INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Fuzzy Genetic Optimization Approach Applied On IC Analog Synthesis

Carla Sofia Paiva Duarte

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente:	Professor Marcelino Bicho Dos Santos
Orientador:	Professor Nuno Cavaco Gomes Horta
Co-Orientador:	Professor Jorge Manuel Correia Guilherme
Vogal:	Professor Rui Manuel Leitão Santos Tavares

April 2011

This page was intentionally left blank

ABSTRACT

The presented thesis is about a system for analog integrated circuit (IC) design automation at synthesis level. FUGA an innovative circuit-level optimization kernel was developed using C language, and python scripting to support the user interface interaction, and also the electric circuit simulator communication.

The proposed approach, first models analog design knowledge using soft computing techniques, than enhances a stochastic optimization kernel by embedding the design knowledge model. This approach uses common IC design environments (i.e. cadence), and also run as a standalone system with its user interface. The electric circuit simulator of choice is spectre and the system is validated for well known design circuit examples.

The proposed system implements a new design automation strategy based on a genetic algorithm kernel combined with fuzzy logic system. The fuzzy logic system introduces design knowledge into the GA kernel providing an effective guidance, increasing the quality of the results and in principal minimizing the number of generations necessary to reach the desired solution.

Key Words:

Analog Integrated Circuits Synthesis

Computer Aided Design

Circuit Synthesis

Electronic Design Automation

Evolutionary Algorithms

Fuzzy Systems

Genetic Algorithms

This page was intentionally left blank

RESUMO

Este trabalho tem como objectivo o estudo e implementação de um sistema de automação de projecto de circuitos integrados (CI) a nível de síntese. Foi desenvolvido um kernel de optimização inovador denominado FUGA. A linguagem C foi utilizada no desenvolvimento do kernel de optimização, e python foi utilizado no suporte da interface com o utilizador, e também na interligação com o simulador eléctrico.

A abordagem proposta centra-se no desenvolvimento de uma nova estratégia que primeiro modela o conhecimento do projecto analógico e depois usa esse modelo para guiar e melhorar a performance do kernel de optimização. O ambiente de trabalho utilizado é o comum no projecto de CI, e utiliza-se para suporte ferramentas comercias como o Cadence. O simulador eléctrico escolhido foi o spectre, e alguns exemplos bem conhecidos foram desenvolvidos para validar o sistema.

A abordagem proposta implementa uma nova técnica para automação de projecto baseado em algoritmos genéticos (AG) combinado com sistemas difusos. O modelo do sistema difuso introduz um nível de conhecimento no kernel AG, que guia o processo de optimização, melhorando a qualidade dos resultados e principalmente diminuindo o número de gerações necessárias para se encontrar a solução desejada.

Palavras-chave:

Síntese de Circuitos Integrados Analógicos

Projecto Assistido por Computador

Dimensionamento de Circuitos

Automação de Projecto Electrónico

Algoritmos Evolutivos

Sistemas Difusos

Algoritmos Genéticos

This page was intentionally left blank

ACKNOWLEDGEMENTS

I would like to thank Prof. Nuno Horta for his enlightenments, and helpful discussions that allow me to strength my knowledge in technological prospective and also the guidance that lead me to develop skills for research attitude. Also thanks to Prof. Jorge Guilherme for his support. All I have learned I believe is a key tool for my professional and life success.

I am pleased to thank Dr Manuel Barros for his time, valuable explanations and for his excellent research contribution in IC Design Automation.

Finally, I want to thank everybody that in some way contributed for the conclusion of this work.

This page was intentionally left blank

TABLE OF CONTENTS

LIST OF FIGURE

List of Tables

LIST OF ABBREVIATIONS

AMS	Analog and mixed-signal
CAD	Computer Aided Design
CMOS	Complementary Metal Oxide Semiconductor
EDA	Electronic Design Automation
IC	Integrated Circuit
GA	Genetic Algorithms
SA	Simulated Annealing
SOA	State Of the Art
SoC	Systems-On-a-Chip
PSO	Particle Swarm Optimization
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference

1 INTRODUCTION

The subject of study in this thesis is Analog and Integrated Circuit (IC) design automation, at synthesis level. An application of a promising optimization technique was developed, in the field of evolutionary computation, combined with fuzzy logic systems as a model engine to improve the developed optimization kernel performance. A detailed comparison between the results achieved, with the implemented tool and other state of the art tools are described, to provide a proof for the applicability of the concepts discussed, and to show the relevance of the produced labour.

This chapter will present the relevant motivation for this study, a brief overview over the subject, the research goals, the contribution given with the develop work, and it ends with the report outline.

1.1 Motivation

Microelectronics market's increasing complexity (number of transistors on a chip) and integration, often expressed as complexity doubles every 2.2 years (according with Moore's Law) is due to a fast growing use of electronic devices for telecommunication such as telephone, television, radio or computer.

These devises have strong performance demands as less power consumption, several features (WiFi connectivity, Internet e-mail, infrared, Bluetooth, USB, etc) combined in a single product as evidenced by systems-on-a-chip (SoC) designs illustrated in Figure 1-1.

SoC merge Analog, digital, and RF sections on a single chip, and the shrinking process technologies to deep sub micrometer levels, aggravated with the market pressures to shorten product life cycles, increase the designs challenges, leading to a major demand for computer aided design (CAD) tools.

