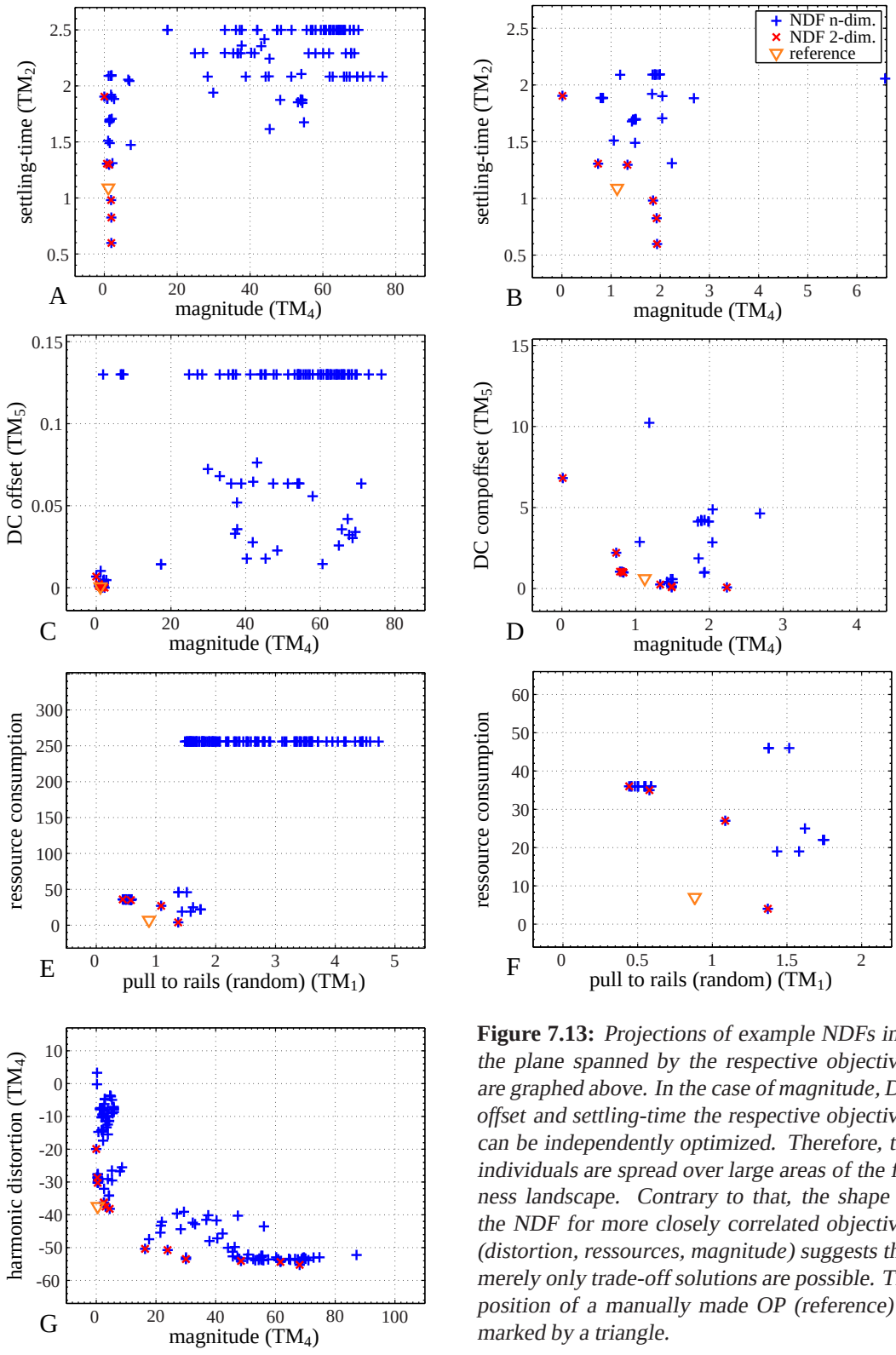


Figure 7.13: *Projections of example NDFs into the plane spanned by the respective objectives are graphed above. In the case of magnitude, DC offset and settling-time the respective objectives can be independently optimized. Therefore, the individuals are spread over large areas of the fitness landscape. Contrary to that, the shape of the NDF for more closely correlated objectives (distortion, ressources, magnitude) suggests that merely only trade-off solutions are possible. The position of a manually made OP (reference) is marked by a triangle.*

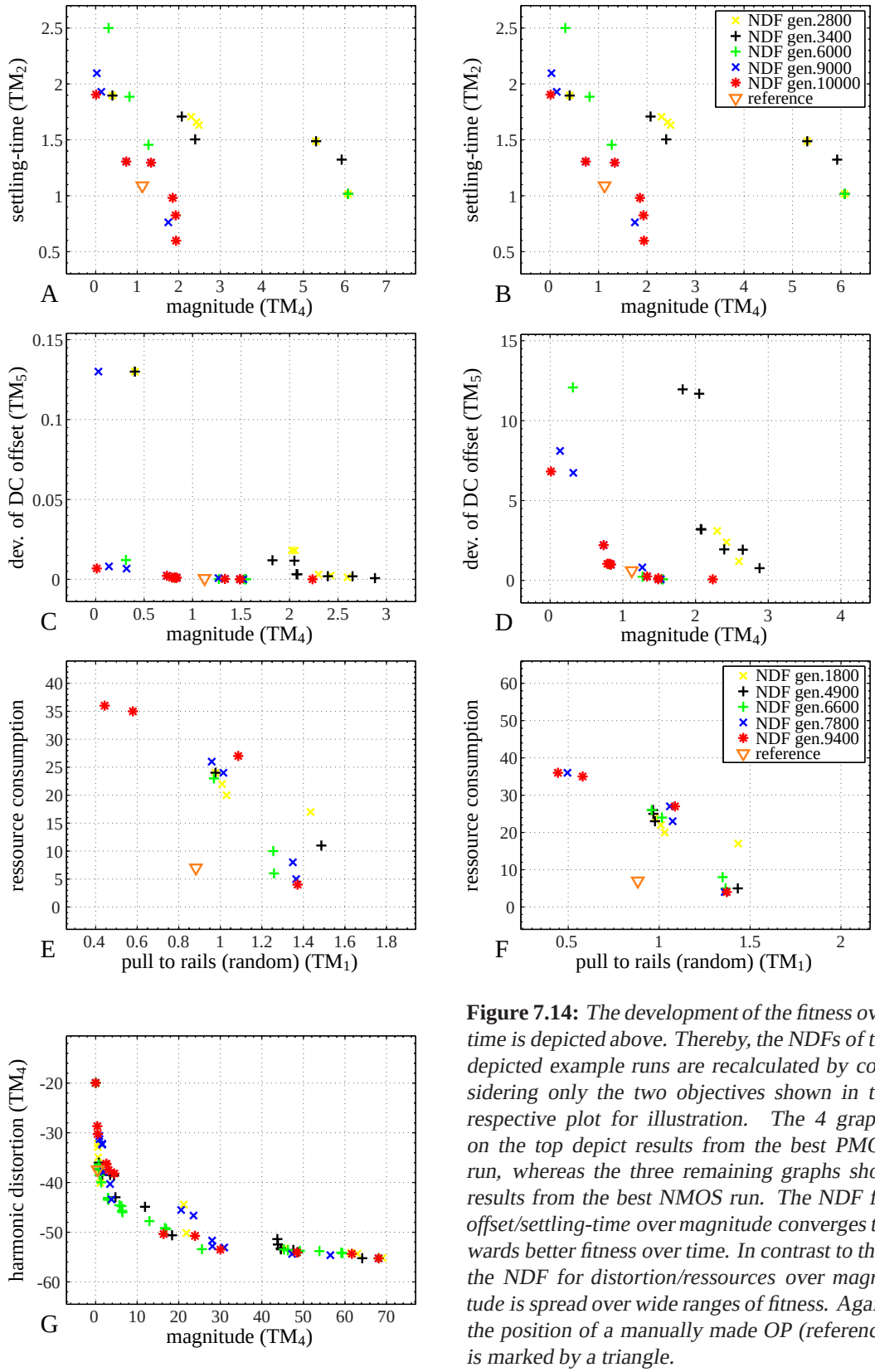


Figure 7.14: *The development of the fitness over time is depicted above. Thereby, the NDFs of the depicted example runs are recalculated by considering only the two objectives shown in the respective plot for illustration. The 4 graphs on the top depict results from the best PMOS run, whereas the three remaining graphs show results from the best NMOS run. The NDF for offset/settling-time over magnitude converges towards better fitness over time. In contrast to this, the NDF for distortion/ressources over magnitude is spread over wide ranges of fitness. Again, the position of a manually made OP (reference) is marked by a triangle.*

7.4.4 Solutions for the Operational Amplifier

Measurement on the FPTA.

comparison with reference designs

In order to assess the performance of the resulting OPs, human-made reference designs are manually realized on the configurable transistor array and are measured with the same test modes as are used for the evolution experiments. Subsequently, the obtained fitness values for all objectives can be compared to those from the solutions, found by the EA. It is tested for how many objectives the performance of the evolved circuits is better, resp. less than 10 % worse, than the performance of the according reference designs (PMOS or NMOS input). Thereby, it is recorded how many resulting populations feature at least one individual, which beats the manual design in N objectives. As can be seen from Tab. 7.4, in which the results of the latter measuring are listed, almost each run contains at least one individual that outperforms the respective reference OP in up to 3 objectives and about half of the runs feature similar performance in up to 6 objectives. Finally, for both setups, with NMOS and PMOS input, at least one evolution run features an individual, which performs equally well in 9 objectives.

at least one solution beats 6 objectives of the reference design

which objectives are more likely beaten?

In addition to this, the likelihood with which a certain objective of the reference designs is outperformed by the individuals of all resulting populations is listed in table 7.5. It can be seen that *ressource consumption*, *magnitude*, *pull to rails (random)*, *DC offset* and *deviation from* V_{tar} are barely beaten by evolved solutions. This is not surprising, since the human-made design contains only relevant components and it's differential input stage is designed to minimize the offset, while an additional gain stage—despite it is represented by a simple inverter in this case—provides sufficient magnitude. Contrary to that, it has not been expected that achieving a good fitness for *settling-time*, *phase-shift* and *harmonic distortion* seems to be relatively easy for evolution. The reason for this is probably that there is a large amount of individuals, which manage to propagate the input voltage pattern directly to the output, resulting in a low *phase-shift*, *settling-time* and *harmonic distortion*. An additional hint for this is the fact that all 'easy' objectives are covered by test modes, where the input voltage pattern corresponds to the (inverse) output voltage pattern. Thus, the tasks could be fulfilled best by a simple inverter, if there were no other objectives.

Simulating the test modes used for evolution.

simulating on level 1 and 2

at least three solutions beat 6 objectives of the reference design

The next task is to test the resulting circuits in simulation. Therefore, one individual of each population, which performs equally well as the reference design in most objectives on the chip is evaluated in simulation. For comparison, the same test mode setup, which is used for the evolution experiments on the FPTA, is used for simulation. TM_1 and TM_5 are represented by DC simulations, while TM_2, TM_3 and TM_4 are set-up as transient simulations. Thereby, in the case of TM_4, magnitude and distortion are obtained by performing a fast fourier transform on the output values. Further, the simulations are performed with extracted netlists of level 1 and level 2 (chapter 4, section 4.4). It can be seen from table 7.6 that the EA was able to find at least about 3 circuits, which perform equally well as the manually made OP in up to 6 objectives in the case of a simulation with plain transistors. This is a promising result insofar that, from a netlist with plain transistors it should be relatively easy to obtain a clear schematic and possibly understand the operation principle of the evolved solution. The output voltage characteristics of the

	no. of objectives											
	1	2	3	4	5	6	7	8	9	10	11	12
NMOS, on-chip												
better than ref.	50	50	46	39	32	18	5	2	1	0	0	0
max. 10 % worse than ref.	50	50	48	42	35	32	17	6	1	0	0	0
PMOS, on-chip												
better than ref.	50	49	42	26	15	3	0	0	0	0	0	0
max. 10 % worse than ref.	50	50	46	34	27	14	5	2	1	0	0	0

Table 7.4: *The no. of runs that contain at least one individual that achieved a better (or not more then 10% worse) fitness value than the manually made circuits for a given no. of objectives. About half of the runs feature a similar performance as the manually made design in up to 6 objectives and at least one evolution run features an individual, which performs equally well as the reference design in 9 objectives. The likelihood with which a certain objective of the reference designs is outperformed by the individuals of the resulting populations is listed in table 7.5.*

TM	objective	ref. beaten by % of evo. individuals	
		NMOS, on-chip	PMOS, on-chip
TM_1	pull to rails (random)	2.1	3.3
TM_2	slew-rate	12.9	7.5
TM_2	settling-time	25.7	1.4
TM_2	deviation from V_{tar}	3.5	2.5
TM_3	phase-shift	23.6	21.1
TM_3	sin-curve deviation	6.7	1.4
TM_4	magnitude	2.1	4.2
TM_4	harmonic distortion	58.9	68.9
TM_5	pull to rails	4.8	3.5
TM_5	DC offset	3.6	2.8
TM_5	dev. of DC offset	9.7	8.2
—	resource consumption	0.7	0.9

Table 7.5: *The likelihood with which a certain objective of the reference designs is outperformed by the individuals of all resulting populations is listed above. It can be seen that resource consumption, magnitude, pull to rails (random), DC offset and deviation from V_{tar} are barely beaten by the evolved solutions, while achieving a good settling-time, phase-shift and harmonic distortion seems to be relatively easy for evolution.*

NMOS and the PMOS OP with the best performance are graphed in figures 7.16, 7.17 and 7.15. The results from the FPTA are thereby compared to those from simulation and the evolved circuits are compared with the manually made designs.

	no. of objectives											
	1	2	3	4	5	6	7	8	9	10	11	12
NMOS, simulation												
level 1												
±10 % of ref. fitness	7	7	7	6	5	5	3	1	0	0	0	0
level 2												
±10 % of ref. fitness	23	21	20	20	15	10	4	4	0	0	0	0
PMOS, simulation												
level 1												
±10 % of ref. fitness	7	7	6	5	5	4	1	0	0	0	0	0
level 2												
±10 % of ref. fitness	23	19	19	17	12	11	4	1	0	0	0	0

Table 7.6: *The individual of each run, which performs equally well as the reference design in most objectives, is evaluated in simulation and therefore considered in this table. The no. of individuals, which achieved a comparable fitness value as the corresponding reference OP in simulation for a certain no. of objectives, is listed above. Simulations are performed with plain transistors (level 1) and with mean switch resistance included (level 2), respectively. It is a promising result that the EA was able to find at least about 3 circuits, which perform equally well as the manually made OP in up to 6 objectives in the case of a simulation with plain transistors.*

assessing evolved circuits in real-world application

Testing typical OP characteristics.
The circuits with the best performance in the simulation with plain transistors—one with NMOS and one with PMOS input—is further examined in the following. Opposite to the competition with the reference circuits on the FPTA, the evolved circuits will come off worse, if typical characteristics of OPs are compared in simulation. As can be seen from table 7.7 especially those properties that cannot be measured directly on the transistor array during evolution—thus, cannot be evaluated by a fitness function (e.g. open-loop gain)—return rather poor results. Contrary to that, the characteristics that are represented by an objective, perform similar, e.g. *offset*, *slew-rate* and *settling-time*, despite they turned out to be hard to evolve, as observed in table 7.5. Since the output voltage swing and the 0dB bandwidth are correlated to a good open-loop gain, those values are also not as good as those of the manually made OPs.

In both cases, the phase-margin of the evolved solution is higher than those of the reference OPs. This is interesting insofar, that it is on the one hand a good result, since the aim of the corresponding objective is to minimize the phase-shift. On the other hand, forcing the phase-shift towards too small values could possibly thwart the evolution of output gain stages. If this was the case, it would be better to allow for a greater phase-margin in the objective function. Output voltage characteristics for open-loop gain, CMRR, phase-margin and output voltage swing/range are graphed in figure 7.18.

parameter	NMOS (evo)	NMOS (ref)	PMOS (evo)	PMOS (ref)
open-loop gain	37 dB	57 dB	29 dB	65 dB
0dB bandwidth	8 MHz	77 MHz	6 MHz	33 MHz
offset	40 mV	28 mV	80 mV	20 mV
slew-rate (+)	40 $\frac{V}{\mu s}$	100 $\frac{V}{\mu s}$	15 $\frac{V}{\mu s}$	25 $\frac{V}{\mu s}$
slew-rate (-)	15 $\frac{V}{\mu s}$	30 $\frac{V}{\mu s}$	35 $\frac{V}{\mu s}$	45 $\frac{V}{\mu s}$
settling-time	0.4 μs	0.2 μs	0.3 μs	0.2 μs
phase-margin	91°	50°	92°	50°
common mode rejection	56 dB	$>$ 50 dB	45 dB	$>$ 40 dB
out voltage swing	2.2 V	4.8 V	2.8 V	4.8 V
input common mode range	4.8 V	4.2 V	4.0 V	4.3 V

Table 7.7: *Comparison between characteristics of evolved circuits with a good performance and the reference circuits (NMOS and PMOS input). The values are obtained from SPICE simulations.*

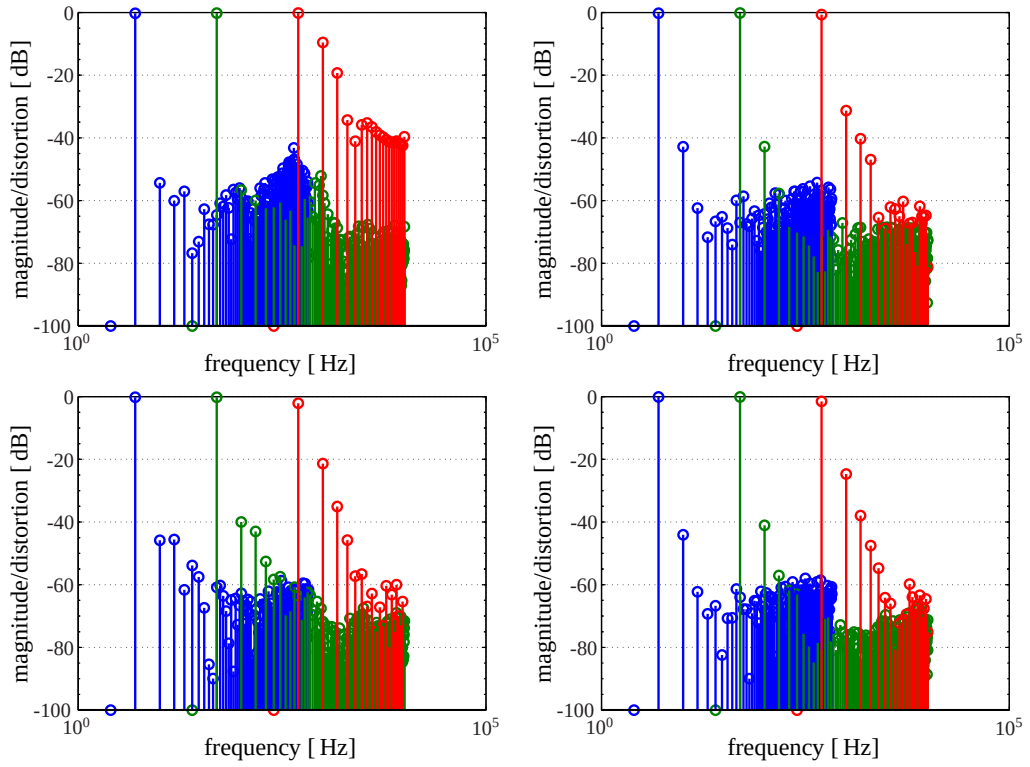


Figure 7.15: *Upper left: magnitude and distortion of a good evolved solution with NMOS input are depicted. Upper right: magnitude and distortion of the reference design with NMOS input are depicted. Lower left, lower right: the according graphs for circuits with PMOS input are shown. The measuring on the FPTA (straight line) is compared with the simulation result (dashed line) and the target voltage pattern (gray dashed line).*