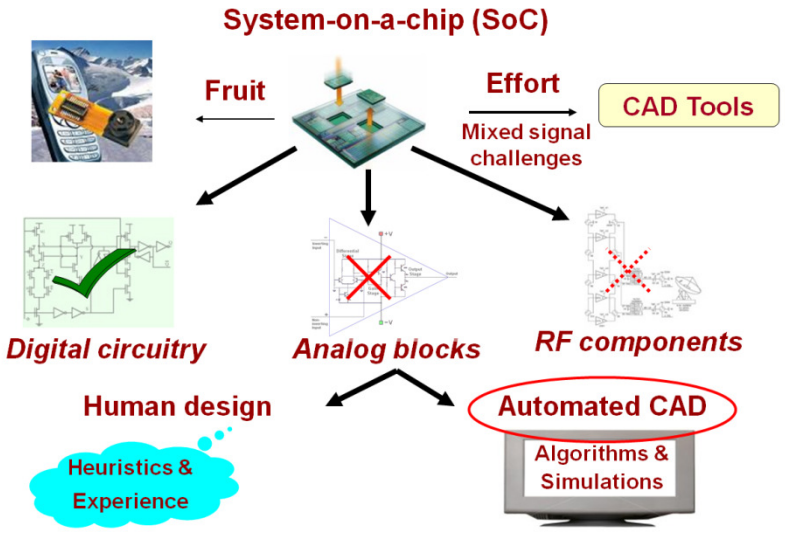


Figure 1-1 System-on-a-chip complexity

Some CAD tools solutions to these challenges are ready for commercialization, while others are still in early stages of exploration and development. These challenges require intense interaction between circuit designers, process development, and system engineering, to satisfy the demands of innovative (IC) technologies applications. Despite some significant research efforts, a full deployment of Analog CAD tools into professional environments is still very limited. Even with the help of highly specialized tools that cover some of the steps of the Analog design flow, the essential act of designing at transistor level is still performed by the trial and error

interaction between the designer and the simulator. The traditional design process approach characterized by a combination of individual experience and intuition still dominates the Analog design proceedings.

The electronics industry is increasingly focused on the consumer marketplace, which requires low-cost high-volume products to be developed very rapidly. That's why a strong need of software tools (CAD) for Analog and Mixed-Signal Integrated Circuits Design to reach the goal of increase the productivity of Analog designers (shorten the time to market) and to improve the quality and optimality of Analog and mixed-signal IC designs (e.g. power reduction, but also avoiding expensive redesign runs).

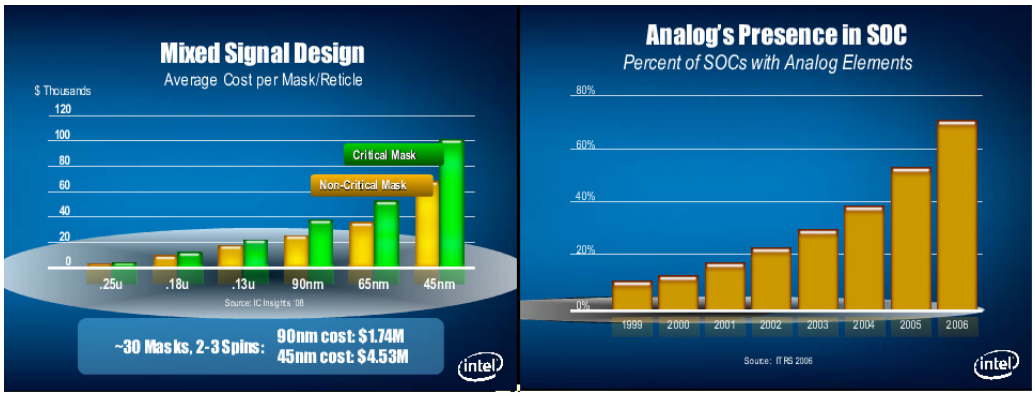


Figure 1-2 Mixed Signal Design Cost & Analog presence in SOC [Intel]

Figure 1-2 illustrates an analysis of the cost versus shrinking of technology and also shows the percentage of SoC designs that contain Analog circuits, the evolution over the year for Analog in SoC evidencing the complexity involved and the need of CAD to manage the entire design process and design complexity, avoiding tedium of manually designing routine and repetitive design task, reducing costs, and time-to-market. Reusability of ICs typical circuits blocks and save more time to focus on the creative aspects of design.

As important as digital circuit, Analog circuit remains an integral piece on mixed-signal ICs and emerging systems-on-a-chip (SoC), based on the fact that the digital systems have to communicate with the continuous-valued external world as shown in Figure 1-3.

Figure 1-3 It is an Analog world [Intel]

Although the Analog blocks typically constitute only a small fraction of the components on mixed-signal ICs and SoC, design effort spent on Analog design process is much more, compared with digital automated reality, where digital CAD tools are well developed and widely available. This is one of the main reasons for the growing need for "CAD" tools that increase the design productivity and improve the quality of Analog integrated circuits.

1.2 IC Design Automation and Technological Advances

Tremendous progress in microelectronics has pushed the MOSFET dimension toward the 10-nm limits and motivated the interest for new devices. Figure 1-4 illustrates this trend. In the near future it is then probable that CMOS will need to share its domination with fundamentally new devices, which are already in research and prototyping stage with promising results.

The use of advanced CMOS nanometer technologies however also brings along significant challenges for circuit design (both Analog and digital) such as:

- The advent of no longer negligible leakage currents and the impact of power on digital designs;
- The increase in variability of technological parameters;
- Dropping supply voltages, that shrink the headroom for Analog circuit designs;
- The use of novel devices like in FinFets, and novel materials like high-k dielectrics;
- The surge of novel degradation mechanisms and the increasing reliability such as EMC/EMI regulation.[2]

To address these problems, new design methodologies and tools are needed. Analog design automation tools are not developing at the same pace of technology, once custom design, characterized by decisions taken at each step of the Analog design flow, relies most of the time on designer knowledge and expertise. In addition, integrating Analog and digital circuits on the same die creates additional problems of crosstalk and signal integrity, which require tool support for modelling and analysis. Pushed by the progress in nanometre technology, the design teams are facing a curve of complexity that grows exponentially, thereby slowing down the productivity design rate [3].