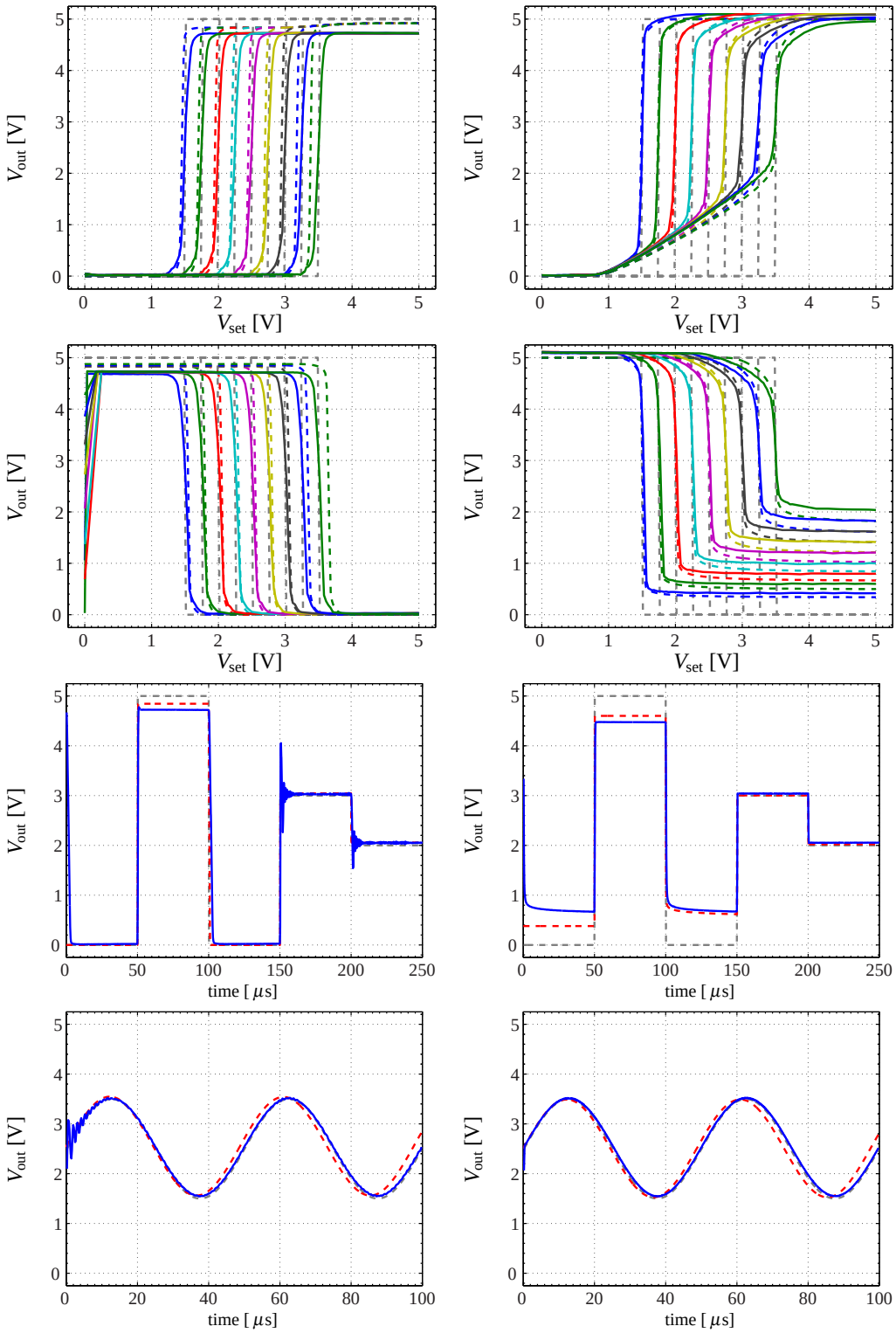


Figure 7.16: *Left: Output voltage characteristics of a good evolved solution with NMOS input are depicted. Right: Output voltage characteristics of the reference design withNMOS input are depicted. The measuring on the FPTA (straight line) is compared with the simulation result (dashed line) and the target voltage pattern (gray dashed lines).*

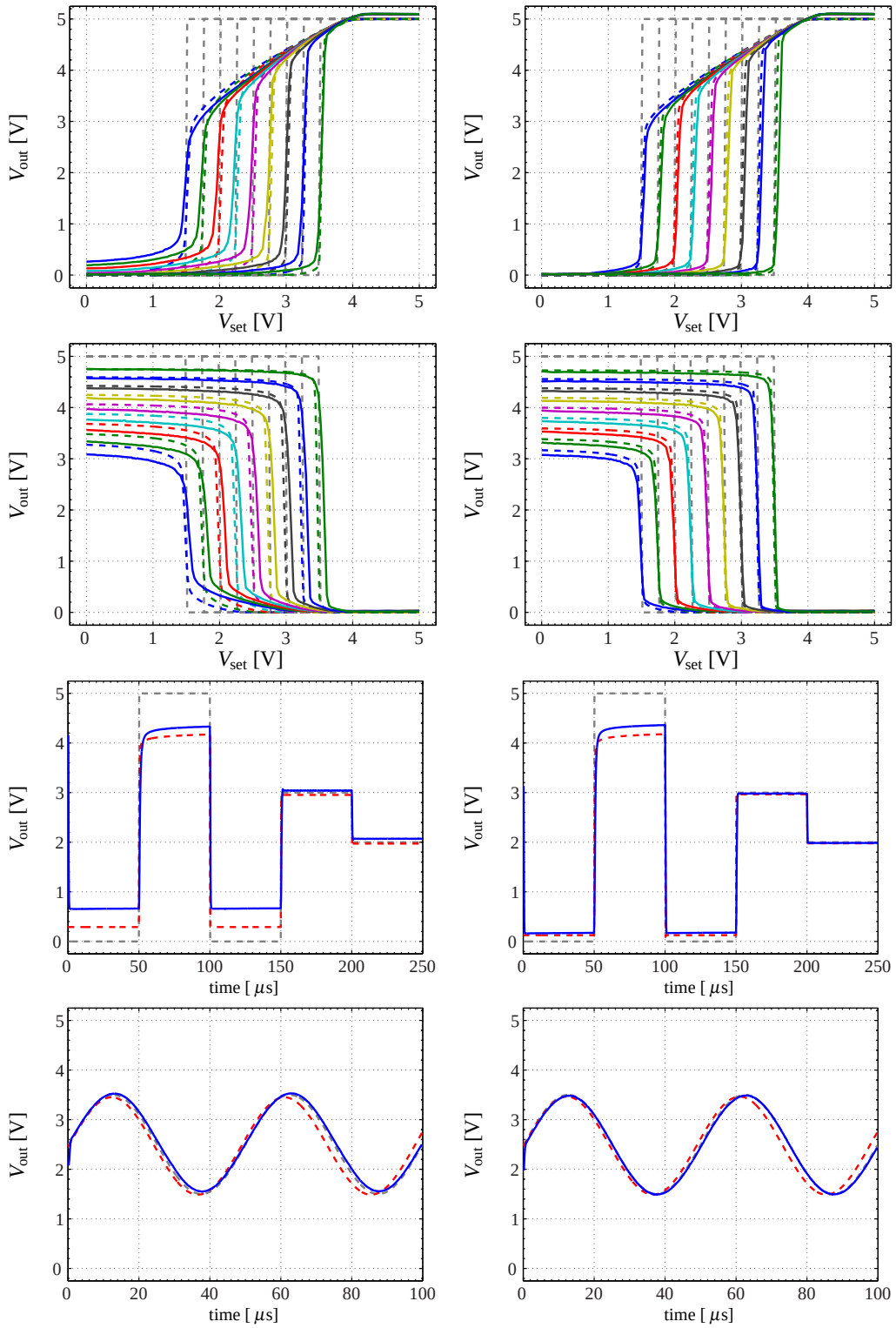


Figure 7.17: *Left: Output voltage characteristics of a good evolved solution with PMOS input are depicted. Right: Output voltage characteristics of the reference design withPMOS input are depicted. The measuring on the FPTA (straight line) is compared with the simulation result (dashed line) and the target voltage pattern (gray dashed lines).*

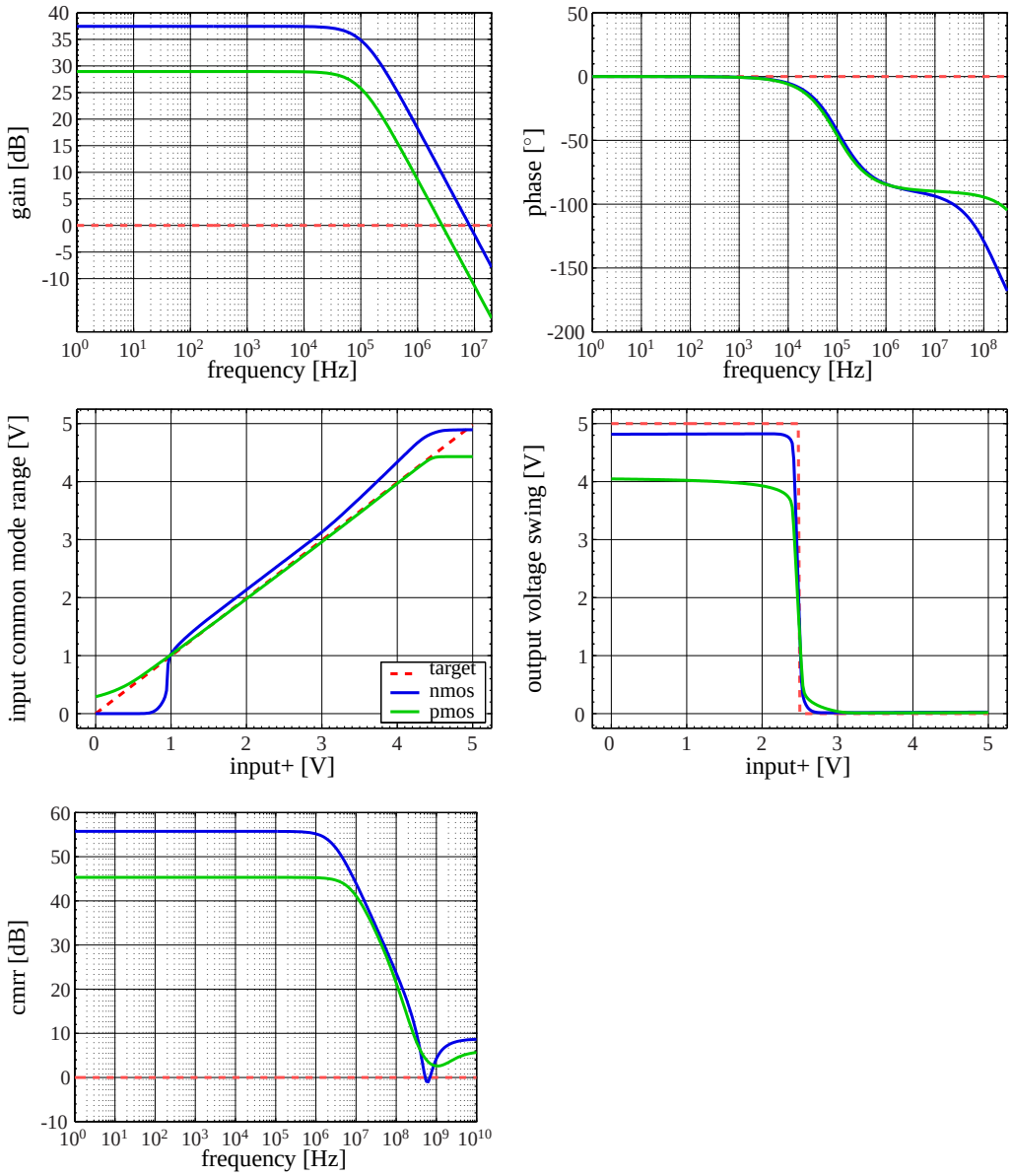


Figure 7.18: *Real world characteristics—namely open-loop gain, phase-margin, CMRR and output voltage swing/range—of good evolved OPs with NMOS and PMOS, respectively, are depicted above. The results for the NMOS solution is graphed as blue straight line, while the result for the PMOS case is graphed as green straight line. Again, the target voltages are pictured as gray dashed line.*

7.4.5 Schematic Extraction of Good Solutions: Deriving New Design Principles?

Due to the lessons learned by looking at the schematics of logic gates and comparators in chapter 6, section 6.4.5, the evolution setup for the OPs is designed in a way that the input voltages are always connected to the gate of either a NMOS or a PMOS transistor pair. As a consequence of this, the algorithm cannot simply mix the input voltages in an unwanted way, as it occurred in the case of the logic gates. Again, the intention is to transfer good solutions into a human readable format and possibly understand the operation principle of the evolved OPs. The procedure, which automatically generates schematics from the evolved circuits, is described in chapter 4, section 4.5.2.

understanding schematics of good solutions

For both types of input, generated schematics for example solutions with both, a good performance on the chip and in a level 2 simulation are shown in figure 7.19. In order to improve understandability, circuits with minimized ressource consumption are chosen. Besides, the depicted circuits are those from the previous subsection 7.4.4. Additionally, the obtained schematics are manually redrawn in figure 7.20 by extracting functional groups. Unfortunately, both depicted circuits will only be working correctly, if the mean parasitic on-resistance of the switches is considered in simulation (level 2). As can be seen from the schematics, in both cases, with NMOS and PMOS input, the algorithm was able to create a differential input stage, which is a promising result, since it is indeed a reinvention of a widely used human design. Nevertheless, two important components are as yet missing: first, contrary to the experiments with the logic gates and the comparators, the algorithm failed to create an inverter at the circuit's output, which would have resulted in significantly higher gain. A possible reason for this is the missing objective for open-loop gain, which cannot be directly measured on the FPTA. Second, the EA comes up with unusual biasing circuitry, which is probably the reason why the circuits are not properly working, if only plain transistors are simulated. In the case of the PMOS inputs, numerous transistors, which are actually closed, are contributing to the bias, while, in the case of the NMOS inputs, the bias voltage is even depending on the output voltage.

rediscovery a differential input stage

failure in creating a gain stage

unconventional biasing circuitry

In addition to the two latter examples, resulting OP circuits can be found, which also feature a good performance in a simulation with only plain transistors, which is a good result. Despite this, it was not yet achieved to further understand the operation principle of those circuits. Therefore, the according schematics are shown in appendix B.

7.4.6 Concluding Remarks

Concluding, it can be stated that, despite the EA did not discover any new groundbreaking design, it is still an impressive result that the algorithm achieved to synthesize a differential input stage in both cases without prior analog design knowledge. Furthermore, a great benefit of using an MO approach for the evolution of operational amplifiers on the Heidelberg FPTA is the possibility to efficiently explore the search space taking care of both, the diversity of the population and the various demands on the target circuit. Thereby, the obtained results suggest that a MO approach is mandatory for the successfull evolution of complex analog circuits.

rediscovery of differential input stage

successfull evolution of complex analog circuits

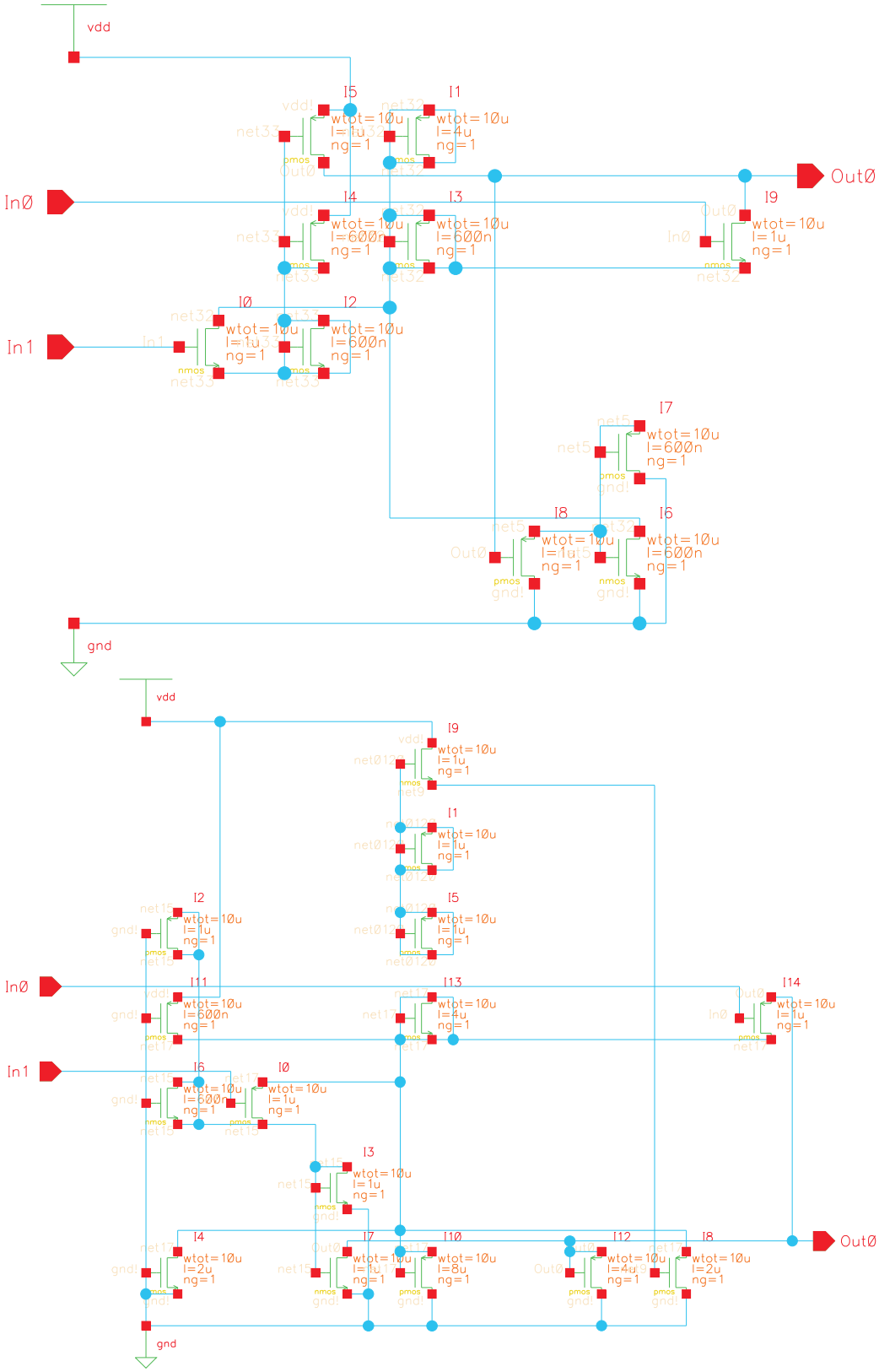


Figure 7.19: *Automatically generated schematics of the evolved OPs are shown above for both types of input: NMOS (top) and PMOS (bottom).*

W/L [µm]	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
left: NMOS	12/1	1/4	7/0.6	6/0.6	3/0.6	7/1	3/0.6	2/0.6	2/1	12/1					
right: PMOS	12/1	5/1	2/1	4/1	3/2	5/1	4/0.6	6/1	4/2	1/1	14/8	5/0.6	15/4	4/4	12/1

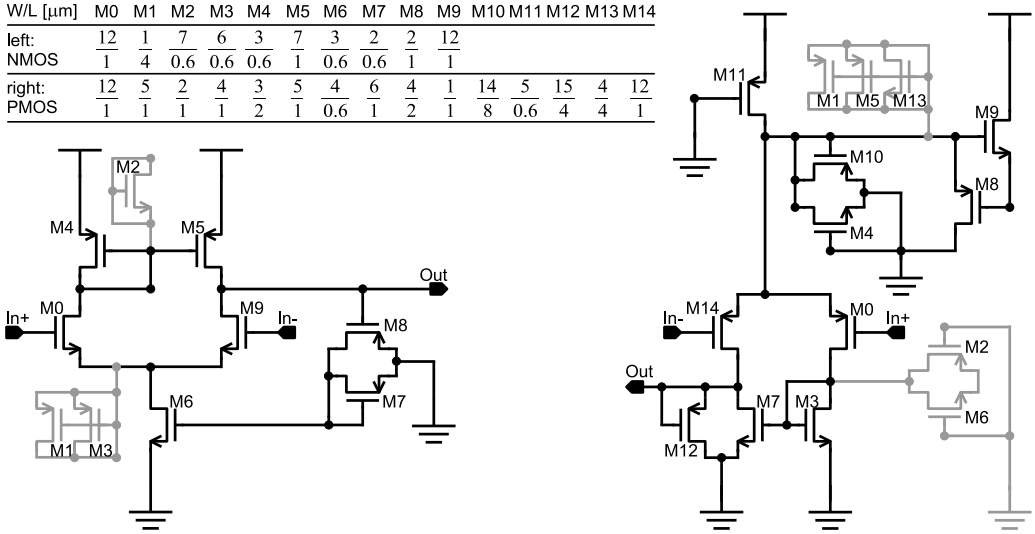


Figure 7.20: *Redrawn schematics of the evolved OPs from figure 7.19 are shown above for both types of input: NMOS and PMOS. Shorted transistors are grayed, in order to improve readability. In both cases the MO-Turtle GA achieved to synthesize differential input stages and quite unusual biasing circuitry. Unfortunately, the evolved solutions thus far lack of an output gain stage.*

Chapter 8

Modeling FPTA Architectures

Initial experiments with a simulation model of the current FPTA, referred to as the SimFPTA throughout the remainder of this thesis, are presented in this chapter. Once more, in order to be able to compare the results with those from the previous chapter (7), the task is to evolve a comparator with the MO-Turtle GA. Due to the significantly higher time consumption of the simulation, a total of 30 evolution runs is carried out with the SimFPTA and compared to 30 randomly picked FPTA runs from chapter 7, section 7.2. It is shown that both substrates feature similar behavior and results in the case of the comparators. Furthermore, the opportunity is seized to comment the current chip, to propose some improvements for possible FPTA architectures in the future and an alternative genome-chip mapping for the candidate circuits. To conclude, this last chapter is, on the one hand, intended as a demonstration of the performace of the SimFPTA and shall, on the other hand, provide a starting-point for future work. Thereby, the developed software framework, introduced in chapter 5, can be used to create and investigate new configurable FPTA topologies and further improved evolutionary algorithms.

After performing a great number of evolution experiments on a particular FPTA chip, the inevitable question arises, to what extent the constraints of the present topology limit the quality of the resulting circuits or exclude possible pathways of evolution towards good solutions. The latter question can only be answered by, on the one hand, investigating the evolution results for different design problems and, on the other hand, compare the evolvability of the current FPTA with the evolvability of other topologies. Due to the fact that the architecture of the available hardware is fixed, hence, can only be changed by designing and fabricating a new chip, it is desireable to be equipped with a customizable simulation model of FPTA architectures, as proposed in chapter 5 and in [93]. Such an FPTA model makes it possible to evaluate and improve the performance of an architecture before actually manufacturing a chip. However, the prize that has to be paid for a cus-

tomizable simulation model is a high evaluation time for the analog circuits in simulation. Since numerous candidate solutions have to be evaluated in the case of EAs, the time consumption of an evolution run drastically increases. A comparison of the evolution time of the experiments in this thesis is given in section 8.1.2. First steps towards modeling and testing FPTA architectures are presented in this chapter, including experiments with a simulation model of the current chip, a description of its main shortcomings as well as proposals for possible future architectures.

8.1 A Simulation Model of the Current FPTA

a simplified model: the SimFPTA

The first step is to create a simplified simulation model of the current FPTA architecture, referred to as the SimFPTA throughout the remainder of this thesis, and to compare the performance of this model with the real hardware by tackling the synthesis of comparators. Thereby, the *Turtle GA*, which is introduced in chapter 6, section 6.1.2, is used for the evolution experiments with both substrates, namely the FPTA and the SimFPTA. Further, the same genetic representation is used for both experiments, although, in the case of the SimFPTA, the candidate circuits are converted to a netlist and evaluated with a SPICE simulator, as described in chapter 5, section 5.3.2 and 5.4.3. In order to provide a sufficiently accurate model of the real hardware and to keep the simulation time feasibly low at the same time, the simulation model has to be simplified. The results from chapter 6 suggest that carrying out the simulations with level 2 netlists (chapter 4, section 4.4), where the transmission gates used are replaced with their mean on-resistance should be a suitable model of the real FPTA.

trade-off between model complexity and simulation time

8.1.1 Performing Experiments with the Simulation Model

Experimental Setup:

The same test modes, fitness measures and EA parameters as for the evolution of comparators in chapter 7, section 7.2.1, are used for the experimental setup, in order to be able to compare the results, obtained with the SimFPTA with those from the FPTA. Due to the significantly increased time consumption of the evolution experiments with the SimFPTA, a total of 30 evolution runs is carried out instead of 50, and compared to 30 randomly selected evolution runs of the comparators from chapter 6. The population size is 50 individuals, and the number of generations is 20.000.

Results

both substrates produce equal results

As can be nicely seen from Fig. 8.1, the evolutionary runs end up in equal ranges of fitness and ressource consumption for both substrates. Consequently, the SimFPTA is a suitable model for evaluating the evolvability of the real hardware, although the behavior is different in the beginning of evolution, due to the fact that individuals which do not work at all in simulation, nevertheless produce an output on the chip. The latter effect will become less important, if, as presented here, an EA like the *Turtle GA* is used for the evolution experiments, since valid circuits are crucial for carrying out a successful simulation. Output voltage characteristics for both test-modes of an individual with a good performance is graphed in Fig. 8.2.

valid circuits are crucial for successfull simulation

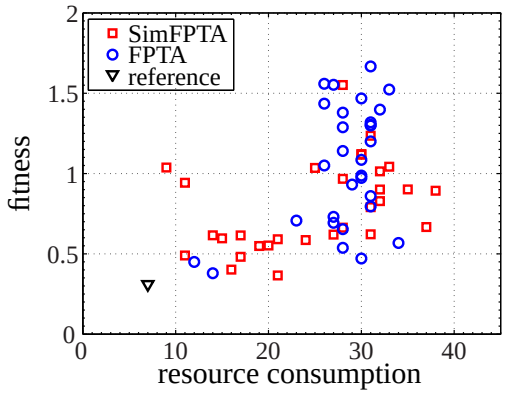


Figure 8.1: *The fitness for the output voltage characteristic is plotted over the ressource consumption. As can be seen from the graph, the best resulting comparators, evolved with the SimFPTA and the FPTA respectively, are distributed over equal ranges of fitness. Thus, the SimFPTA is suitable for assessing the evolvability of the FPTA.*

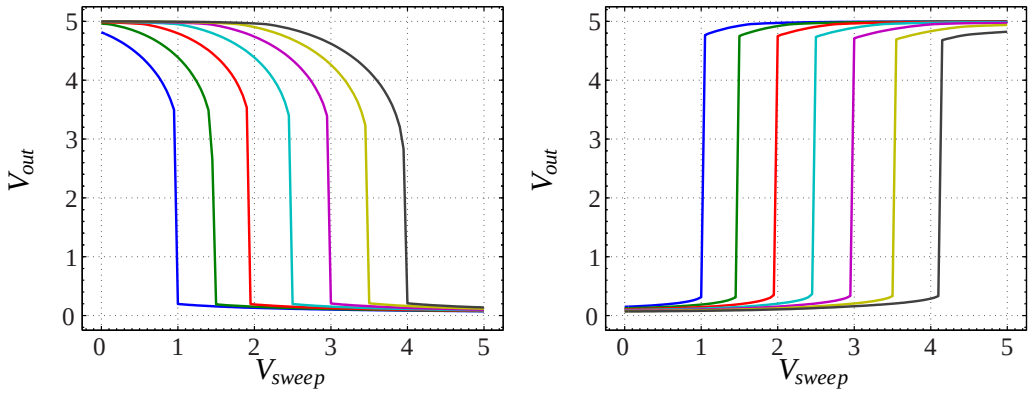


Figure 8.2: *DC output voltage characteristics of the best comparator, evolved with the MO-Turtle GA and the SimFPTA respectively, are depicted.*

8.1.2 Time Consumption for Different Evolution Experiments

factors, that influence simulation time

As already mentioned, the time consumption for one evolutionary run will significantly increase, if the evaluation of the circuits is performed with an analog circuit simulator instead of with the real hardware. In both cases, the simulation time is additionally depending on the number of inputs and outputs used, the number of voltage samples and the number of test modes. Furthermore, if a simulator is used, a higher amount of components will exponentially increase the simulation time and, additionally, the duration of the simulation depends on the type of analysis, e.g. a transient analysis takes considerably long. Contrary to simulation, only transient measuring is possible with the chip, although, in the case of real hardware, the output can be measured (almost) immediately, which is a great advantage. A further great advantage of the hardware is the fact that the measuring time is independent from the number of components used.

	substrate	no. TMs	no. of fit. vals	no. ind./gen.	time [min.]
logic gates	FPTA	2	1+1	50 / 10.000	65 ± 2
comparators	FPTA	2	1+1	50 / 20.000	200 ± 8
comparators	FPTA	2	2+1 (MO)	50 / 20.000	210 ± 8
comparators	SimFPTA	2	1+1	50 / 20.000	1380 ± 95
oscillators	FPTA	1	5+1 (MO)	100 / 5.000	25 ± 2
op. amplifiers	FPTA	5	11+1 (MO)	100 / 10.000	720 ± 8

Table 8.1: *An overview of the mean duration of one evolution run of all experiments in this thesis. Note, that in the case of complex fitness evaluation, e.g. in those cases where a fourier transform is done or simply one test mode delivers numerous fitness values, the time consumption further increases. The ressource consumption is considered as '+1' in this table, due to the fact that it is calculated outside the test mode setup.*

time consumption of presented experiments

An overview of the mean duration of one evolution run of all experiments in this thesis is given in the table below, in order to give an impression of how long it actually takes—to date—to obtain solutions for the respective kind of circuit. Note, that in the case of complex fitness evaluation, e.g. in those cases where a fourier transform is done or simply one test mode delivers numerous fitness values, the time consumption may further increase.

8.1.3 Influence of the Parasitic Effects and its Consequences

smooth improvement on the FPTA vs. sudden jumps in simulation

The mean parasitic on-resistance of the transmission gates, which are used for interconnecting nodes on the FPTA, is included in the SimFPTA, with the intention to provide similar conditions for evolution on both substrates. Despite the fact that the evolution runs end up in the same regions of fitness for both substrates, it is observed that the circuits on the FPTA are more 'smoothly' improved by evolution than on the SimFPTA, where rather sudden jumps occur in the course of the fitness. Thus, it can be stated that the presence of parasitic effects in the case of real hardware results in a smoother improvement of the fitness, as can be seen from the example in figure 8.3. On the one hand, such a smoother improvement is considered to be advantageous to the algorithm, since the output voltage characteristic can be changed in smaller steps and therefore, e.g. the slightest change of the W/L ratio of one of the transistors can possibly be exploited. On the other hand, by investigating resulting circuits for the logic gates and the comparators, shown in chapter 6, section 6.4.5, it will be seen that, if the selection is oriented excluding at the slightest change of the shape of the output voltage characteristic, this can also lead to unwanted solutions. It is therefore suggested that it would be most beneficial, if it was possible to design a substrate and/or algorithm which allows to smoothly improve the circuit without relying on parasitic effects. One possibility could be to optimize the W/L of all transistors of a candidate circuit in an intermediate step.

selection should not rely only on rms

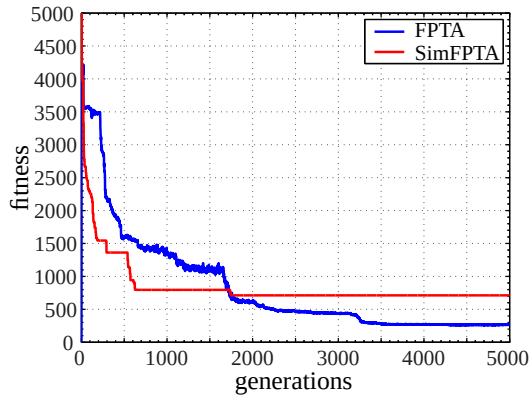


Figure 8.3: *Fitness courses for typical example runs with the FPTA and the SimFPTA respectively, are depicted on the left hand side.*

8.2 Comments on the Current Architecture

8.2.1 Advantages

evolving circuits on transistor level

The fine grained architecture of the Heidelberg FPTA, i.e. the possibility to evolve circuits on the transistor level, is considered to be more interesting for research on EAs and unusual circuit topologies than an FPTA with more complex configurable cells, e.g. tuneable filters or amplifiers ([8, 79]). Despite the fact that with more complex cells, which are aimed at predefined tasks, it is more feasible to find good solutions for just the task they are designed for, the idea of being able to almost freely interconnect single transistors is intriguing. Thereby, it is intended not only to find working circuits but, in turn, possibly learn new ways of analog design from evolution. Obviously, this is not possible in the case of a substrate with a predefined or too constrained architecture. Unfortunately, the disadvantage of increasing configurability is that the presence and influence of parasitic effects increases at the same time, although there are examples where EAs were able to use just those parasitic effects to build circuits with surprising properties ([30, 60, 78, 87]). Thereby, the most famous example is the tone discriminator experiment of Adrian Thompson [85], who achieved to evolve analog functionality on a digital substrate (FPGA). Nevertheless, it has to be stated that, to date, Thompson's ground-breaking experiment remained unique.

aiming at new design principles

8.2.2 Shortcomings and Desired Features

limited routing and connecting capabilities

The main shortcomings surely are the limited routing and component connecting capabilities of the current transistor array. Direct connections between transistors, i.e. with the minimum required number of 2 transmission gates, can only be realized between neighboring cells. Further, connections can only be established with the local routing capabilities of the chip, resulting in at least one additional switch per cell, which is added to the respective wire. Further, the four outside connections of the FPTA cells are shared between the three transistor terminals and the six possible routes. Thus, in the case of manually configuring a more complex circuit on the transistor array a relatively large area

is needed, compared with the number of actually needed components. E.g. a comparator with 7 transistors requires at least 6×6 transistor cells, thus, in this case, only about 1/5 of the available components are actually used. Consequently, an improved routing scheme is desired, which makes it on the one hand possible to connect distant transistors via a few switches and, on the other hand, does not share all its nodes with the transistor terminals.

high source and drain resistances

The second disadvantage is the presence of switches at both, source and drain terminal of the transistors. As a consequence of this, it is not possible to configure a circuit on the FPTA, where two transistors are connected to exactly the same source potential, which would be desireable e.g. in the case of a differential pair, representing a differential input stage. Despite this, in the OP experiments from chapter 7, section 7.4, the EA was surprisingly rather able to find a differential input stage, than an additional gain stage. Nevertheless, from the view of analog design, it is still desired to have real differential pairs at hand, although it can only be suggested that it would also help the EA in finding better solutions. Moreover considering the principles of analog circuit design, a more net oriented connection scheme is desired, instead of a cell oriented one.

a test-bench with extended features

Lastly, there is a need for more flexible IO cells, or, more generally spoken, a test-bench with extended features. It would be beneficial, if it was possible to attach test-benches, which include also passive components (resistors, capacities) to the evolving circuits. Alongside with supplies for reference voltages and the possibility of measuring the current of some nodes, this would make it possible to measure properties like gain or the characteristic of a configurable transistor directly on the chip. E.g. the evaluation of OPs would be significantly improved by a gain measuring. Furthermore, in a voltage net based routing environment, different accurate reference voltages could be provided to the circuit under development, which would be expecially beneficial for e.g. filters, DACs and ADCs ([55, 103, 105]).

8.3 A Proposal for Improvements

This section is intended to present some thoughts about an improved FPTA architecture and a possibly beneficial alternative genetic representation for the analog circuits in the future. It is not yet proven by implementation and experiments that the proposed architecture or genetic representation will be more powerfull. Nevertheless, this is supposed to be, based on the experience gained from the experiments in this thesis.