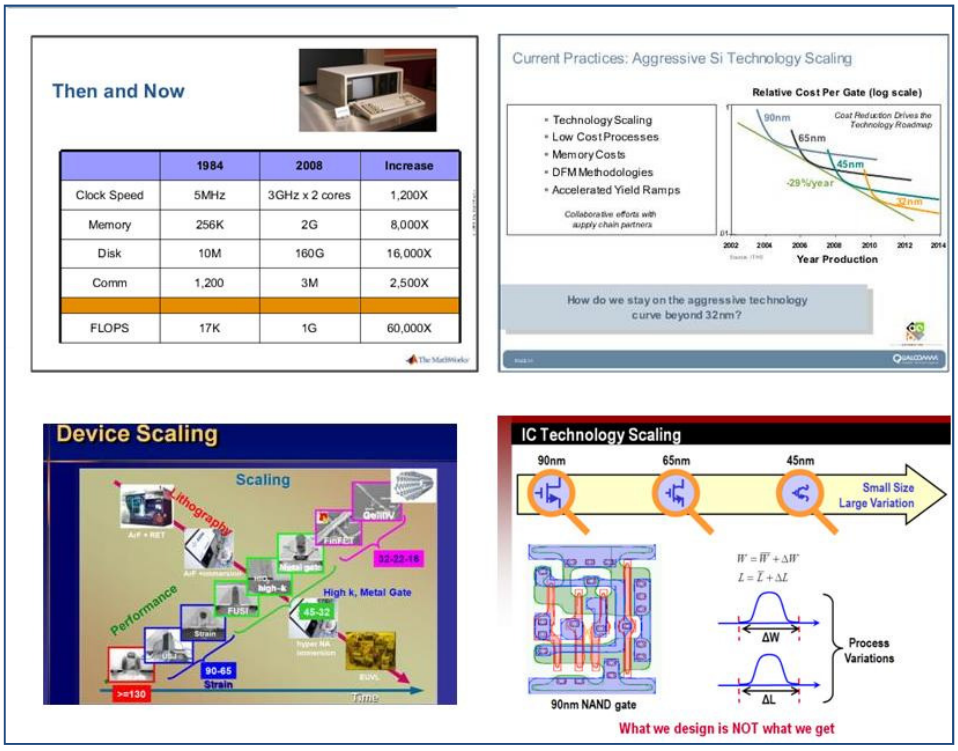


Figure 1-4 Technology advance, a close analysis.

1.3 Research Goals

The main goal of this thesis is to build a synthesis system (software) for Analog and IC design automation. This proposed approach aims to present a functional system that can be used in practice, as a research prototype. Therefore considering the most important design criteria for Analog design: speed (operating frequency, bandwidth …) and accuracy (noise, distortion, offset …) in combination with power consumption (cost) these fundamental specification form a bounded design space determined by technological constraints as illustrated in Figure 1-5.

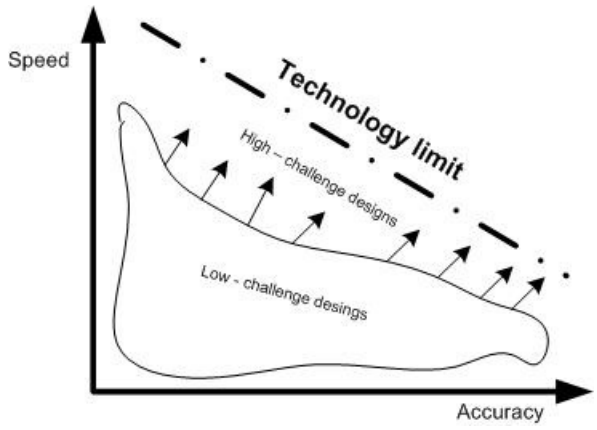


Figure 1-5 Design space: Low challenge and high-challenge designs and technology limit as a function of speed and accuracy [1]

All known circuit topologies together form this boundary, new circuit schematics and design techniques push the available application area toward this technology determined limit. Figure 1-6 shows the current evolution towards ultra-deep-submicron and nanometer technologies. However design complexity increases with the technology advance, still many designs do not challenge the design capabilities of the designer fully. That's why the major trends in design

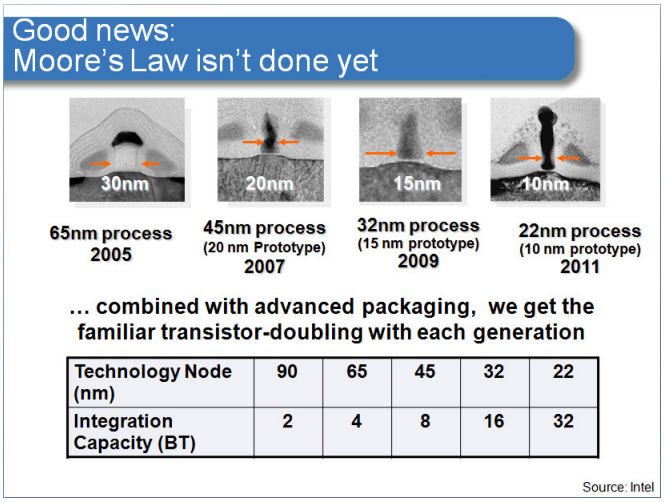


Figure 1-6 Good news: Moore's Law isn't done yet [Intel]

automation can be recognized as:

- Automated or semi–automated synthesis: the low challenge Analog design problems, for which known topologies and design procedures suffice and the design problem are reduced to applying them. Automation is feasible and even appreciated by Analog designers: Analog design experts prefer not to spend time on these repetitive designs.