8.3.1 New Configurable Architecture

Architecture

The proposed FPTA architecture is depicted in figure 8.4. For this possible new architecture, it is tried to avoid or at least reduce the shortcomings of the current chip, that are described in the previous section 8.2. Thus, it is tried to, first, reduce the number of switches for distant connects while maintaining connectivity. Second, a rather net-based routing scheme is provided, instead of a cell-based one and last, groups of configurable transistors are, avoiding the switches in between, directly attached to common source nodes, as shown in figure 8.6. As a consequence of this, contrary to the current FPTA

reducing the no. of switches

realizing a net-based routing scheme

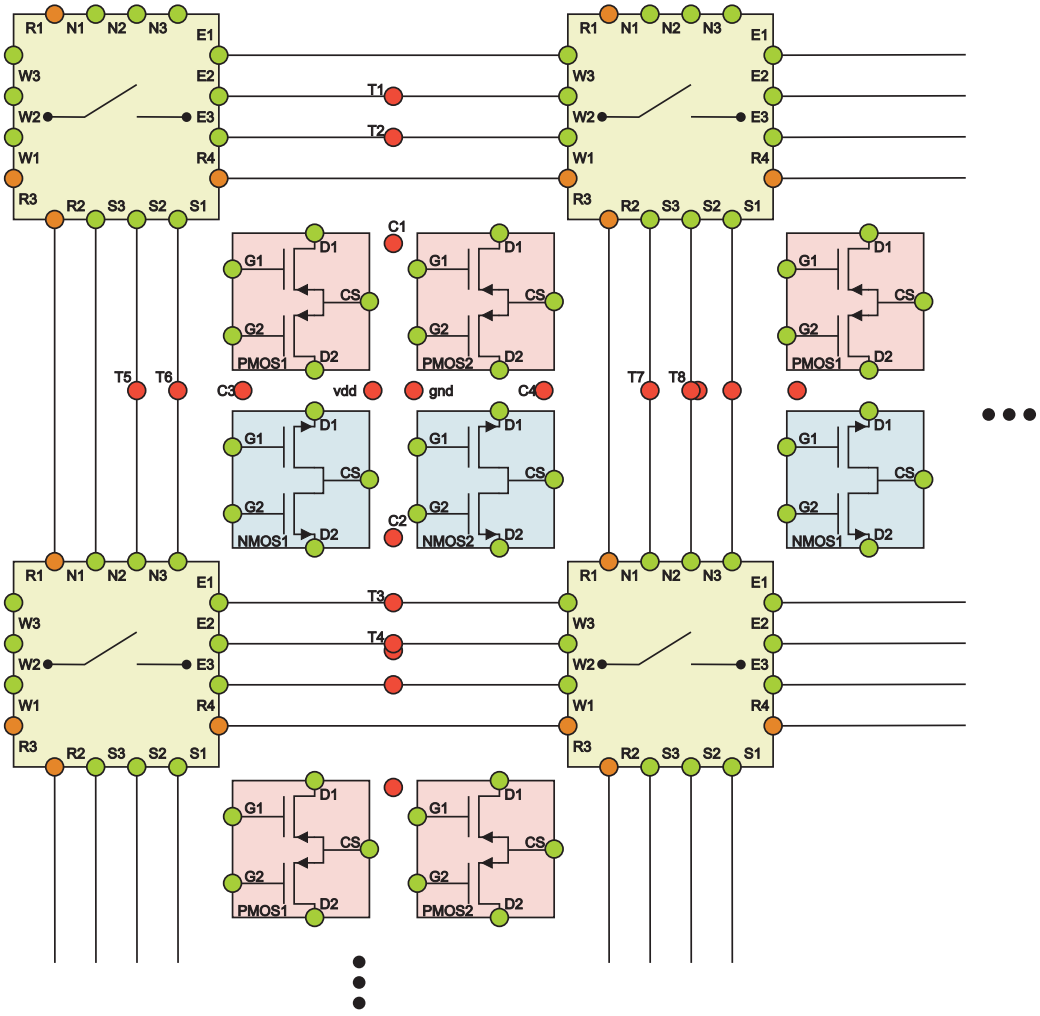


Figure 8.4: *The upper left corner of the proposed PropFPTA architecture is depicted. For the new architecture, the aim is to avoid or at least reduce the shortcomings of the current chip. Thus, it is tried to, first, reduce the number of switches for distant connects while maintaining connectivity. Second, a rather net-based routing scheme is provided, instead of a cell-based one. Last, groups of configurable transistors are, avoiding the switches in between, directly attached to common source nodes, in order to provide real differential pairs.*

architecture, transistors that shall not contribute to the circuit are not removed by closing a switch, but have to be switched off by their gate voltage: vdd $\mapsto$ PMOS is closed and gnd $\mapsto$ NMOS is closed. Despite those turned-off transistors represent a capacitive load for the node, to which they are connected, it has to be considered that this capacitive load would be even doubled in the case of switches, which are realized with transmission gates. In order to facilitate distinguishing between the current and the proposed architecture, the latter device is referred to as PropFPTA throughout the remainder of this thesis. *the PropFPTA*
Furthermore, it is intended to implement and test the proposed architecture with the modelmodelingework, presented in chapter 5 and proven to work in this chapter, section 8.1,

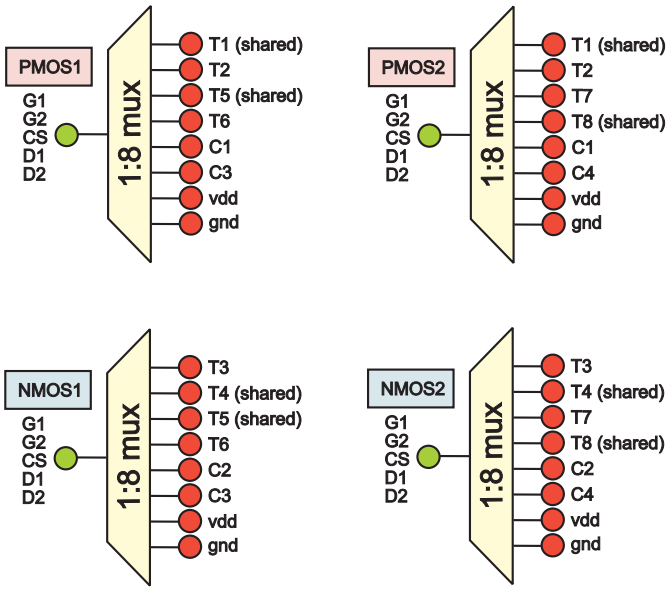


Figure 8.5: *The depicted multiplexers illustrate the possible target nodes for the 5 terminals of all 4 configurable transistor devices respectively.*

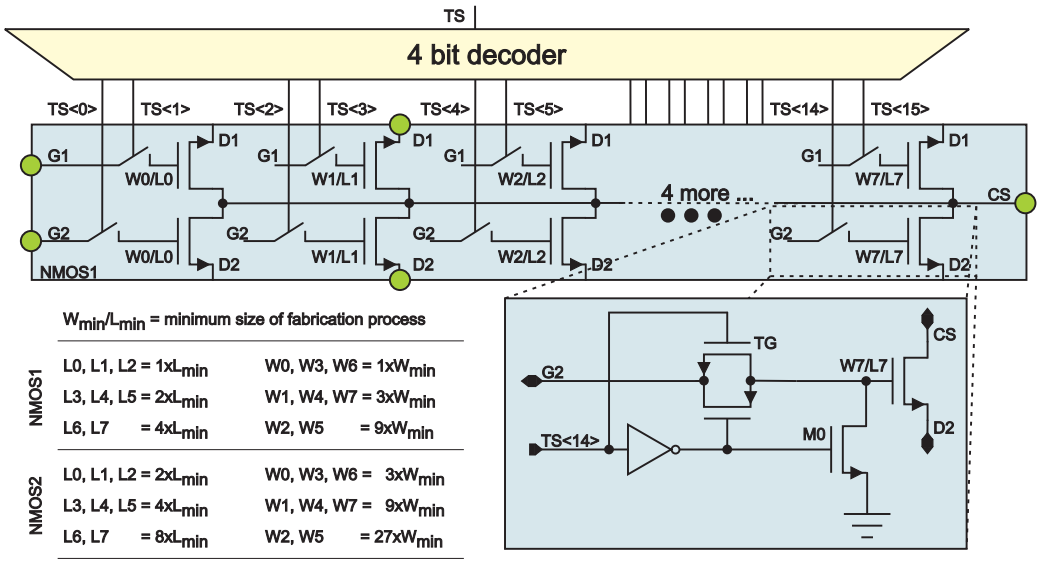


Figure 8.6: *As depicted above, the 16 transistors are switched on or off according to the demultiplexed 4 configuration bits. The device can either be operated in '5 terminal mode', providing a configurable real differential pair or in '3 terminal mode', providing a wider range for W/L. Thereby, the '3 terminal mode' is achieved by simply connecting both gates and drains to the same target node. In the case of the PropFPTA it is possible to use transistors with different L in parallel, which is neither known to be beneficial, nor known to be harmful, thus, interesting to investigate.*

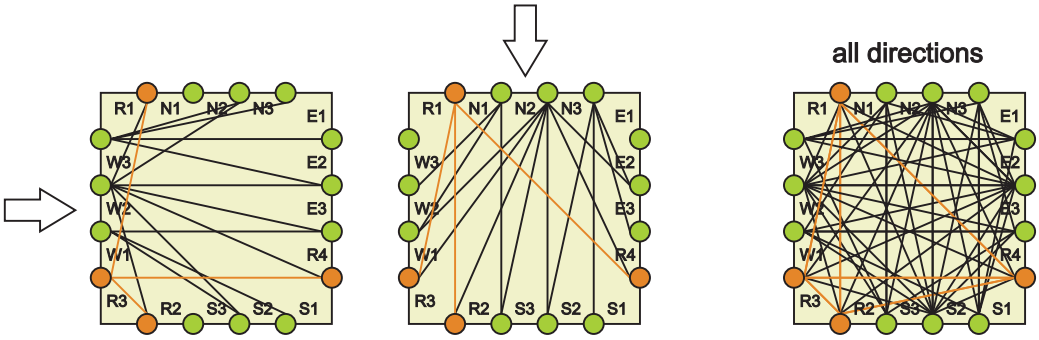


Figure 8.7: *The full connectivity of the switch boxes is depicted on the right hand side. Additionally, on the left hand side, merely the possible routes from the perspective of the east and the north is shown, in order to provide a better overview.*

	FPTA	PropFPTA
transmission gates per border node (routing)	6	12(routing) 10(border) 20(shared)
transmission gates per border node (terminals)	6	0(routing) 10(border) 10(shared)
accessible transistors (cells) per border node	2(2)	0(routing) 4(2)(border) 8(2)(shared)
transmission gates terminal	24	16
no. of configuration bits (routing)	6	6
no. of configuration bits (terminals)	9	15($\times$4)
no. of configuration bits (W/L)	7	4

Table 8.2: *The number of transmission gates, which are connected to the nodes and transistor terminals of the FPTA and the PropFPTA respectively, is listed above. It can be seen from the table that the sum of parasitic capacitances of the switches is not expected to be significantly greater for the PropFPTA, compared with the FPTA. Nevertheless, it is supposed that the enhanced routing capabilities and the 5 terminal configurable differential pairs will yield improved circuits.*

in the future. Other proposals on evolvable hardware architectures and according routing schemes can be found in [62, 79]. Further, an interesting approach for breaking the resistive barriers, created by the configuration circuitry, by using microelectromechanical systems (MEMS) is presented in [20].

The Building Blocks

As can be seen from figure 8.4, the architecture is designed in a manner, that it is again possible to create a scalable, two dimensional array with the building blocks. Thereby,

four configurable transistors: 2 PMOS and 2 NMOS

one building block consists of four configurable transistors, 2 PMOS and 2 NMOS respectively. The terminals of those configurable transistors can be connected to the red nodes, according to the schema in figure 8.5. Consequently, each of the four configurable transistors can be connected to their two next neighbors, four of their next routing lines, vdd or gnd. A direct connection to the neighboring building blocks can established with the shared nodes, marked as *shared* in figure 8.5. As described in figure 8.6, the configurable transistors themselves feature 5 terminals, namely a common source of 2×8 symmetrically arranged transistors and a common gate/drain of 8 different transistors respectively. This provides some advantageous configuration possibilities: first, in *'5 terminal mode'*, the cells can be used as real differential pairs. Second, in *'3 terminal mode'*, the cells can be used as single transistors with a wider W/L range. Third, it is possible to configure the four configurable transistors of one building block as e.g. inverter, current source/mirror or even differential input stage without the need for additional routing. Nevertheless, the substrate is still fine grained, in the sense of single configurable transistors. Furthermore, one configurable transistor of the same type features small W/L ratios, while the other one features greater W/L ratios, in order to provide a wide W/L range without attaching to many components to the same node. A comparison with the configurable transistor cells of the FPTA is given in table 8.2.

5 device terminals: 1 common source, 2 gates and 2 drains

5 terminal mode and 3 terminal mode

Routing Concept

separated routing capabilities with switch boxes

The routing capabilites are separated from the actual configurable transistor cells and the proposed architecture is inspired by the routing scheme of FPGAs. Thus, as shown in figure 8.7, switch boxes are used for creating a great variety of networks of voltage nodes. As the connections between the routing nodes ($R_{1\text{-}4}$) cannot be directly connected with the configurable transistors, the latter nodes are intended to provide additional routing capabilities for the circuit's IO, external reference voltages or for bypasses. As yet, a total of 52 switches for each switch box is proposed. Thereby, each routing node ($R_{1\text{-}4}$) is attached to 6 switches, the border nodes (N,S,W,$E_{1,3}$) to 5 switches and the shared border nodes (N,S,W,E_2) to 10 switches. It has to be found out, if the proposed connectivity will be sufficient or not. Considering 6 configuration bits for each switch box, there are still free ressources for up to 12 additional switches. A comparison with the FPTA is given in table 8.2.

Concluding Remarks

It can be seen from table 8.2 that the sum of parasitic capacitances of the switches is not expected to be significantly greater for the PropFPTA, compared with the FPTA. Nevertheless, it is supposed that the enhanced routing capabilities and the 5 terminal configurable differential pairs will yield improved circuits, due to the greater variety of possible configurations. Moreover, it will be interesting to see, to what extent the configuration possibilities of the new architecture are actually used in the case of circuit synthesis with EAs. Lastly, it has to be remarked that, in addition to the configurable array, IO circuitry, e.g. sample and hold buffers and reference voltage sources, will be necessary to operate the architecture. Such IO circuitry can be easily attached to the unused nodes of the border switch boxes and can subsequently be routed to inner nodes of the substrate.

8.3.2 Alternative Genotype Representation for the Current FPTA

a more development based genetic encoding

It is supposed that a genetic representation, which is not as closely related to the phenotype as the current one, would be beneficial, since small changes in the genotype could result in greatly different configuration bit-strings for the phenotype. This will not necessarily result in circuits with completely different functionality, if e.g. only the placement of the transistors is changed. Thereby, a possible genetic representation is simply a list of components, their size and node numbers, that define the connectivity. Consequently, the complexity of creating the phenotype is moved from the EA to an additional mapping algorithm. In the case of a configurable hardware substrate, the latter mapping provides an additional advantage: if the mapping contains some randomness in the placing of the components, the robustness of the evolving circuits will be improved, since the circuit is not depending on the properties of always the same components, which are depending on the fabrication process themselves (fixed pattern noise).

mapping to the substrate

In the case of the current FPTA substrate, the same algorithm, which is used to automatically create schematics from a netlist, could be used for mapping a circuit to the transistor array. Assuming a maximum amount of $5 \times 5 = 25$ transistors in the genome,

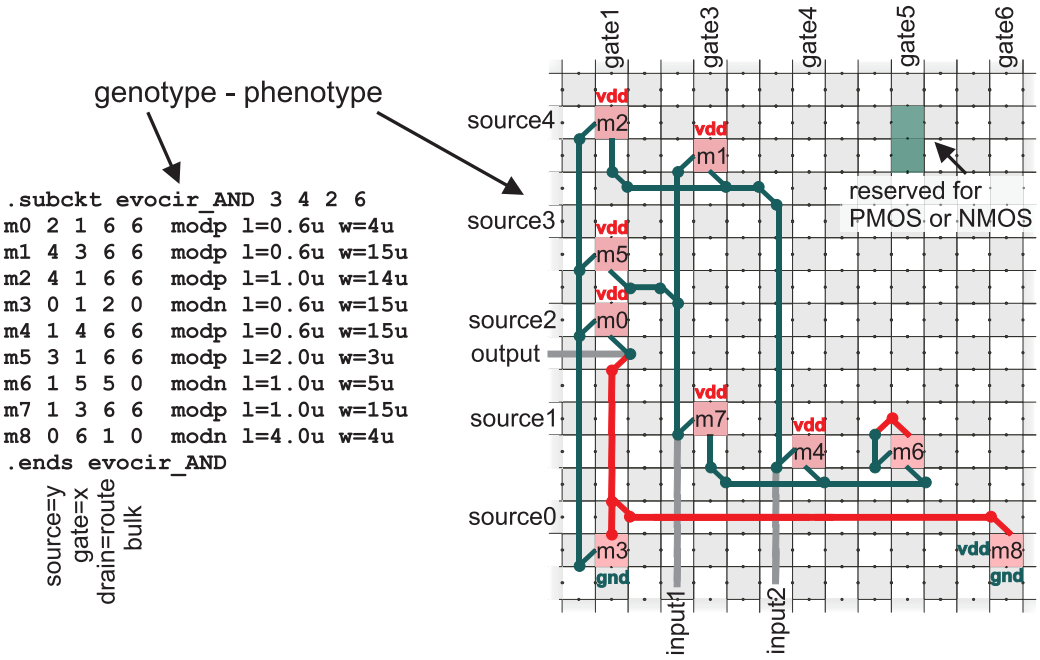


Figure 8.8: *An example of how a netlist based genetic representation can be mapped to the FPTA. In order to achieve this, an area of 3×3 FPTA cells is reserved for each component of the netlist. Subsequently, the coordinate of the transistors within the array is defined by their gate node numbers (x-coordinate) and their source node numbers (y-coordinate). Depending on the MOS type, an according FPTA cell is configured by selecting the W/L, connecting the gate to the west, the source to the north and the drain to the south. First, gates with equal node numbers are connected with vertical NS routes. Second, drains are connected with WE routes. Third, the free routing capabilities can be used by a simple routing algorithm, to connect the sources to their respective target net.*

and constraining the number of transistor terminals of the same kind, which are allowed to connect to the same node, to 5, it should be possible to realize all possible configurations on the current FPTA. An example is given in figure 8.8. On the one hand, a great advantage of the latter approach would be the possibility to use the whole array area without exponentially increasing the search space, and, on the other hand, it is supposed that a genome with 25 freely interconnectable transistors still provides sufficient configurability. Besides, with a smart mapping algorithm, the number of transistors can probably be further increased. Compared to using a simulator for evaluation, in the case of 25 configurable transistors, the speed of hardware still significantly outperforms the speed of simulation by an estimated factor of about 6, as shown in table 8.1.

Summary and Outlook

The focus of this thesis has been set on the design automation of CMOS transistor circuits on field programmable (configurable) transistor array architectures (FPTAs) by means of evolutionary algorithms. Compared with their widely used and elaborate digital counterparts, namely FPGAs, FPTAs remain currently still on a basic level of research. Within this thesis, the Heidelberg FPTA, which has been designed by Jörg Langeheine [49], was used as a substrate for the evolution of analog circuits on the transistor level. It consists of 16×16 configurable PMOS and NMOS transistor cells, that are arranged in a checkerboard pattern. The routing capabilities of the chip and the possibility to configure the characteristic of the transistors allow for realizing a great variety of circuits on the FPTA. In combination with a real time configuration and test environment, the FPTA is suitable for efficient chip-in-the-loop evolution experiments. The idea behind utilizing a fine-grained substrate is to avoid biasing evolution towards conventional designs and to model the analog design process more closely. Besides, it is a notable fact that the Heidelberg FPTA is the only fine-grained configurable analog substrate in the world and there is only one other similar chip, which however features more complex cells, developed by the group of A. Stoica at the JPL.

the Heidelberg FPTA chip

evolution on the transistor level

This thesis tackled three main problems within the research area: first, in order to improve understanding of circuits that have been evolved on the FPTA, they should be reduced to relevant components and be transferable to other technologies. Since this work was aimed to provide the possibility to derive new design principles from evolution and for closing the loop to human design, it has been desireable to draw clear schematics of evolved circuits. Second, analog circuit design is a very complex task, where numerous variables have to be optimized in parallel. Thus, a multi-objective approach suggested itself for successfully handling circuits with the latter standards, i.e. with high complexity. Third, it was interesting to investigate the influence of the FPTA architecture on the success of evolution. Thus, a tool was desired, which provided the possibility to create models of possible FPTA architectures and to immediately use them within the evolution framework.

main aims of this thesis

Addressing the problem of transferability of the found solutions, new variation operators have been developed within this thesis in order to make it possible to synthesize circuits, which are not bound to the particular substrate or architecture on which they have been evolved. The EA that runs the new variation operators is denoted as the *Turtle GA* and the one that runs straight forward implementations is referred to as the *Basic GA*. Due to the intrinsic realism of configurable hardware, the set of possible configurations equals the number of feasible solutions, although not all of those circuits are necessarily valid outside the chip. In turn, if a circuit shall be successfully evolved extrinsically, it

transferable circuits

will have to follow the constraints of a SPICE simulator used, but is not generally realizeable on the FPTA. Consequently, since the aim was to synthesize transferable, reusable circuits and to possibly derive new design principles, the new variation operators have been implemented in a manner that the candidate circuits were feasible on both the chip and in simulation. The new operators have been successfully applied to the evolution of logic gates and comparators and it was possible to verify the resulting circuits in simulation, hence, to transfer them to other technologies. The found solutions have thereby been extracted into netlists and a SPICE simulator could then be used for the off-chip verification. It has been interesting to see that the performance of good solutions was not getting much worse, if a simplified simulation model was used for testing, i.e. most parasitic effects of the chip had been omitted, although about 50 % of all circuits did no longer work in simulation. Additionally, it has been shown, that the new genetic operators performed equally well in finding good solutions for logic gates and comparators as those of the previously used straight forward implementation.

new variation operators

off-chip verification in simulation

Aside from verifying the resulting circuits in simulation, they have been tested with two other substrates. It was observed that for both the *Basic GA* and the *Turtle GA* all found solutions performed equally well on all substrates. This was on the one hand a positive result, since the solutions featured a certain robustness by not depending on a particular substrate although, on the other hand, the hope that the EA would be able to create and optimize circuits by using some kind of 'hidden' features of the hardware was not corroborated.

testing on different substrates

Unfortunately, in the field of evolutionary electronics, there are as yet no suitable benchmarks available, that allow for comparing the performance of algorithms of different research groups independently of the problem. Thus, a statistical method for characterizing variation operators was proposed in chapter 6, with which expected probabilities can be calculated for whether the respective variation operator improves or degenerates individuals. These expected probabilities can help to anticipate convergence speed and the course of fitness. Table 6.3 shows that the variation operators of the *Basic GA* and the *Turtle GA* featured a similar performance.

characterization of variation operators

A further remarkable achievement of the *Turtle GA* is that the ressource consumption of the resulting circuits has been reduced, compared with the *Basic GA*. The latter effect got more obvious—and more important—with increasing complexity (see figures 6.8 and 6.11: XOR, XNOR, comparator and also the oscillators and OPs from chapter 7). Furthermore, due to the reduced number of transistors, human-understandable schematics of good solutions could be drawn. The generation of the latter schematics has been done automatically within the circuit editor of the Cadence design framework. As a consequence of this, it has been possible to analyze the resulting circuits and to investigate how the algorithm is solving problems on the hardware substrate. It is a satisfying result that the EA was even able to find a few solutions for the logic gates and the comparators, which are performing well in a simulation where no parasitics of the chip are included at all. These solutions are indeed independent from the FPTA and for the logic NAND, AND and OR, it is possible to understand how the evolved circuits actually work. This is a valuable result, since it provides some insight into the optimization process on the chip and thereby suggests improvements for implementations of the EA in the future.

reduced ressource consumption

schematic extraction: closing the loop to human design

The next aim was to tackle the evolution of more challenging analog circuits, namely oscillators and operational amplifiers (OPs). The evolutionary algorithm has been ex-

tended with the capability of multi-objective optimization, in order to be able to successfully find solutions for advanced problems where numerous—often even contradicting—properties need to be optimized in parallel. With this new approach, it has been possible to efficiently explore the design space and successfully find solutions for comparators, oscillators and operational amplifiers. The evolution experiments for finding solutions for comparators have been carried out in order to be able to compare the performance of the MO approach to the non-MO implementations. In both cases equally good solutions are found and although the MO algorithm converges slower, it achieves to maintain greater diversity within the population, which is crucial for avoiding premature convergence.

multi-objective optimization

avoiding premature convergence

In the case of the oscillators, it is a specific feature of the setup that there was no input present. Thus, no external stimulus could be exploited by the EA, which significantly increased the difficulty level. It has been observed that it is necessary to formulate open fitness criteria in order to exploit any variation of the output voltage and to get an oscillation started. The results for the oscillators revealed a further advantage of multi-objective optimization: the resulting population of a successful run yielded different solutions with different frequencies. Hence, in the case of MO it has been possible to harvest numerous trade-off solutions from the resulting population instead of only one, if the formulation of the fitness function was not too restrictive and allowed for certain variations. Unfortunately, none of the found oscillators works in simulation outside the chip.

harvesting numerous solutions

According to the lessons learned from the logic gates and comparators experiments, the setup for the evolution of operational amplifiers has been modified in a way that the inputs are always connected to the gates of either a pair of NMOS or PMOS transistors. As a consequence of this, the algorithm reliably converged and the behavior of the resulting circuits was in all cases at least similar on the chip and in simulation. The resulting OPs with a good performance were extracted into netlists and were simulated outside the substrate on which they have been evolved. However in the case of multi-objective optimization, it can not easily be decided which individual is the best to pick for further testing. Due to this, the performance of all resulting individuals on the FPTA has been compared with basic manually made reference designs and the individual, which outperformed the reference designs in most objectives was considered as the best solution. More than half of these best solutions have been performing equally well on the chip and in simulation in up to 6 out of 12 objectives and they achieved fitness values comparable to those which were obtained by the basic human reference designs measured on the chip.

comparison with reference designs

The best OPs were again converted to clear schematics, in order to understand how they work. In this case, the schematics were manually redrawn for clarity by grouping the transistors according to their functionality. It is a promising result that for both PMOS and NMOS input, the EA (re)discovered an architecture similar to a differential input stage, which is a widely used human design. Despite this solution is already well known, it is remarkable that it has been achieved to derive a design principle from evolved circuits without including prior circuit design knowledge. Unfortunately, the algorithm failed in synthesizing additional gain stages. The reason for this was probably the lack of a suitable gain test bench, which could have delivered an according fitness value.

deriving design principles

To conclude the experiments with the FPTA, it is contenting to see that the presented algorithms are able to evolve transferable circuits and yield some good solutions that are understandable from a designer's point of view. Moreover, complex analog circuits have been successfully evolved from scratch, i.e. on the transistor level, which was previously

achievements of the algorithm

not achieved. In the case of the OPs it was even possible to (re)discover a human design principle. Finally, the loop to human design has been closed by transferring the circuits to a simulator and by drawing clear schematics of good solutions. The fact that the evolution of comparators, oscillators and OPs is an already difficult task suggests that the *MO-Turtle GA* can be successfully applied to a variety of problems.

a comprehensive modular evolution framework

During this thesis, a comprehensive modular evolution framework has been developed in C++, which facilitates the implementation and immediate application of any module of an evolutionary algorithm. Furthermore, it allows for assessing the architecture of the FPTA by providing the possibility to develop simulation models of custom FPTA architectures and to immediately use them for experiments within the presented framework. In this case, the results obtained from the FPTA and the SimFPTA—a simplified model of the real hardware—showed equal performance for the task of synthesizing comparators. It is thereby observed that generally all circuits that were evolved on the SimFPTA perform similar on the FPTA, however, since a simplified simulation model cannot cope with any parasitic effect of the chip, the inverse is not necessarily true. Regardless of the supported view—avoiding or extensively exploiting parasitic effects—the aim should be to understand or even control the influence of those effects, in order to benefit from as much substrate properties as possible. Since evolution on real hardware is significantly faster than in simulation, it is on the one hand an advantage to use real hardware in order to quickly evaluate the performance of the algorithm used. On the other hand, the architecture of the chip cannot be changed unless a new version is designed and fabricated.

modeling FPTA architectures

Thus, the presented work aims at providing a tool for developing and testing improved FPTA architectures in the future. Once a software model of an architecture with a good performance is found, it will be possible to realize a more powerful hardware implementation.

visions

It is proposed to emphasize the research into new FPTA architectures and possibly new approaches to evolution substrates with the presented framework. If powerful FPTA architectures are found, they will become equally important as their digital counterparts, the FPGAs. Configurable analog hardware is suitable to perform any task for which interfaces to the real world are necessary, e.g. in telecommunication applications, controllers and sensors, that have to interact with the physical world. Moreover, fault tolerance and build-in self test methods can be more efficiently implemented using additional analog ressources, even for digital circuits. Here it is not necessary to waste a whole spare logic building block if there is only one single faulty transistor. Last but not least, it has always to be kept in mind, that digital functional blocks are merely an abstraction layer over analog circuitry and their performance is therefore strongly depending on analog properties. It is also imagineable to abandon even the transistors as basic components and to try to directly evolve the underlying structures, namely layers and areas of silicon, polysilicon

evolution works

and ion concentrations of a cutting edge fabrication process in the future. Lastly, it persists that, as incredible as it may seem, evolution works and it is known to achieve a lot when starting from the lowest level.

Acronyms

AC	alternating current
ADC	analog-to-digital converter
AMS	Austria Micro Systems International AG
ASIC	application specific integrated circuit
ASCII	american standard code for information interchange
BIST	build-in self test
CAD	computer aided design
CMOS	complementary metal oxide semiconductor
CMRR	common-mode rejection ratio
DRC	design rule check
DAC	digital-to-analog converter
DC	direct current
DNA	desoxyribonucleic acid
EA	evolutionary algorithm
EC	evolutionary computation
ENOB	equivalent no. of bits
FPAA	field programmable analog array
FPTA	field programmable transistor array
FPGA	field programmable gate array
FFT	fast fourier transform
GA	genetic algorithm
GP	genetic programming

gcc	gnu C compiler
HDL	hardware description language
IO	input/output
JPL	NASA Jet Propulsion Laboratory
L	length
LVS	layout versus schematic
LPE	layout parameter extraction
MO	multi-objective
NDF	non-dominated front
MOS	metal oxide semiconductor
MEMS	microelectromechanical systems
NMOS	n-type metal oxide semiconductor
PMOS	p-type metal oxide semiconductor
UML	unified modelling language
VHSIC	very high speed integrated circuit
VHDL	VHSIC hardware description language
VLSI	very large scale of integration
W	width
OP	operational amplifier
PCI	Peripheral Component Interconnect
PC	personal computer
rms	root mean square
RNA	ribonucleic acid
SKILL	silicon compiler interface language
SRAM	static random access memory
SPICE	simulation program with integrated circuits emphasis
STL	standard template library
SNR	signal-to-noise-ratio

THD total harmonic distortion

THD+N THD + noise

TM test mode

TTL transistor-transistor logic

XML extensible markup language

Appendix

Appendix A

Pseudocode

A.1 Evolution Software Framework

Algorithm A.1: Example code for deriving a custom circuit component, namely a configurable transistor (`CCMOSTransistor`), is depicted below. The setup and configuration of the component is done in the constructor of the respective class, while the genotype-phenotype mapping is implemented in additional member functions. In this example, the internal nodes correspond to the external nodes. Despite this, the differentiation between internal and external nodes is inevitable for the design of FPTA architectures, as can be seen from section 5.4.3.

```
# ccmostransistor.h
# include<componentbase.h>
class CCMOSTransistor : public ComponentBase{ }

# ccmostransistor.cpp
procedure CCMOSTRANSISTOR::CCMOSTRANSISTOR
    create and add external nodes: NODE1(23), NODE2(12), NODE3(5)
    create and add internal nodes: GATE(5), SOURCE(23), DRAIN(12)
    create and add parameters: W(4.0um), L(1.0um)
    create and add switch: CMOSTYPE(NMOS)          // also possible: PMOS
end procedure

procedure CCMOSTRANSISTOR::GETFPTAREPRESENTATION
    map the transistor to an FPTA cell with according properties
                                   // there is no one-to-one mapping in this case!
end procedure

procedure CCMOSTRANSISTOR::GETSPICEREPRESENTATION
    if CMOSTYPE is PMOS then                          // create netlist entry
        return "mx 23 5 12 bulk=vdd pmos w=4um l=1um"
    else
        return "mx 23 5 12 bulk=gnd nmos w=4um l=1um"
    end if
end procedure
```

Algorithm A.2: The following example code demonstrates how to customize the evolutionary algorithm by deriving from the basic framework, illustrated in figure 5.1. Solely those parts, which have to be implemented, are shown. A description of how to implement custom genetic operators is given in section 5.2.3 and is shown in algorithms A.3 and A.4

```
# GAExampleGA.h
class GAExampleGA : public GABaseGa{
...
private:
GaBasicPopulation oldPop
GaBasicPopulation newPop
}

# GAExampleGA.cpp
procedure INITIALIZE( )
    assign the genetic operators
    configure the evolutionary algorithm
    call the initializer of all populations
    measure and evaluate all populations
end procedure

procedure STEP( )
    swap oldPop with newPop
    for all individuals of oldPop ← 1 to population size do
        select individual from oldPop
        cross the selected individual
        mutate the selected individual
        add the selected individual to the newPop
    end for
    measure and evaluate the newPop
end procedure

# GAExampleGenome.h
class GAExampleGenome : public GABaseGenome{
private:
custom genetic coding (data structure)
}

# GAExampleGenome.h
function GETPHENOTYPEREPRESENTATION( )
                                // for example, the configuration bit string for the FPTA
    map genotype to phenotype
    return phenotype representation
end function
```

Algorithm A.3: Part I: An initializer, mutator and crossover operator, that belong together, are chosen as an example of demonstrating how to implement a set of custom genetic operators. It is assumed, that an according evolutionary algorithm framework, described earlier in this section, is already available and an according EA, with accessible population and genomes is present. Note that the initializer is shown in algorithm A.4.

```
# MutateExample.h
class MutateExample{
public:
mutate(GABaseGenome, float) : int
}

# MutateExample.cpp
procedure MUTATEEXAMPLE::MUTATE(GABaseGenome exGen, float probability)
    cast to GAExampleGenome exGen
    if random float < probability then
        mutate genes of exGen
        mark exGen for later measurement and evaluation
    end if
    return no of performed mutations
end procedure

# CrossExample.h
class CrossExample{
public:
cross(GABaseGenome, GABaseGenome, float) : int
}

# CrossExample.cpp
procedure CROSSEXAMPLE::CROSS(GABaseGenome exGen1, GABaseGenome ex-
    Gen2)
    cast to GAExampleGenome exGen1, exGen2
    randomly select crossover points of exGen1 and exGen2
    perform crossover and store results in exGen1 and exGen2
    mark exGen1 and exGen2 for later measurement and evaluation
    return error code
end procedure

# GAExamplePop.h
initializer=InitExample::initialize(GABasePop basePop)

# GAExampleGenome.h
initializer=InitExample::initialize(GABaseGenome exGen)
mutator=MutateExample::mutate(GABaseGenome exGen, float probability)
crossover=CrossExample::cross(GABaseGenome exGen1, GABaseGenome exGen2)
```

Algorithm A.4: Part II: This is the second part of algorithm A.3. Since the genomes feature their own initializers, it is also possible to put code for the genome initialization there and merely call those initializers from *InitExample*.

```
# InitExample.h
class InitExample{
public:
initialize(GABaseGenome or GABasePopulation) : int
}

# InitExample.cpp
procedure INITEXAMPLE::INITIALIZE(GABaseGenome exGen)
    reset and cast to GAExampleGenome exGen
    create genetic encoding / components for exGen
    mutate exGen randomly
    measure and evaluate exGen
    return no of created genes
end procedure
procedure INITEXAMPLE::INITIALIZE(GABasePopulation basePop)
    reset GABasePopulation basePop
    for i ← 1 to population size do
        new GAExampleGenome exGen
        create genetic encoding / components for exGen
        mutate exGen randomly
        measure and evaluate exGen
        add exGen to basePop
    end for
    return no of created genomes
end procedure
```

A.2 Variation Operators

Algorithm A.5: Part 1 of 2. The variation operators of the *Turtle GA* are described in the following pseudocode. Both, the *random wires mutation* and the *implanting crossover* are carried out recursively and produce circuits on the current FPTA architecture, which contain no floating nodes. For part 2 see algorithm A.6 in this appendix.

```
procedure STARTRANDOMWIRE
    select a random outside node of a random cell as starting point
    decide whether to start in create mode or erase mode
    if starting point = N,S,W,E then
            // the recurse methods have to be carried out twice, otherwise the starting
        node possibly floats
        2 × recurseRandomWire(cell, node, erase/create mode)
    elsereturn no node selected
    end if
end procedure

function RECURSERANDOMWIRE(cell, node, erase/create mode)
    decide, whether to prefer current cell or neighbor cell for proceeding
    decide, whether to prefer connecting to N,S,W,E or to source,gate,drain
    if create mode then
        if current node has at least 2 connections then
            return stop condition occurred
        end if
        select random unconnected target node (N,S,W,E,source,drain,gate)
        connect node to target node
    else if erase mode then
        if current node has 0 or at least 2 connections then return stop condition oc-
            curred
        end if
        select random connected target node (N,S,W,E,source,drain,gate)
        disconnect node from target node
    end if
    if selected target node = N,S,W,E then
        recurseRandomWire(cell, target node, erase/create mode)
    else if selected target node = source,gate,drain then
        recurseRandomTerminal(cell, target node, erase/create mode)
    end ifreturn
end function
```

Algorithm A.6: Part 2 of 2. Part 1 is given in algorithm A.5 in this appendix.

```
function RECURSERANDOMTERMINAL(cell, target node, erase/create mode)
    if create mode then
        select two random unconnected target nodes (N,S,W,E,vdd,gnd)
        connect the two remaining terminals to respectively one target node
    else if erase mode then
        decide whether to proceed or to rewire
        if rewire transistor then
            switch to create mode
            select random unconnected target node (N,S,W,E)
            connect the current terminal to the selected target node
        else if continue erasing then
            select two random connected target nodes (N,S,W,E,vdd,gnd)
            disconnect the two remaining terminals from those target nodes
        end if
    end if
    if first selected target node = N,S,W,E then
        recurseRandomWire(cell, first target node, erase/create mode)
    end if
    if second selected target node = N,S,W,E then
        recurseRandomWire(cell, second target node, erase/create mode)
    end ifreturn
end function

procedure IMPLANTCROSSOVER(individual1, individual2)
    select randomly positioned blocks of cells of the same random size of individual1
        and individual2
    exchange those blocks between individual1 an individual2
    make a list of all floating nodes
    for i ← 1 to sizeoflist do
        recurseRandomWire(cell, node(i), erase/create mode)
    end for
end procedure
```

Appendix B

Additional Schematics of Evolved Circuits

B.1 Logic Gates

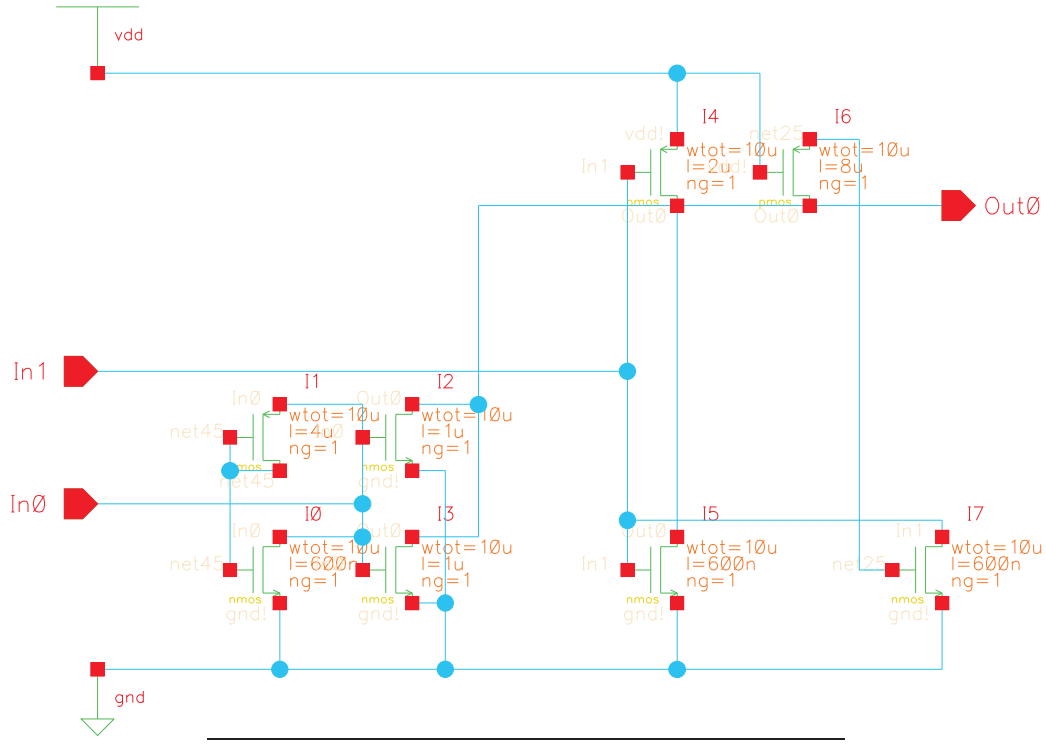
The best solutions for the logic AND, NAND and OR, presented in chapter 6, will perform well outside the FPTA, even if only plain transistors are considered in the simulation. Unfortunately, the solutions for the logic NOR, XOR and XNOR will only work in simulation, if at least the mean on-resistance of the transmission gates is included in the netlist. Possible reasons for this failure are given in the captions of the respective schematics. Note that the presented schematics are generated from the best solutions for the respective logic gate and all configuration circuitry is omitted, in order to be able to provide clear schematics. The results are obtained with the *Turtle GA*.