- High –challenge designs: high – speed or high-accuracy have to be combined or new ideas, architectures and design techniques /methodologies are necessary to push the design closer to technology limits. This interesting, creative work, that is by definition not possible with automation, but can be supported by CAD tools [1].

CAD Tools can help push the limit. Although competitive edge is usually derived from time-to-market, more importantly it is derived from time-to-yield advantages. As geometries shrink, design challenges require innovative solutions to deliver maximum profit. However, tuning full custom nanometre designs to maximise yield and profit is an iterative and largely manual design effort. A non-automated methodology can no longer be considered competitive.

In this work a genetic algorithm optimization based kernel was developed in continuous interaction with a known circuit simulator and model engine strategies (i.e. fuzzy system, design of experiment techniques) to help to address the exposed challenges. The main objective is to be a solution to automate the synthesis of Analog integrated circuit, which can increase the design efficiency. Detailed descriptions for the research and development goal are given here:

- Develop a synthesis system (software tool) to assist the design flow at circuit level, to increase the design productivity and to free up valuable Analog design expert for more challenging designs. The system should be able to handle sufficiently complex circuit to be useful in practice.

- Research and implement new design methodologies, design flows, algorithms, models and techniques to improve Analog synthesis results or speed up Analog synthesis. Use the system to explore alternatives, benchmarks, and demonstrate a functional system, able to solve real cases, Analog circuit of reasonable complexity. The tool should have a user friendly interface that can be run by different users, not only by the creator.

- The system should have independent modules that interact with different features internal to optimization kernel and also external, like the circuit simulator, and model engines strategies, user interface, or other commercial tool which make it possible to replace individual modules, without developing again the all system.

- Verify the effectiveness of the implement solution by doing experiments and comparisons of the obtained results with available SOA tools results and with the internal Cadence optimizer tool. This comparisons made over previous approaches should reflect the advances and significant improvement in results. The significance of the results obtained – by measurable quantitative criteria (runtime for tools, optimality of results, time for design process steps, automation of manual effort, etc.). Finally a good discussion of the limitations of the approach and concepts, and possible areas for future improvement.

1.4 Related Publications

The work developed in this thesis is registered in some international publications, were the research work and the produced tool was presented, and shows the relevance of the study done in this thesis.

There were two publications resulted from a partial contribution in a MatLab prototype developed in collaboration with the student Pedro Sousa, on Integrated Circuits and Systems Group (ICSG), under Professor Nuno Horta coordination.

The first submission entitled “Enhancing a GA Optimization Kernel based on Fuzzy Rules” on Xth Int. Workshop Symbolic & Numerical Methods, Modelling and Application to Circuit Design (SM2ACD), Erfurt, Germany, in October 2008.

The second submission entitled “FUGA: A Fuzzy-Genetic Analog Circuit Optimization Kernel” on (GECCO) Genetic and Evolutionary Computation Conference 2009.

This developed prototype shows the promising application of fuzzy system in combination with genetic algorithms and equation based evaluation engine, applied in simple circuit examples, in a MatLab environment.

“Enhancing Analog IC Design Optimization Kernels with Simple Fuzzy Models.” on ECCTD’09, European Conference on Circuit Theory and Design 2009, were the Synthesis System tool based on genetic algorithm and a fuzzy system model engine with electrical circuit simulator was described and the achieved results discussed.

“Optimal OpAmp Sizing based on a Fuzzy-Genetic Kernel” has been accepted for a Workshop at GECCO-2011.

1.5 Dissertation Outline

Chapter 1 presents the motivation for this work, research goals and the main contribution.

Chapter 2 provides an overview for the state of the art on design automation of Analog and Mixed-Signal IC. A comparative analysis between different approaches, following several metrics developed over the year, and the contribution each tool and methodologies brought to the CAD world. Finally a description of the developed solution is introduced, and the advantage it brings to design automation.

In **Chapter 3** a general summary of the fuzzy system concepts and the different existent techniques, to combine and enhance genetic algorithms are demonstrated.

Chapter 4, this chapter present a detailed top down description of the implemented tool, the decisions made to choose among many algorithms, model, and techniques to adopt in every internal and external module.

In **Chapter 5** experimental results will be provided, that demonstrates the functionalities and accuracy of the tool.

In **Chapter 6** general conclusions will be formulated along with the research contribution this work achieve and suggestions for future work improvements will be proposed.

2 STATE OF THE ART: AN OVERVIEW ON DESIGN AUTOMATION

This chapter presents the “state of the art” (SOA) for Analog and mixed-signal integrated circuits design automation. Figure 2-1 illustrates a possible complete design flow for Analog/mixed-signal systems, from an abstract concept to the final product.

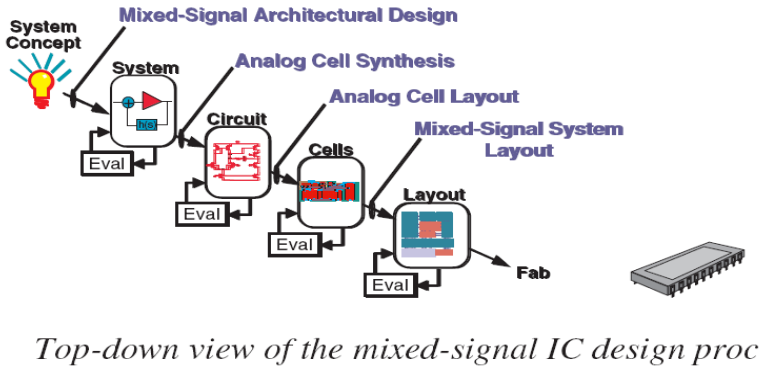


Figure 2-1 Complete design flow.

2.1 Design Automation General View

A typical design flow for Analog and mixed-signal (AMS) IC consists of a number of design steps repeated top-down, from the system hierarchical level, and bottom-up for composition and verification. The steps between any two of these hierarchical levels are: architecture/topology selection, circuit sizing and layout generation, and design verification task, illustrated in Figure 2-2. In order to handle the increasing complexity of Analog and mixed-signal ICs design, the definition of a hierarchical design flow is essential. Despite the advances during the last decades the design automation (DA) tools in Analog domain cannot support the complete design process, since they either concentrate on specific parts of design flow or require the intervention of an expert designer. Moreover, they mainly address circuit level design as a whole, which makes it difficult to generalize to high complex circuits and systems. Therefore, as the SoC complexity increases, the solution seems obvious, i.e., the design automation tools must incorporate an hierarchical design decomposition feature in order to apply the well known divide-to-conquer strategy already applied by most Analog designers in a manual approach.

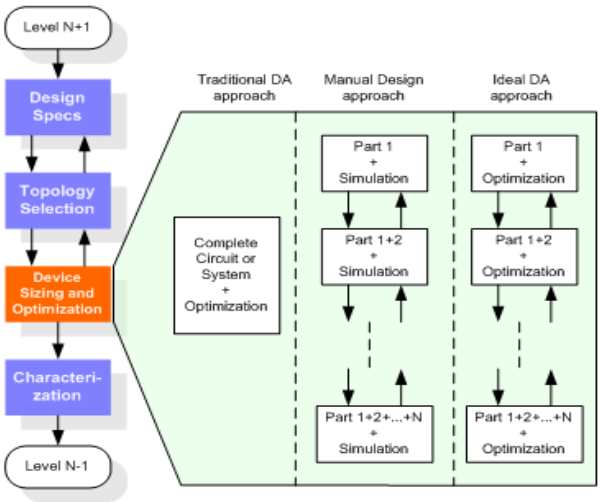


Figure 2-2 Hierarchical level and design tasks of design flow architectures [4]

2.2 Design Automation Tools and Methodologies

An overview of optimization techniques and modelling methodology progress, used on design automation tools for different design steps, and in more detail for synthesis and optimization designs solutions are described below.

Trends in this area have been running towards a class of design automation methodology under three aspects: improving flexibility, allowing the designer to have a higher interaction during the synthesis process and providing a more general approach to deal with multiple architectures or circuit types; modularity, permitting the use of different tools and techniques to address different design tasks, such as topology selection, circuit sizing and layout; and finally hierarchy, allowing the handling of complex system designs and implementing strategies involving several abstraction levels.

The first step in the process is finding a topology (or shematic, netlist), i.e. an interconnection of lower level instances that is capable of realizing the wanted behaviour. The selection of an adequate architecture is the main condition to achieve a good circuit performance design [Gielen-Integration, the VLSI Journal]. The sizing task receives a topology description, the performance specs and produces the sizing solution for each block or component dependent on the abstraction level. Several solutions were proposed derived from either knowledge-based methods or optimization-based approaches.

The layout generation process follows a bottom-up path once the sizing task has been accomplished at the lowest abstraction level. The physical layout geometry has a strong impact on circuit performance. Device matching, parasitics, thermal and substrate effects must all be taken into account to design high performance analog circuits [30]. If not properly estimated and controlled, these layout effects will result in over designed circuits, wasting power and area and, in same cases, to mal-function circuits that do not meet the required specifications. To overcome these problems some schemes have arisen to automate the layout design. An overview of Layout tools is shown in Appendix E.

2.2.1 Knowledge Based Approach versus Optimization Based Approach

The two principal approaches are the knowledge based approach, which employs a predefined sizing plan, and the optimization-based approach, which is based on different optimization methods. The first approach doesn't get an industrial acceptance since for every new circuit topology and technology, a new set of rules and equations had to be created by the hand of an expert designer. Lately, a more potential alternative is obtained considering the circuit sizing as an optimization problem. In these approaches the design problem is first mapped or modeled into an optimization problem and then solved by an appropriate optimization method, as illustrated in Figure 2-3.

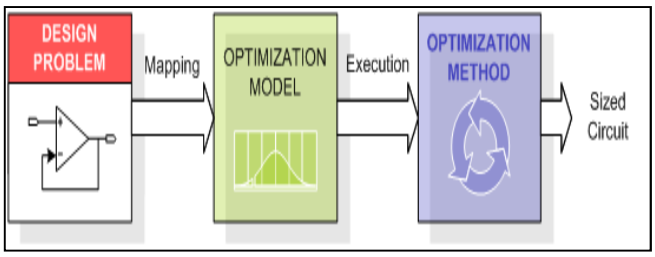


Figure 2-3 Circuit Optimization Process

There may be many variations to set up of the optimization problem, for instance, the optimization problem may have either multiple objectives and multiple constraints or one objective and multiple constraints or a series of optimization problems with one objective and multiple constraints. Since these steps are not independent and have influence on each other, the optimization method will be decided by the chosen model of the problem. However, due to the great variety of optimization methods and models and the inexistence of specific

recommendations on how to choose and apply a definite algorithm to analog circuit optimization, several alternatives have been emerging.

The computer-aided design methodology for AMS circuits in short-run foresees the integration of design automation tools to automate the several steps of the design methodology. This trend began in 80's when the first automation tools applied to different tasks of analog design appeared [5] and [6]. Several solutions were proposed derived from either knowledge-based methods, using some kind of knowledge and heuristics, or optimization-based approaches for both topology selection and specification translation, circuit sizing or layout generation.