B.2 Comparators

Although the best solutions for a comparator (chapters 6 and 7), obtained with the *Turtle GA* and the *MO-Turtle GA* respectively, feature a good performance on the chip, both solutions fail in the simulation with plain transistors. Therefore, it is not yet understood how the circuits are actually working. Note that the circuits will perform equally well in simulation and on the chip, if the parasitic resistances of the transmission gates are included in simulation.

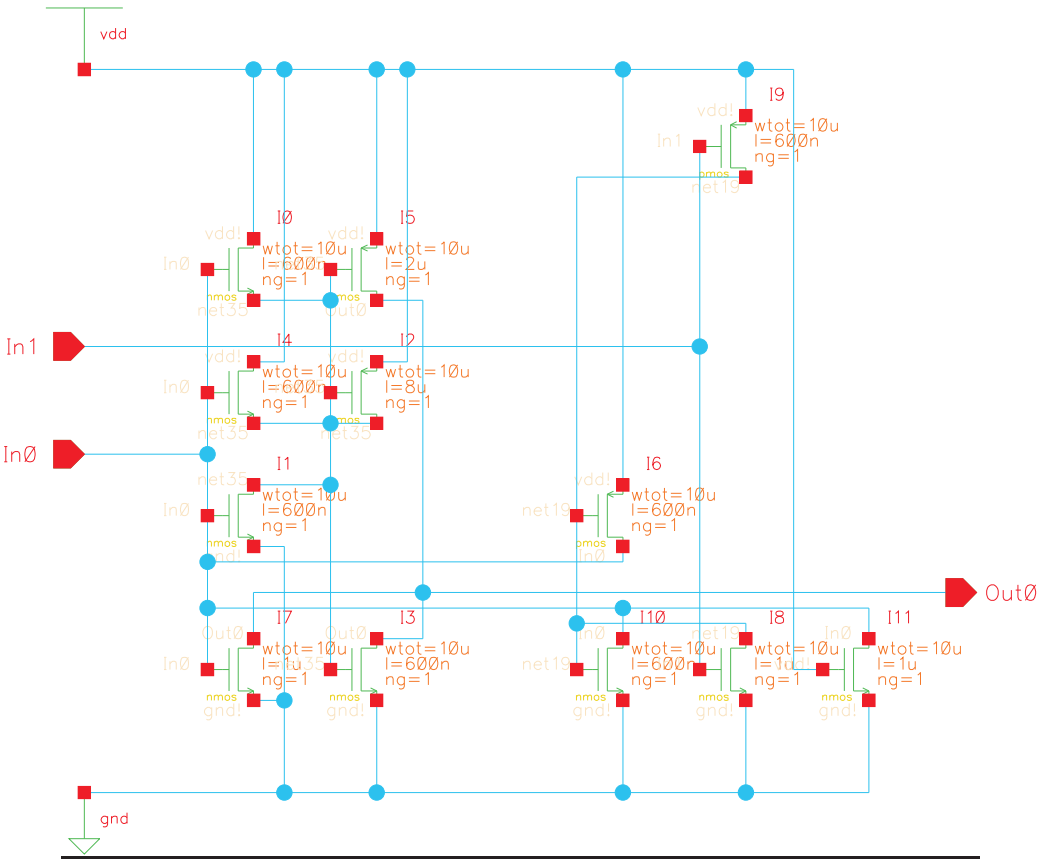
B.3 Operational Amplifiers

It is a nice result that the presented solutions for OPs with NMOS and PMOS input respectively, are performing equally well on the FPTA and in simulation, although it has not yet been achieved to figure out their operation principle. As can be seen from figure B.6 and figure B.8, both designs consist of more than 30 transistors which already stands for a high level of complexity.



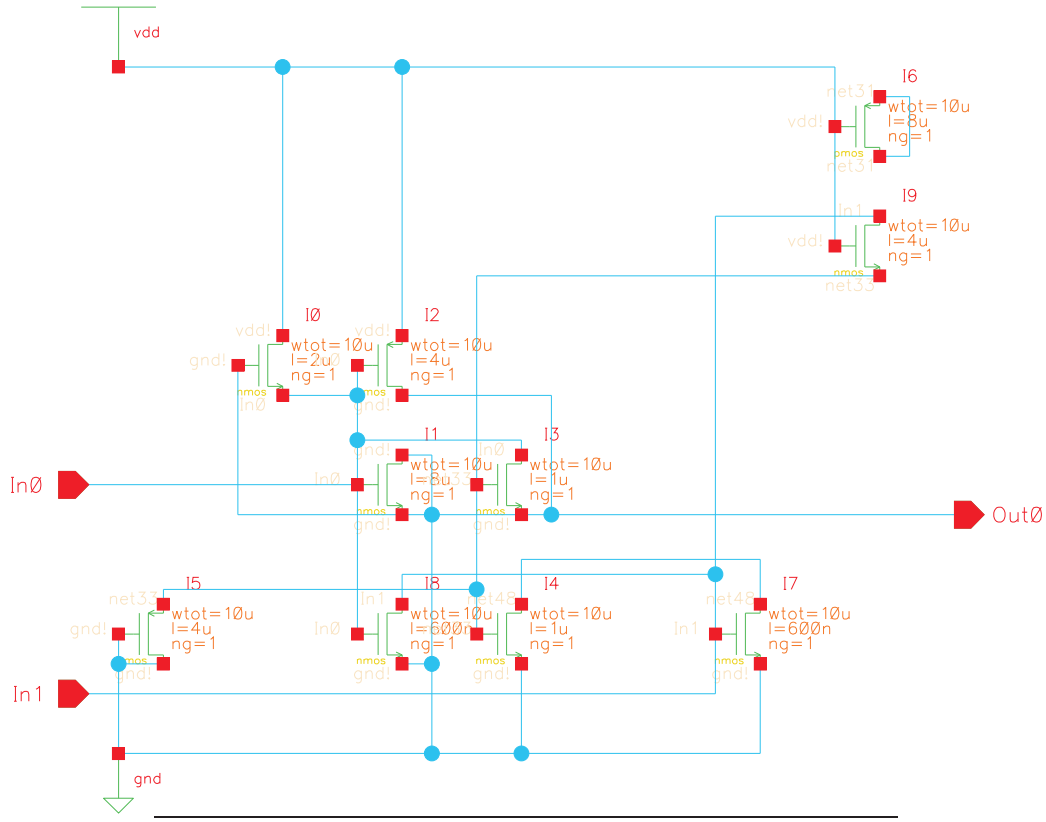
	I0	I1	I2	I3	I4	I5	I6	I7
type	N	P	N	N	P	N	P	N
W/L [μm]	$\frac{15}{0.6}$	$\frac{2}{4}$	$\frac{15}{1}$	$\frac{15}{1}$	$\frac{9}{2}$	$\frac{15}{0.6}$	$\frac{9}{8}$	$\frac{4}{0.6}$

Figure B.1: *Schematic of the best evolved NOR. Although the circuit features a good performance on the FPTA, it fails in simulation, because the interconnected gates of I0 and I1 are floating. Unlike in simulation, floating nodes are not in undefined states on the chip, due to leakage currents.*



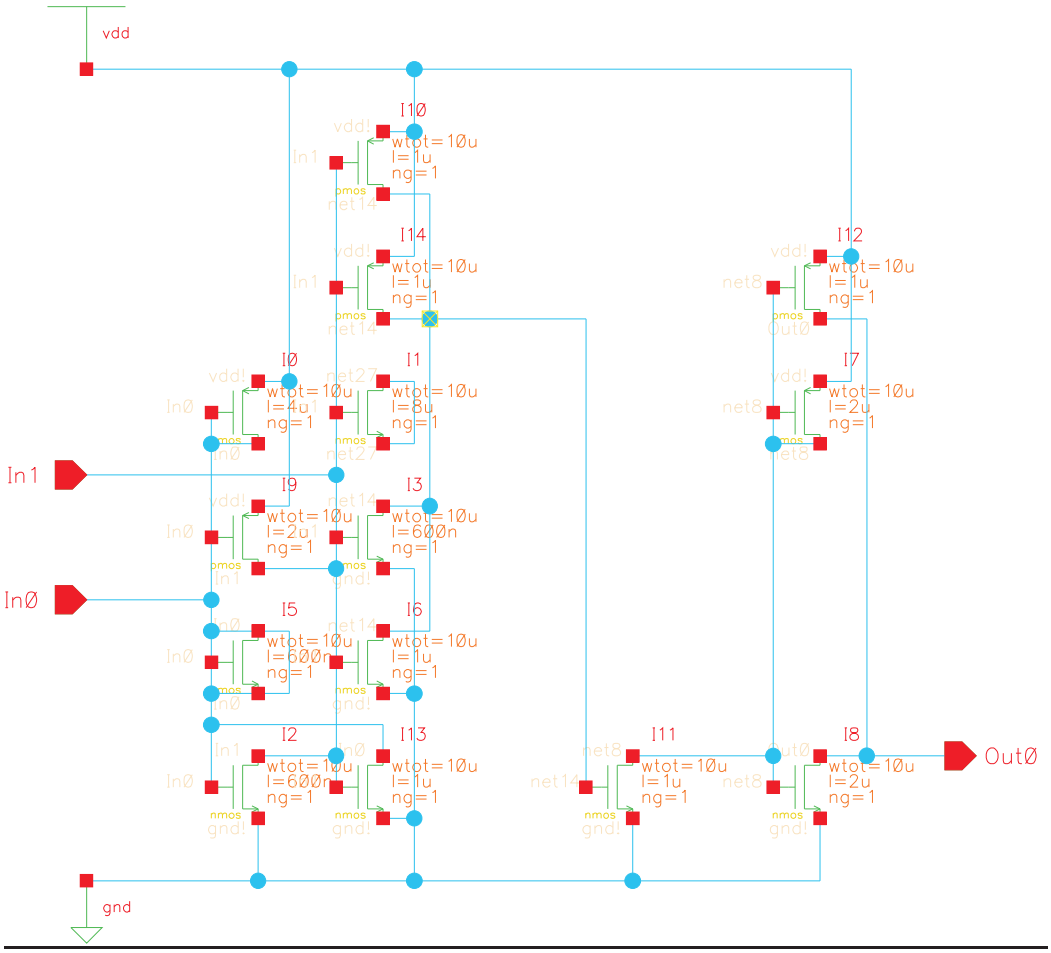
	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11
type	N	N	P	N	N	P	P	N	N	P	N	N
W/L [μm]	$\frac{15}{0.6}$	$\frac{8}{0.6}$	$\frac{4}{8}$	$\frac{15}{0.6}$	$\frac{15}{0.6}$	$\frac{6}{2}$	$\frac{7}{0.6}$	$\frac{15}{1}$	$\frac{15}{1}$	$\frac{11}{0.6}$	$\frac{13}{0.6}$	$\frac{11}{1}$

Figure B.2: *Schematic of the best evolved XOR. Unfortunately, it is neither yet understood why the circuit works on the FPTA nor why it fails in simulation with plain transistors.*



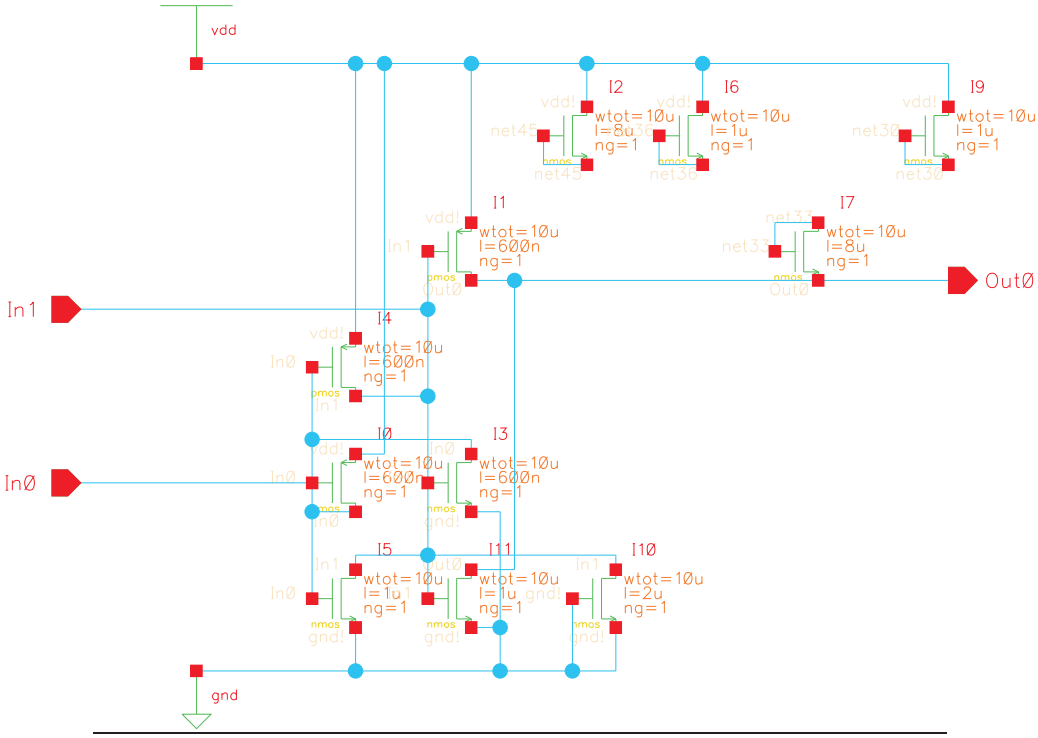
	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9
type	N	N	P	N	N	P	P	N	N	N
W/L [μm]	$\frac{6}{2}$	$\frac{10}{8}$	$\frac{11}{4}$	$\frac{15}{1}$	$\frac{7}{1}$	$\frac{2}{4}$	$\frac{3}{8}$	$\frac{11}{0.6}$	$\frac{8}{0.6}$	$\frac{5}{4}$

Figure B.3: *Schematic of the best evolved XNOR. Not yet understood.*



	I0	I1	I2	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14
type	P	N	N	P	N	N	P	N	P	P	N	P	N	P
W/L [μm]	$\frac{15}{4}$	$\frac{0}{8}$	$\frac{3}{0.6}$	$\frac{0}{2}$	$\frac{13}{0.6}$	$\frac{6}{1}$	$\frac{12}{2}$	$\frac{15}{2}$	$\frac{14}{2}$	$\frac{15}{1}$	$\frac{13}{1}$	$\frac{13}{1}$	$\frac{5}{1}$	$\frac{9}{1}$

Figure B.4: *The best comparator evolved with the Turtle GA.*



	I0	I1	I2	I4	I5	I7	I8	I9	I10	I11	
type	P	P	N	N	P	N	N	N	N	N	N
W/L [μm]	$\frac{15}{0.6}$	$\frac{5}{0.6}$	$\frac{15}{8}$	$\frac{6}{0.6}$	$\frac{5}{0.6}$	$\frac{6}{1}$	$\frac{5}{8}$	$\frac{0}{0.6}$	$\frac{5}{1}$	$\frac{3}{2}$	$\frac{15}{1}$