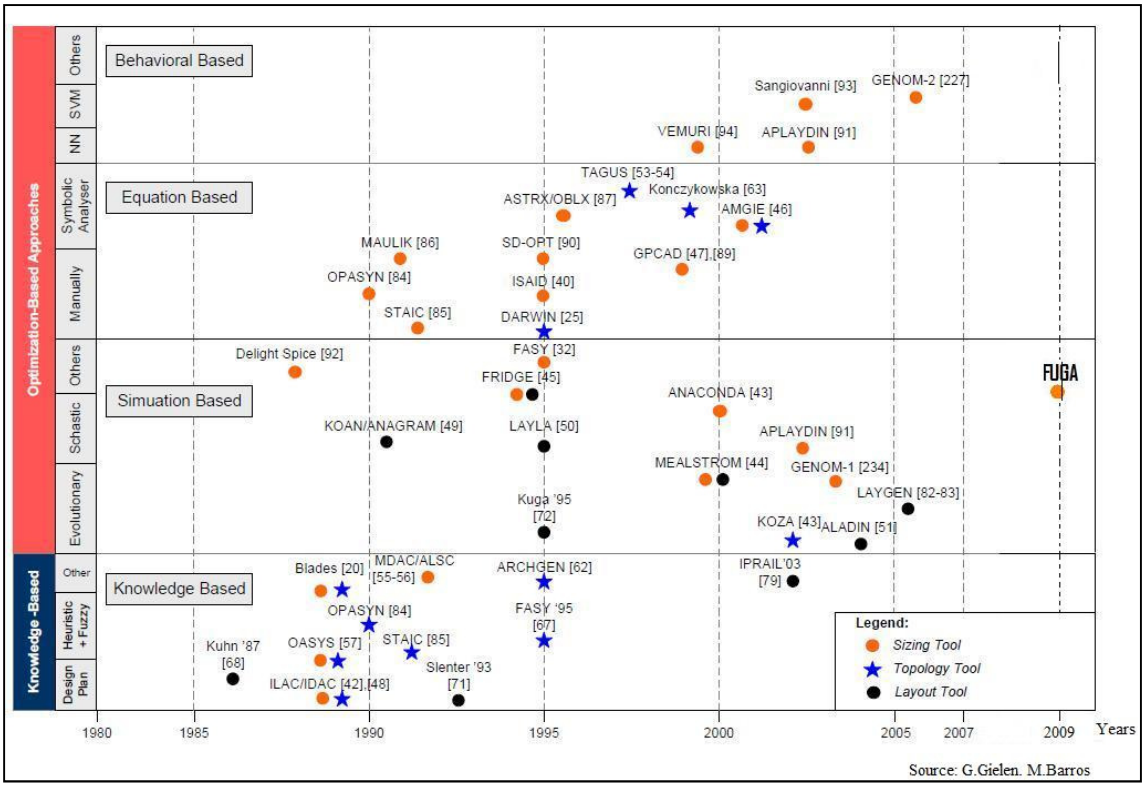


Figure 2-4 Design Automation State of the Art, Tools

From Figure 2-4 is evident that the optimization based approach is the solution of choice recently for most of the develop tools, there will be made a comparison between different methologies, and the correspondent advantages and weekness are analysed.

The knowledge-based approach is characterized by including a complete design plan, describing how the circuit components must be sized to reach the optimum solution of the design problem, Figure 2-5 illustrates a generic block view of this approach. The optimization-based approach uses an optimization engine, instead of a design plan, to perform the design task. The optimization process is an iterative procedure where design variables are updated at each iteration until they achieve an equilibrium point. Concerning performance evaluation, the evaluation engine is typically implemented using an equation-based optimization, or a simulation-based optimization approach.

The knowledge-based approach presented (Figure 2-5), in programs like BLADES. IDAC, OASYS and MDAC/ALSC [11], was the first to appear and is characterized by including a complete design plan, describing the circuit components sizes to reach the optimum solution. For example, the IDAC tool takes advantage of the designer experience to manually derive or rearrange design plans, while OASYS is built over a library of design plans defined for each elementary building block of the library, allowing the representation of hierarchical topologies defined as the interconnection of several elementary building blocks.

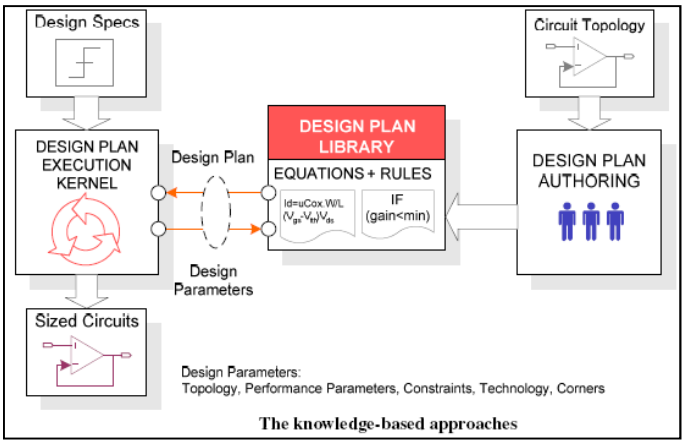


Figure 2-6 Knowledge-base approaches

In these methods (knowledge-based) the main purpose is to encapsulate the designer's knowledge building a pre-design plan with design equations and a design strategy that produce the component sizes values in order to meet the performance requirements (for sizing or layout). This approach presents as major drawbacks the large overhead required to define a new design plan, the reformulation of the entire design plan when expanding the system to new topologies, and finally, the migration to other technologies. Not only it is a very time-consuming process to encode design knowledge for a given set of specifications, but design knowledge also has a limited lifetime. The high rate of progress in process technologies made the acquired knowledge quickly out-of-date. Therefore, the application of these programs to industrial environment has been limited. However, after the design plan is defined, the execution speed of sizing procedure is extremely fast and the quality of the solution only depends on the precision of the models involved in the extraction of the design equations. Naturally, this approach finds its applications restricted to small circuits or to more complex circuits but using simplified equations with the goal of achieving the first cut design[4].

The optimization-based approaches illustrated in Figure 2-6, consist of an iterative loop, including an optimization engine or kernel together with an evaluation engine which exchange information. The optimization algorithm searches through a vast design space for values for each of the circuit components whereas the performance evaluation tool gives an evaluation of the circuit and verifies if performance constraints are met. If the system requirements are satisfied, then a solution is found and the sizes are selected for the topology. The optimization engine function should apply the appropriate techniques to efficiently conduct the search mechanism in order to minimize the number of iterations during the optimization process. Different approaches can be described depending on the type of performance evaluation and the optimization technique employed. Concerning performance, the evaluation engine is typically implemented using an equation-based optimization or a simulation-based optimization approach.

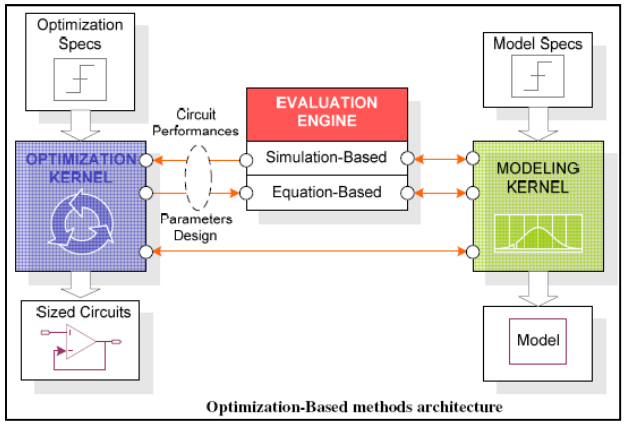


Figure 2-7 Optimization Based Approach

The **equation-based** methods use analytic design equations to evaluate the circuit performance. These equations can be derived manually or automatically by symbolic analysis tools. Then, the problem can be formulated as an optimization problem and normally solved using a numerical algorithm. Some of the most relevant approaches are OPASYN [12], STAIC [13], MAULIK [14], ASTRX/OBLX [15], AMGIE, GPCAD [16], SD-OPT [17]. A promising methodology that has received much attention is related to circuit problems formulated in posynomial form and seen in tools like GPCAD. These techniques take advantage of the development of extremely powerful and efficient interior-point methods for general convex optimization problems [17]. The advantages of the equation-based approach are the short evaluation time and the flexibility. The main drawback is that analytical models have to be used to derive the design equations for each new topology and, despite recent progress in symbolic circuit analysis, not all design characteristics can be easily captured in analytic equations with sufficient accuracy.

The **simulation-based** approaches, such as, FRIDGE, DELIGHT.SPICE [27], FASY [19], ANACONDA [28], MAELSTROM [20] and DARWIN [21], consist of using some form of simulation to evaluate the circuit's performance. In general, these types of tools employ a circuit analysis tool in the inner loop of the optimization cycle to determine the circuit's performance. This is pointed out as a very flexible solution when compared with other methodologies (equation-based, knowledge-based) because it accommodates to any type of circuit topology and accuracy, only, depending on simulator models. Presently the use of SPICE-like simulators are almost generalized and essential to support the optimization engine with all the feedback related to an accurate circuit evaluation, involving different performance characteristics, technological parameters and worst case "corners" validation.

These methods are more flexible than the knowledge-based approaches and can be extended more easily to new circuit types. It is not necessary to have a specific or deep knowledge about the functionality of a circuit to be able to obtain a sized circuit.

A step forward to enhance the efficiency of optimization based methods is introduced by a new class of modeling techniques based in learning strategies. In this class of methods, the behavior of the circuit to be optimized is modeled by a learning mechanism based on the distribution of variation parameters, thus allowing a quick evaluation of the performance for a specific set of design parameters. Some of the most significant behavioral-based methodologies are described by Alpaydin [22], Vincentelli [23] and Vemuri [24]. Besides these efforts some commercial EDA tools for circuit sizing have emerged in the past few years, such as the ADA's. Genius product line now integrated in Synopsis, Barcelona Desig which employ convex optimization techniques and recently the NeoCircuit from Neolinear Inc, which implements a simulation-based approach.

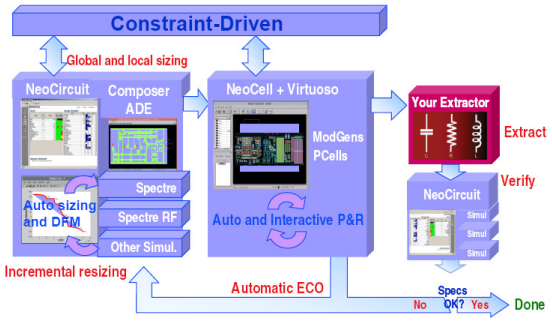


Figure 2-8 Neolinear - NeoCircuit and NeoCell [CADENCE]

2.2.2 Comparative Analysis over Most Recent Approaches

In this section the existing design automation approaches are analysed, taking into account qualitative and quantitave measures following the several metrics that define a set of the desired properties that measure the quality of the final solution.

The computation time is highly correlated with the nature of the evaluation engine. In the knowledge-based approaches the execution speed is the highest of all methods, considering that, the design plan is already defined. Modeling based approaches derived either by numeric equations or by some artificial learning machine method are able to reach solutions quickly, however, the quality of results are always estimated approaches and the solution quality only

depends on the models precision. By contrast, simulation based methods that play with a high accurate circuit simulator are able to produce good quality results, but at the expense of higher execution times.