Figure B.5: *The best comparator evolved with the MO-Turtle GA.*

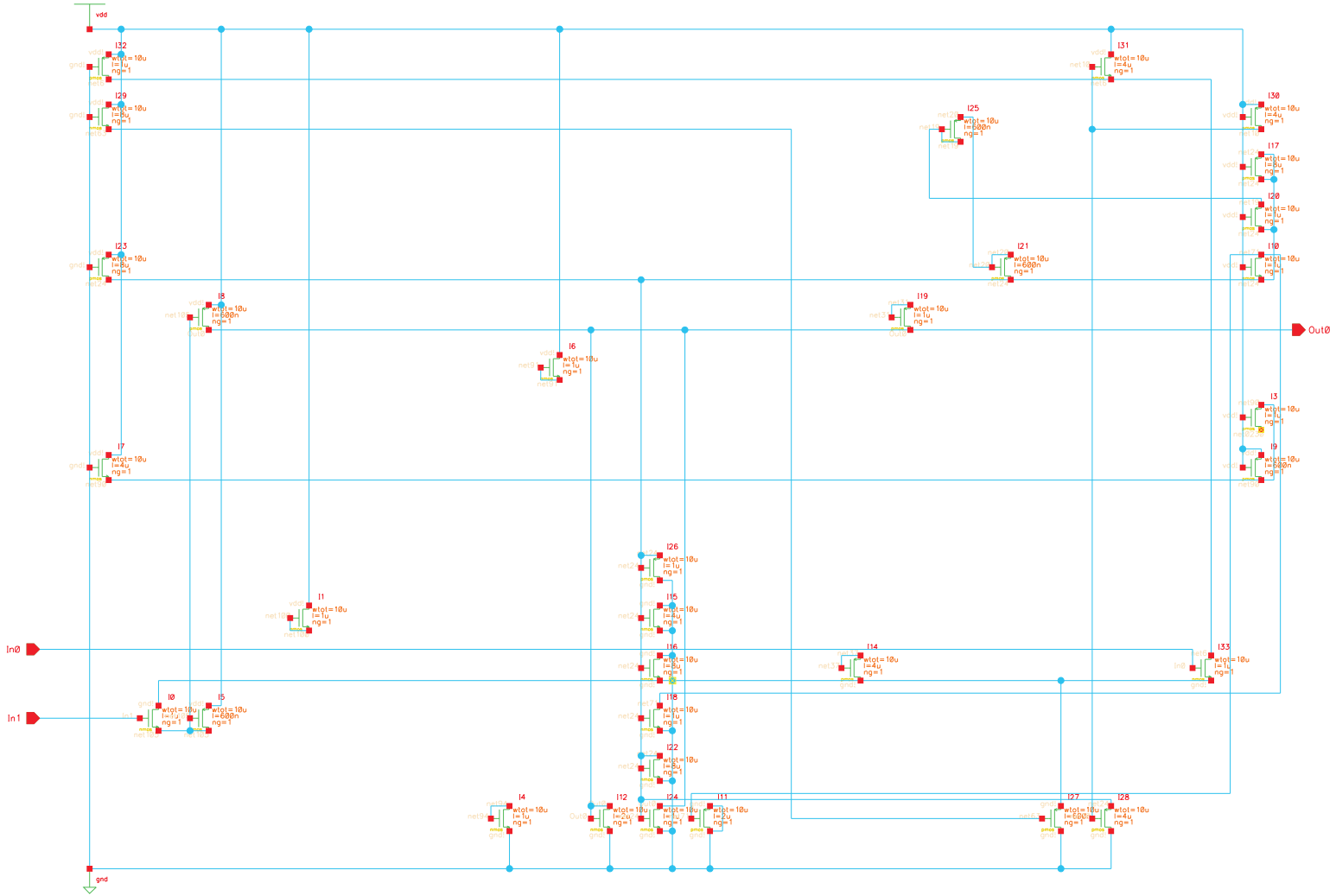


Figure B.6: *Schematic of an evolved OP with NMOS input transistors.*

```
.subckt EvolvedOpAmp-NMOS In0=22 In1=1 Out0=10 vdd=23
m0 2 1 3 0 modn l=1u w=12u
m1 4 4 23 0 modn l=1u w=0u
m2 5 5 6 0 modn l=0.6u w=3u
m3 7 23 7 23 modp l=1u w=4u
m4 0 8 8 0 modn l=1u w=9u
m5 2 2 23 23 modp l=0.6u w=2u
m6 9 9 23 0 modn l=1u w=0u
m7 7 0 23 0 modn l=4u w=2u
m8 10 2 23 23 modp l=0.6u w=15u
m9 7 23 23 23 modp l=0.6u w=12u
m10 11 23 12 0 modn l=1u w=0u
m11 0 12 0 23 modp l=2u w=2u
m12 0 10 10 0 modn l=2u w=0u
m13 14 13 14 0 modn l=1u w=5u
m14 3 15 15 23 modp l=4u w=2u
m15 0 11 3 0 modn l=4u w=6u
m16 0 11 0 0 modn l=8u w=3u
m17 11 23 11 23 modp l=8u w=0u
m18 0 11 12 0 modn l=1u w=0u
m19 10 16 16 23 modp l=1u w=11u
m20 11 23 17 23 modp l=1u w=10u
m21 11 18 18 23 modp l=0.6u w=4u
m22 0 11 11 0 modn l=8u w=13u
m23 11 0 23 23 modp l=8u w=1u
m24 0 11 10 0 modn l=1u w=4u
m25 17 17 18 23 modp l=0.6u w=8u
m26 0 11 11 23 modp l=1u w=0u
m27 0 19 3 23 modp l=0.6u w=3u
m28 0 20 11 23 modp l=4u w=5u
m29 19 0 23 0 modn l=8u w=0u
m30 20 23 23 23 modp l=4u w=4u
m31 21 20 23 0 modn l=4u w=0u
m32 21 0 23 23 modp l=1u w=3u
m33 3 22 21 0 modn l=1u w=12u
.ends EvolvedOpAmp-NMOS
```

Figure B.7: *SPICE netlist for the evolved operational amplifier with NMOS input, which is depicted in figure B.6.*

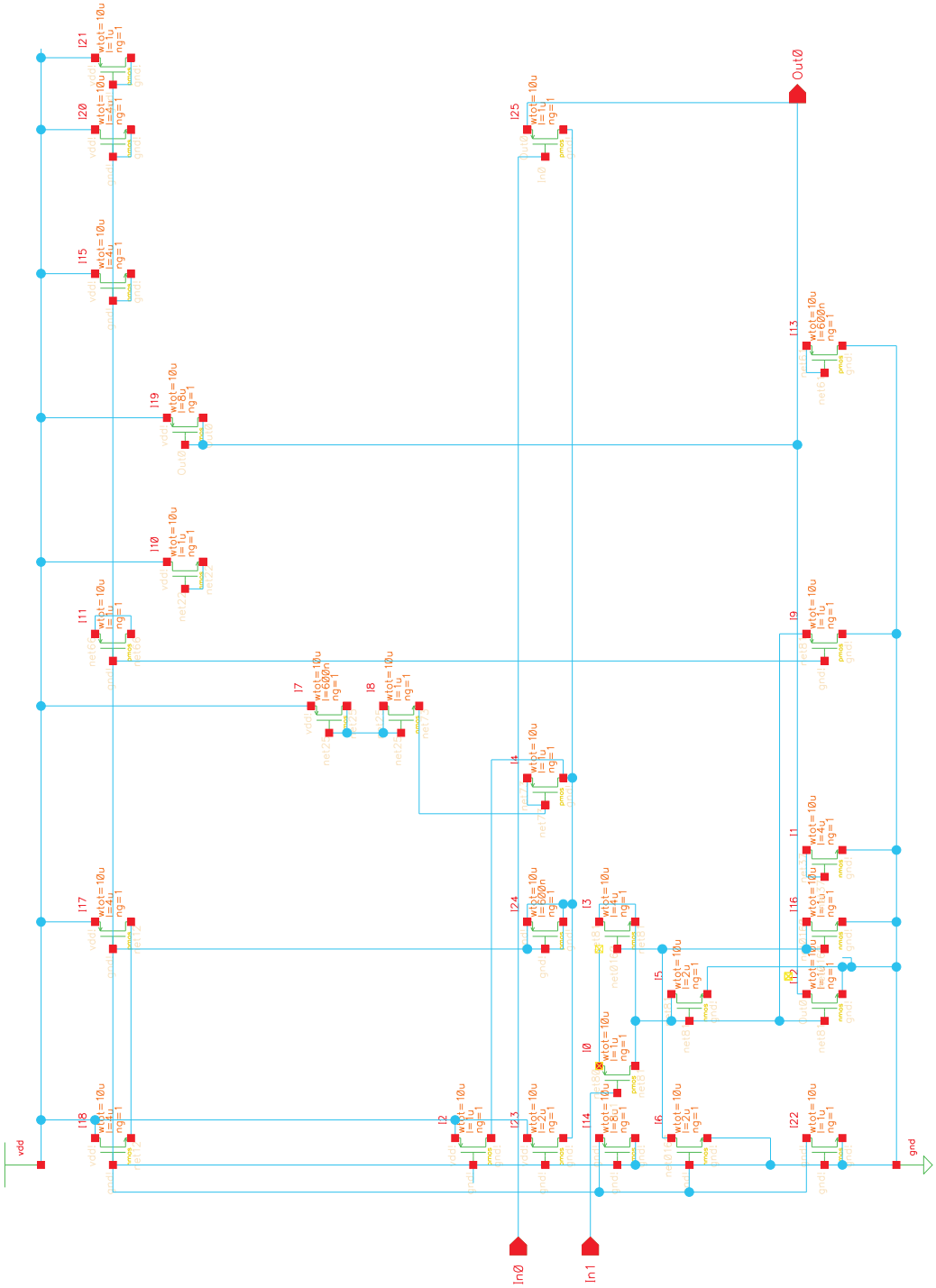


Figure B.8: *Schematic of an evolved OP with PMOS input transistors..*

```
.subckt EvolvedOpAmp-PMOS In0=16 In1=1 Out0=10 vdd=17
m0 2 1 3 17 modp l=1u w=12u
m1 0 4 4 0 modn l=4u w=8u
m2 3 0 17 17 modp l=1u w=1u
m3 2 3 2 0 modn l=4u w=1u
m4 3 5 5 17 modp l=1u w=0u
m5 0 2 2 0 modn l=2u w=13u
m6 0 0 3 0 modn l=2u w=0u
m7 6 6 17 17 modp l=0.6u w=3u
m8 5 6 6 0 modn l=1u w=7u
m9 0 7 2 17 modp l=1u w=2u
m10 8 8 17 0 modn l=1u w=6u
m11 9 7 9 17 modp l=1u w=13u
m12 0 2 10 0 modn l=1u w=5u
m13 0 11 11 17 modp l=0.6u w=11u
m14 0 0 12 17 modp l=8u w=10u
m15 12 12 17 0 modn l=4u w=12u
m16 0 3 3 0 modn l=1u w=0u
m17 13 3 17 17 modp l=4u w=15u
m18 13 0 17 0 modn l=4u w=8u
m19 10 10 17 17 modp l=8u w=0u
m20 14 14 17 0 modn l=4u w=11u
m21 15 15 17 17 modp l=1u w=4u
m22 0 0 15 0 modn l=1u w=4u
m23 3 0 17 17 modp l=2u w=5u
m24 3 3 3 17 modp l=0.6u w=8u
m25 3 16 10 17 modp l=1u w=12u
.ends EvolvedOpAmp-PMOS
```

Figure B.9: *SPICE netlist for the evolved operational amplifier with PMOS input, which is depicted in figure B.8.*

Appendix C

Tracking the Course of Evolution: a Side-Result

A tracking meachanism for the EA has been developed together with Stefan Zimmer during his internship with the intention to investigate the dynamics of evolution. To date, the tracking algorithm is proven to work and it is achieved to generate impressive family trees of the resulting individuals, as depicted in figure C.2. In order to achieve this, the tracking algorithm writes the mutation, crossover and selection operations of each evolutionary step to a file. This file is structured in a way that it is possible to automatically generate a family tree graph by using the free *graphviz* software package. As can be seen from the zoom view in figure C.1, various information of the evolution process is visualized. Hopefully, it will be possible to use this data for getting an insight into the optimization process of the evolutionary algorithm.

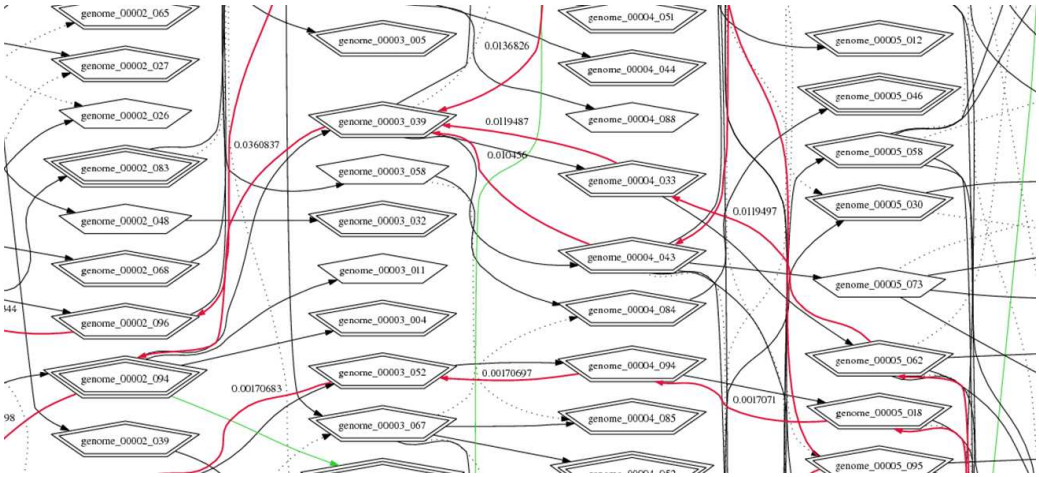


Figure C.1: *A zoom view of the family tree in figure C.2. The selection for mutation is marked with a black arrow and if crossover is carried out, the crossover partner is marked with a dotted black arrow. Elitist selection is illustrated with a green arrow. Furthermore, the ancestors of the best resulting individual can be tracked back along the red lines. The labels of the red lines show the fraction of the genetic information of the parent, that is still contributing to the resulting individual.*

A 250 Generations Family Tree

- graph drawn with free software from www.graphviz.org
- red-lines: backtracking a genome's ancestors
- hopefully reveal the secrets of the clan ...

Figure C.2: .

Appendix D

Algorithmic Take-Outs

D.1 Logic Crossover Operator

The *logic crossover* operator can be principally used with the *Turtle GA* and the *Basic GA*. As the name suggests, this operator calculates the logic combination of two selected individuals. The OR operator e.g. combines the features of both circuits, although if a transistor is present in both ciruits the W/L ratio will be taken from only one of them. Applied to highly diverse individuals, this results in a strong impact on the individuals structure. On the one hand, this usually changes the circuits output completely. On the other hand, since the *logic or crossover* does not destroy previous structures, it enriches the diversity of the individuals within the population and is therefore possibly helpful in avoiding local minima.

However, major problems are observed during testing the logic crossover operators: first, it will not be possible to calculate the logic OR of a FPTA cell configuration, if transistor terminals of the parent individuals are connected to different target nodes without adding additional routing in a post-processing step. Second, calculating the logic AND of two individuals results either in deserted circuits or in a great number of unconnected circuit islands. In addition to the latter effects, the performance of the algorithm did not increase when the logic OR/AND crossover were used. Therefore, (and for the lack of elegance of complicated post-processing steps) the logic crossover is not used for the experiments in this thesis.

D.2 Subpopulation of 'Mutants'

It is a complex task to avoid premature convergence of the population. First, it was tried to overcome this problem by treating a subpopulation of individuals with variation operators for which the mutation and crossover rates were significantly increased by a factor of 10. The individuals of this subpopulation were denoted as *mutants* and were intended to preserve a great diversity within the whole population by selecting crossover partners from the normal individuals and the *mutants*.

In later experiments, it has been observed that adapting the rates for the whole population according to its current best fitness features a better performance. The rates are scaled down with improving fitness and are thereby modeling a kind of simulated annealing. The equation, that is used to calculate the rates from the fitness values represents the cooling function.

Bibliography

[1] *SILVACO Spice Models*.

[2] *NGSPICE: Next Generation SPICE*, 2004.

[3] V. Aggarwal. Evolving Sinusoidal Oscillators Using Genetic Algorithms. In *5th NASA / DoD Workshop on Evolvable Hardware (EH 2003)*, pages 67–76, Chicago, IL, USA, 9-11 July 2003. IEEE Computer Society.

[4] P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design*, chapter 2, pages 36–55. Oxford University Press, Inc., 198 Madison Avenue, New York, NY, USA, 2 edition, 2002.

[5] P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design*, chapter 10, pages 617,622,654. Oxford University Press, Inc., 198 Madison Avenue, New York, NY, USA, 2 edition, 2002.

[6] P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design*. Oxford University Press, Inc., 198 Madison Avenue, New York, NY, USA, 2 edition, 2002.

[7] E. Altshuler and D. Linden. Design of a wire antenna using a genetic algorithm. *Journal of Electronic Defense*, 20(7):50–52, July 1997.

[8] Anadigm, Inc.˙ *AN121E04, AN221E04 Field Programmable Array – User Manual*, 2003.

[9] P. Antognetti and G. Massobrio. *Semiconductor Device Modelling with SPICE*. McGraw Hill, New York, 1987.

[10] T. T. Arpad Buermen, Janez Puhan. Robust Design and Optimization of Operating Amplifiers. pages 745–750, December 2003.

[11] T. Bäck. An Overview of Evolution Strategies. jun 2004.

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

[13] J. P. Becker. Ein FPGA-basiertes Testsystem für gemischt analog/digitale ASICs. Master's thesis, University of Heidelberg, 2001.

[14] A. Breidenassel. *A High Dynamic Range CMOS Image Sensor witht Adaptive Integration Time Control*. PhD thesis, University of Heidelberg, 2005.

[15] Cadence Design Systems, San Jose, CA, USA. *Cadence Online Library Openbook: SKILL Reference*, 5.0 edition, 10. SKILL scripting language.

[16] Cadence Design Systems, San Jose, CA, USA. *Cadence Online Library Openbook: spectreS reference manual*, 5.0 edition, 10. Description of the spectreS simulator.

[17] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

[18] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimisation: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 849–858, 2000.

[19] K. Deb and T. Goel. Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence. In E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 67–81. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[20] R. T. Edwards and C. J. Kim. Breaking the resistivity barrier. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *The Third NASA/DoD workshop on Evolvable Hardware*, pages 167–171, Long Beach, California, 12-14 July 2001. Jet Propulsion Laboratory, California Institute of Technology, IEEE Computer Society.

[21] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, Berlin, Heidelberg, New York, 2003.

[22] D. B. Fogel. *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York, 1995.

[23] D. B. Fogel. An Introduction to Simulated Evolutionary Optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, January 1994.

[24] D. P. Foty. *MOSFET modeling with SPICE: principles and practice*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[25] M. Garvie. *Reliable Electronics through Artificial Evolution*. PhD thesis, University of Sussex, 2004.

[26] G. G. E. Gielen and R. A. Rutenbar. Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits. *Proceedings of the IEEE*, 88(12):1825–1852, Dec 2000.

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

[28] T. G. W. Gordon and P. J. Bentley. On Evolvable Hardware. In S. Ovaska and L. Sztandera, editors, *Soft Computing in Industrial Electronics*. Physica-Verlag, Heidelberg, Germany, 2002.

[29] G. W. Greenwood, D. Hunter, and E. Ramsden. Fault Recovery in Linear Systems via Intrinsic Evolution. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 115–122, Seattle, WA, USA, June 2004. IEEE Press. ISBN: 0-7695-2145-2.

[30] S. Harding and J. F. Miller. Evolution in materio: A Tone Discriminator In Liquid Crystal. 2004.

[31] S. Harding and J. F. Miller. Evolution in materio: Initial experiments with liquid crystal. 2004.

[32] A. Hernandez Aguirre, R. S. Zebulum, and C. A. Coello Coello. Evolutionary Multiobjective Design targeting a Field Programmable Transistor Array. In Zebulum, Ricardo S., Gwaltney, David, Hornby, Gregory, Keymeulen, Didier Lohn, Jason, and Stoica, Adrian, editor, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 199–205, Los Alamitos, 2004. IEEE Computer Society Press.

[33] M. Hershenson, S. Boyd, and T. H. Lee. Optimal design of a CMOS op-amp via geometric programming. In *IEEE Transactions on Computer-Aided Design*, pages 1–21, 2001.

[34] T. Higuchi. 2nd international conference on the simulation of adaptive behavior. In *Proceedings of the 2nd International Conference on the Simulation of Adaptive Behavior*. MIT Press, 1992.

[35] S. Hohmann. *Stepwise Evolutionary Training Strategies for Hardware Neural Networks*. PhD thesis, University of Heidelberg, 2005.

[36] S. Hohmann, J. Schemmel, F. Schürmann, and K. Meier. Exploring the Parameter Space of a Genetic Algorithm for TRaining and Analog Neural Network. In K. Langdon et. al. editor, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 375–382, New York City, NY, USA, Jul 2002. Morgan Kaufmann Publishers. ISBN 1-55860-878-8.

[37] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. of Computing*, 2:88–105, 1973.

[38] Intel. Intel Itanium 2 Processor. http://www.intel.com/products/processor/itanium2/, 2006.

[39] ISO/IEC 14882. *Programming Language C++*, July 1998.

[40] N. Josuttis. *Die C++ Standardbibliothek, eine detaillierte Einfhrung in die vollstndige ANSI/ISO-Schnittstelle*. Addison-Wesley, 1 edition, 1996.

[41] Jungo Ltd., Netanya. *Windriver 6 User's Manual*, 2003.

[42] D. Keymeulen, G. Klimeck, R. S. Zebulum, A. Stoica, and C. Salazar-Lazaro. EHWPack: A Parallel Software/Hardware Environment for Evolvable Hardware. In W. Darrell, editor, *Proc. of the Genetics and Evolutionary Computation Conference (GECCO-2000)*, pages 538–539, Las Vegas, Nevada, USA, July 2000. Morgan Kaufmann.

[43] D. Keymeulen, R. S. Zebulum, V. Duong, X. Guo, I. Ferguson, and A. Stoica. High Temperature Experiments for Circuit Self-Recovery. In K. Deb et. al. editor, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part I*, pages 792–803, Seattle, WA, USA, Jun 2004. Springer-Verlag, LNCS 3102. ISBN 3-540-22344-4.

[44] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica. Fault-Tolerant Evolvable Hardware Using Field-Programmable Transistor Arrays. *IEEE Transactions on Reliability, Special Issue on Fault-Tolerant VLSI System*, 49(3):305–316, September 2000.

[45] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Design of a high-gain operational amplifier and other circuits by means of genetic programming. In In Angeline, Peter J., Reynolds, Robert G., McDonnell, John R., and Eberhart, Russ, editor, *Evolutionary Programming VI. 6th International Conference, EP97, Proceedings*, volume 1213 of *Lecture Notes in Computer Science*, pages 125–136, Indianapolis, Indiana, USA, 1997. Springer-Verlag, Berlin.

[46] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Evolution using genetic programming of a low-distortion 96 Decibel operational amplifier. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, pages 207–216, San Jose, California, USA, 1997. New York: Association for Computing Machinery.

[47] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, 1999.

[48] W. Kruiskamp and D. Leenaerts. DARWIN: CMOS opamp Synthesis by means of a Genetic Algorithm. In *DAC '95: Proceedings of the 32nd ACM/IEEE Conference on Design Automation*, pages 433–438, New York, NY, USA, 1995. ACM Press.

[49] J. Langeheine. *Intrinsic Hardware Evolution on the Transistor Level*. PhD thesis, Rupertus Carola University of Heidelberg, Seminarstrasse 2, 69120 Heidelberg, July 2005.

[50] J. Langeheine, J. Becker, S. Fölling, K. Meier, and J. Schemmel. A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits. In *Proc. of the Third NASA/DOD Workshop on Evolvable Hardware*, pages 172–175, Long Beach, CA, USA, July 2001. IEEE Computer Society Press.

[51] J. Langeheine, J. Becker, S. Fölling, K. Meier, and J. Schemmel. Initial Studies of a New VLSI Field Programmable Transistor Array. In Y. Liu, T. Kiyoshi, I. Masaya, T. Higuchi, and M. Yasunaga, editors, *Proc. 4th Int. C onf. on Evolvable Systems From Biology to Hardware (ICES2001)*, pages 62–73, Tokio, Japan, October 2001. Springer Verlag.

[52] J. Langeheine, S. Fölling, K. Meier, and J. Schemmel. Towards a silicon primordial soup: A fast approach to hardware evolution with a VLSI transistor array. In J. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, editors, *Proc. 3rd Int. Conf. on Evolvable Systems From Biology to Hardware (ICES2000)*, pages 123–132, Edinburgh, Scotland, UK, April 2001. Springer Verlag.

[53] J. Langeheine, K. Meier, and J. Schemmel. Intrinsic Evolution of Analog Electronic Circuits Using a CMOS FPTA Chip. In G. Bugeda, J.-A. Désidéri, J. Périaux, M. Schoenauer, and G. Winter, editors, *Proc. of the 5th Conf. on Evolutionary Methods for Design, Optimization and Control (EUROGEN 2003)*, pages 87–88, Barcelona, Spain, September 2003. IEEE Press. Published on CD: ISBN: 84-95999-33-1.

[54] J. Langeheine*, M. Trefzer*, D. Brüderle, K. Meier, and J. Schemmel. On the evolution of analog electronic circuits using building blocks on a CMOS FPTA. In K. Deb et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part I*, volume 1 of *LNCS 3102*, pages 1316–1327, Seattle, WA, USA, June 2004. Springer- Verlag.

[55] J. Langeheine, M. Trefzer, J. Schemmel, and K. Meier. Intrinsic Evolution of Digital-To-Analog Converters Using a CMOS FPTA Chip. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 18–25, Seattle, WA, USA, June 2004. IEEE Press. ISBN: 0-7695-2145-2.

[56] L. Logan. Data Acquisition: All about ENOB. Technical report, Data Translation, 1998.

[57] J. D. Lohn, G. Hornby, and D. S. Linden. Evolution, re-evolution, and prototype of an x-band antenna for nasa's space technology 5 mission. In J. M. Moreno, J. Madrenas, and J. Cosp, editors, *Evolvable Systems: From Biology to Hardware, 6th International Conference, ICES 2005, Proceedings*, volume 3637 of *Lecture Notes in Computer Science*, pages 205–214, Sitges, Spain, 2005. Springer.

[58] J. D. Lohn, D. S. Linden, G. Hornby, W. F. Kraus, and A. Rodriguez-Arroyo. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In *5th NASA / DoD Workshop on Evolvable Hardware (EH 2003)*, pages 155–163. IEEE Computer Society, July 2003.

[59] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programms*. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[60] J. F. Miller and K. Downing. Evolution in materio: Looking beyond the silicon box. In *Proc. of the Fourth NASA/DOD Workshop on Evolvable Hardware*, pages 167–176, Alexandria, VA, USA, July 2002. IEEE Press.

[61] G. E. Moore. Moore's law. http://www.intel.com/technology/mooreslaw/index.htm, 2006.

[62] J. M. Moreno Arostegui, E. Sanchez, and J. Cabestany. An in-system routing strategy for evolvable hardware programmable platforms. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *The Third NASA/DoD workshop on Evolvable Hardware*, pages 157–166, Long Beach, California, 12-14 July 2001. Jet Propulsion Laboratory, California Institute of Technology, IEEE Computer Society.

[63] S. Obayashi, D. Sasaki, Y. Takeguchi, and N. Hirose. Multiobjective evolutionary computation for supersonic wing-shape optimization. *IEEE Transactions on Evolutionary Computation*, 4(2):182–187, July 2000.

[64] PLX Technology, Inc., Sunnyvale. *PLX 9054 Data Book*, version 2.1 edition, January 2000.

[65] T. Quarles, A. Newton, D. Pederson, and A. Sangiovanni-Vincentelli. *SPICE3 Version 3f3 User s Manual*. Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Ca., 94720, May 1993.

[66] D. Sasaki, M. Morikawa, S. Obayashi, and K. Nakahashi. Aerodynamic shape optimization of supersonic wings by adaptive range multiobjective genetic algorithms. In K. Deb, L. Theile, C. Coello, D. Corne, and E. Zitler, editors, *In Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001, Proceedings*, volume 1993 of *Lecture Notes in Computer Science*, pages 639–652, Zurich, Switzerland, March 2001. Springer-Verlag.

[67] T. Schmitz, S. Hohmann, K. Meier, J. Schemmel, and F. Schürmann. Speeding up Hardware Evolution: A Coprocessor for Evolutionary Algorithms. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Proceedings of the 5th International Conference on Evolvable Systems ICES 2003*, pages 274–285. Springer Verlag, 2003.

[68] F. Schürmann, S. Hohmann, J. Schemmel, and K. Meier. Towards an Artificial Neural Network Framework. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 266–273. IEEE Computer Society, 2002.

[69] K. Sheehan and S. J. Flockton. Intrinsic Circuit Evolution Using Programmable Analogue Arrays. 1999.

[70] K. Sheehan and S. J. Flockton. Subsystem and Interconnection Strategies for Intrinsic Evolution of Analogue Hardware. 2000.

[71] H. Shibata. *Computer-Aided Design of Analog Circuits Based on Genetic Algorithm*. PhD thesis, Tokyo Institute of Technology, 2001.

[72] Z. Skolicki. An analysis of island models in evolutionary computation. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 386–389, New York, NY, USA, 2005. ACM Press.

[73] Z. Skolicki and K. A. De Jong. Improving evolutionary algorithms with multi-representation island models. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. M. Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*, volume 3242 of *Lecture Notes in Computer Science*, pages 420–429. Springer, 2004.

[74] T. Sripramong and C. Toumazou. The Invention of CMOS Amplifiers Using Genetic Programming and Current-Flow Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1237–1252, November 2002.

[75] A. Stoica. Reconfigurable Transistor Array for Evolvable Hardware. In *Caltech/JPL Novel Technology Report*, July 1996.

[76] A. Stoica. Evolutionary technique for automated synthesis of electronic circuits. In *Caltech/JPL Novel Technology Report*, July 1998.

[77] A. Stoica, D. Keymeulen, R. S. Zebulum, A. Thakoor, T. Daud, G. Klimeck, Y. Jin, R. Tawel, and V. Duong. Evolution of Analog Circuits on Field Programmable Transistor Arrays. In *Proc. of the Second NASA/DOD Workshop on Evolvable Hardware*, pages 99–108, Palo Alto, CA, USA, July 2000. IEEE Computer Society Press.

[78] A. Stoica, R. S. Zebulum, and D. Keymeulen. Polymorphic Electronics. In *Proc. ICES 2001, LNCS 2210*, pages 291–302, Tokio, Japan, October 2001. Springer Verlag.

[79] A. Stoica, R. S. Zebulum, D. Keymeulen, R. Tawel, T. Daud, and A. Thakoor. Reconfigurable VLSI Architectures for Evolvable Hardware: From Experimental Field Programmable Transistor Arrays to Evolution-Oriented Chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):227–232, February 2001.

[80] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, Reading, MA, August 1997.

[81] S. M. Sze. *Physics of Semiconductor Devices*. Wiley-Interscience, 2 edition, September 1981.

[82] S. M. Sze. *Semiconductor Devices: Physics and Technology*. Wiley, John and Sons, 2 edition, September 2001.

[83] A. Thompson. Evolutionary Techniques for Fault Tolerance. In *Proc. UKACC Int. Conf. on Control 1996 (CONTROL'96)*, pages 693–698. IEE Conference Publication No. 427, 1996.

[84] A. Thompson. Silicon Evolution. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proc. 1st Annual Conf. (GP96)*, pages 444–452, Stanford, CA, USA, Jul 1996. Cambridge, MA: MIT Press.

[85] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In T. Higuchi, M. Iwata, and L. Weixin, editors, *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, volume 1259 of *LNCS*, pages 390–405. Springer-Verlag, 1997.

[86] 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.

[87] A. Thompson and P. Layzell. Analysis of Unconventional Evolved Electronics. *Communications of the ACM*, 42(4):71–79, April 1999.

[88] A. Thompson and P. Layzell. Evolution of Robustness in an Electronics Design. In 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*, pages 218–228. Springer-Verlag, 2000.

[89] A. Thompson, P. Layzell, and R. S. Zebulum. Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution. *IEEE Trans. on Evolutionary Computation*, 3:167–196, September 1999.

[90] U. Tietze and C. Schenk. *Halbleiter-Schaltungstechnik*. Springer-Verlag, Berlin, 10 edition, 1991.

[91] M. Trefzer, J. Langeheine, J. Schemmel, and K. Meier. New Genetic Operators to Facilitate Understanding of Evolved Transistor Circuits. In R. S. Zebulum, D. Gwaltney, G. Hornby, D. Keymeulen, J. Lohn, and A. Stoica, editors, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 217–224. IEEE Computer Society Press, 2004.

[92] M. Trefzer, J. Langeheine, J. Schemmel, and K. Meier. Operational Amplifiers: An Example for Multi-Objective Optimization on an Analog Evolvable Hardware Platform. In J. M. Moreno, J. Madrenas, and J. Cosp, editors, *Evolvable Systems: From Biology to Hardware, Sixth International Conference, ICES 2005*, number 3637 in LNCS, pages 86–97, Sitges, Spain, September 2005. Springer-Verlag.

[93] M. Trefzer, J. Langeheine, J. Schemmel, and K. Meier. A Modular Framework for the Evolution of Circuits on Configurable Transistor Array Architectures. In *AHS 2006*, Istanbul, Turkey, June 2006.

[94] P. F. Vieira, L. B. Botelho, and A. Mesquita. Evolutionary Synthesis of Analog Circuits Using Only MOS Transistors. In Zebulum, Ricardo S., Gwaltney, David, Hornby, Gregory, Keymeulen, Didier Lohn, Jason, and Stoica, Adrian, editor, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 38–45, Los Alamitos, 2004. IEEE Computer Society Press.

[95] M. Wall. *C++ Genetic Algorithm Library, GALib 2.4.6*. Massachusetts Institute of Technology, MIT, 1999.

[96] K. Weicker. *Evolutionäre Algorithmen*. Leitfaden der Informatik. B. G. Teubner, Stuttgart, Germany, 2002. ISBN: 3-519-00362-7.

[97] D. Whitley, S. B. Rana, and R. B. Heckendorn. Island model genetic algorithms and linearly separable problems. In *Evolutionary Computing, AISB Workshop*, pages 109–125, 1997.

[98] Wikipedia. Transistor. http://en.wikipedia.org/wiki/Transistor, 2006.

[99] E. Williams, W. Crossley, and T. Lang. Average and maximum revisit time trade studies for satellite constellations using a multiobjective genetic algorithm. *Journal of the Astronautical Sciences*, 49(3):385–400, July-September 2001.

[100] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[101] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124-3400, USA. *Virtex E Datasheet*.

[102] Xilinx Inc., San Jose. *Xilinx XC9536XL High Performance CPLD*, September 2004.

[103] R. S. Zebulum, M. A. Pacheco, and M. Vellasco. A multi-objective optimisation methodology applied to the synthesis of low-power operational amplifiers. In I. J. Cheuri and C. A. dos Reis Filho, editors, *Proceedings of the XIII International Conference in Microelectronics and Packaging*, volume 1, pages 264–271, Curitiba, Brazil, 1998.

[104] R. S. Zebulum, D. Keymeulen, V. Duong, G. Xin, M. Ferguson, and A. Stoica. Experimental Results in Evolutionary Fault-Recovery for Field Programmable Analog Devices. In J. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, and M. I. Stoica, Adrian an Ferguson, editors, *Proc. of the Fifth NASA/DOD Workshop on Evolvable Hardware*, pages 182–186, Chicago, IL, USA, July 2003. IEEE Press. ISBN 0-7695-1977-6.

[105] R. S. Zebulum, M. A. P. Pacheco, and M. Vellasco. Artificial Evolution of Active Filters: A Case Study. In *Proc. of the First NASA/DOD Workshop on Evolvable Hardware*, pages 66–75, Pasadena, CA, USA, July 1999. IEEE Press.

[106] R. S. Zebulum, A. Stoica, and D. Keymeulen. Design process of an evolutionary oriented reconfigurable architecture. In *Proc. of the Congress on Evolutionary Computation*, pages 529–536, San Diego, CA, USA, July 2000. IEEE Press.

[107] R. S. Zebulum, A. Stoica, and D. Keymeulen. A Flexible Model of a CMOS Field Programmable Transistor Array Targeted for Hardware Evolution. In J. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, editors, *Proc. of the Third Int. Conference on Evolvable Systems: From Biology to Hardware (ICES2000), LNCS 1801*, pages 274–283, Edinburgh, UK, April 2000. Springer. ISBN 3-540-67338-5 LNCS 1801.

[108] R. Zebulum, M. Pacheco, and M. Vellasco. Synthesis of CMOS Operational Amplifiers Through Genetic Algorithms. In *Proceedings of the SBCCI98 (Brazilian Symposium on Integrated Circuits)*, pages 125–128, Rio de Janeiro, Brazil, September 1998.

[109] R. Zebulum, M. Pacheco, and M. Vellasco. A Novel Multi-Objective Optimization Methodology Applied to the Synthesis of CMOS Operational Amplifiers. *Journal of Solid-State Devices and Circuits*, pages 10–15, February 2000.

Danksagung (Acknowledgements)

Zum Abschluß möchte ich mich bei all jenen bedanken, die mit ihrer fachlichen und persönlichen Unterstützung zum Gelingen dieser Arbeit beigetragen haben. An dieser Stelle ganz herzlichen Dank an

- Herrn Prof. Dr. Karlheinz Meier für die freundliche Aufnahme in seine Arbeitsgruppe und die Möglichkeit, mit viel Freiheit in einem so interessanten und unkonventionellen Gebiet forschen zu können.

- Herrn Prof. Dr. Fred A. Hamprecht, der freundlicherweise das Zweitgutachten übernommen hat.

- Dr. Johannes Schemmel, der mit der Idee zum FPTA Projekt den Grundstein für alles Weitere gelegt hat und von dem ich fachlich und organisatorisch viel gelernt habe. Einen solchen Post-Doc, der praktisch zu jedem Thema Wissen in petto und zusätzlich eine offene und freundliche Art hat, kann man jedem nur wünschen.

- Jörg Langeheine, meinen 'Evolutionäre-Elektronik-Sensei', der nicht nur den FPTA entwickelt, sondern mir auch geduldig beigebracht hat wie man ihn benutzt. Herzlichen Dank für die angenehme und bereichernde Zusammenarbeit und die selbe Art von Humor! Das Büro ward öd und leer ohne dich und deine Familienpackung Nivea Creme.

- Daniel Brüderle, der wärend meines ersten Jahres viel mitprogrammiert hat und den wir dann leider an ein Neuronales Netz verloren haben. Vielen Dank auch an Stefan Zimmer, meinen Informatik-Praktikanten der leider viel zu kurz da war, für die Implementierung des 'Tracking GA'.

- Thorsten Maucher und Andreas Grübl für viele fachliche und ebenso viele nichtfachliche Gespräche, sowie zahlreiche gemeinsame Aktionen ausserhalb der Arbeit.

- Johannes Fieres und Steffen Hohmann, von deren Unmengen an tollem C++-Code ich sehr viel lernen konnte und weil es einfach sympathische Burschen sind.

- Stefan Phillip und Felix Schürmann für die jahrelange kollegiale Zusammenarbeit und die Betreuung unserer Server.

- Andreas Grübl, Margarita Kallweit, Jörg Langeheine, Stefan Phillip und Tillmann Schmitz für das Korrekturlesen der Arbeit.

- allen Mitgliedern der Electronic Vision(s) Gruppe für die angenehme und harmonische Arbeitsatmosphäre, die vielen erheiternden ordentlichen und ausserordendlichen Zusammentreffen und die Bereitschaft, Feste zu feiern wie sie fallen.

- allen Mitarbeitern des Kirchhoff-Instituts, für die freundliche Atmosphäre, dafür dass sie den ganzen Laden am Laufen halten und auch mal fünfe gerade sein lassen.

Ich möchte mich auch bei all den Menschen bedanken, die nicht direkt Teil meiner Arbeitswelt sind, deren freundschaftlicher und emotionaler Beistand jedoch nicht minder wichtig für das Gelingen dieser Arbeit und vieler anderer Dinge in meinem Leben sind. Herzlichen Dank daher auch an:

- vor allem meinen Eltern und Grosseltern, die die Grundvorraussetzung für diese Arbeit schufen, nämlich mich, von denen ich viel fürs Leben gelernt habe und die mich während meiner Ausbildung stets in vielerlei Hinsicht unterstützt haben.

- den Mensaclub: Gerd, Hans, Jan, Tobias, Torsten und gelegentliche Gastschauspieler, die mit hitzigen Diskussionen beim Nachmittagskaffee den sozialen Ausgleich zum täglichen coden schaffen.

- alle Freunde, vor allem Hendrik, Jean-Marc und Tillmann, die mein Leben mit ihren verschiedenen Persönlichkeiten bereichern.

- meine allerliebste Gordana, die es nun schon viele Jahre mit mir aushält, mein naturwissenschaftliches Wissen um das Geisteswissenschaftliche erweitert und mich oft mit ihrer erfrischend anderen Sicht der Dinge wieder erdet.