Factors affecting tools performance

	Knowledge	Equation	Simulation	Learning
Computation Time	+	+	-	+
Setup Time	- -	- - M/-A	+	-
Accuracy	-	-	+	-
Robustness	-	-	+	-

M- means by manual equation and A-automatic by symbolic methods
Symbols ordered from the best to the worst: '+', '–', '– – '

Source: G.Gielen

Figure 2-9 Comparisons between different approach performances

The setup time is a measure of the time spent by the designer to adequate the problem to the synthesis tool. In equation and knowledge-based approaches, this value is normally high and is directly related to the precision of the designed equations that need to be considered. The use of tools to generate equations, like symbolic analyzers, can significantly reduce the input overhead and increase automation levels. In the simulation based approach the level of interaction is the lowest of all methods. Only a few configuration parameters are necessary to setup the data for the external evaluation tool and the optimization algorithm.

With regard to robustness, the most promising classes come from methodologies which are able to produce high accurate solutions like the simulation-based methods, although they require multiple simulations which adversely affect the run-time of the algorithm in a few orders of magnitude. Theoretically, all other approaches could reach the desired robustness in case they are able to produce efficiently accurate models. However, this solution would be impractical due to the large time spent in the preparatory phase to obtain those models.

Trends verified in this area show that the solution for some of the most important approaches lies on the integration of several methods to combine the best of each one, and on the employment of models in order to reduce computation times.

It is difficult to establish quantitative metrics to compare the several analog sizing tools presented on Appendix G. There is not any benchmark program able to realize such task among such different techniques. The usual approach is to test the algorithm/methodology performance over a broad range of circuit topologies observing their behavior against a couple of comparative metrics and draw the conclusion in the basis of these results. Instead, the characteristics of analyzed applications were compared taken into account only some qualitative features[29].

The synthesis approaches described above can be evaluated by several metrics:

- **Robust Design**: As far as sizing is concerned, quality robust design has to do with the accuracy and robustness of the solution. Accuracy is a measure of the quality that shows the difference between the synthesis tool's internal performance prediction mechanisms and the real performance of the obtained solutions, possibly including the layout-induced degradation. Robustness can be described as the capacity of the sizing tool to build and test circuits tolerant to manufacturing faults and operating point variations.

- **Automation Level**: It can be described as the ratio of time needed to accomplish the task of designing a circuit manually to the time spent on designing the same circuit with the help of a synthesis tool. In this metric two aspects must be considered:

    - **Run time response**: The period of time the optimization tool takes to give the first solution to the problem should be as quick as possible.

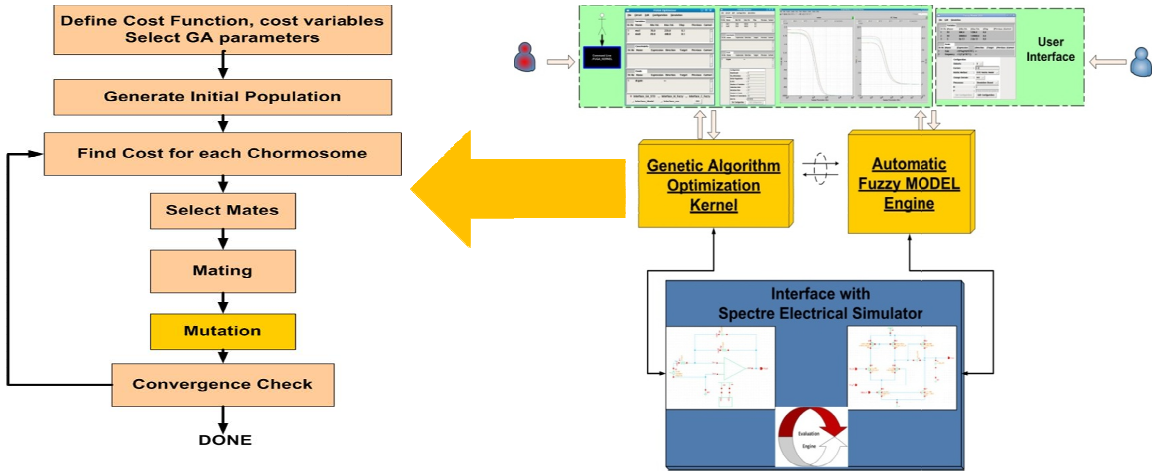
- **Setup time**: it is the setup time and energy needed to prepare a problem in an adequate way for the input to the synthesis tool. This time is often longer than the execution of the synthesis tool. This feature is particularly important because it is strongly correlated with the success and acceptability of the tool. What advantage do we get if some design tool has the remarkable prodigy to output some result in seconds, if it is necessary two months to setup the complete algorithm of a hypothetic circuit when it is known it could be designed by hand in one month? None!

- **Scope of the tool**: It can be described as a group of analog design problems, which can be solved by this tool. This is an important characteristic/feature for analog design, because these problems usually require several types of optimization techniques. An analog synthesis tool which aims at solving a wide range of design problems will be successful in the long run/during a long time period, whereas tools planned to solve a narrow range of problems will soon be out of date.

- **Design facilities**: It can be described as the set of additional tools that can enrich a synthesis tool:
  - **Multi-objective Optimization**. The DA tool presents the final solutions in terms of a set of designs representing the different complementary tradeoffs of application of specific objectives (for example, area versus consumption) instead of single design response.
  - **Interactive Design**. The tool optionally produces intermediate performance reports (in the form of text or graphics) throughout the design execution time to inform the IC designer about the progress of optimization. At the same time, the IC designer optionally has the possibility to interact with the tool in real time manner to tune up some parameters, e.g., the dimension of a transistor or the redefinition of some design bias or simply to quit the tool.
  - **Bookkeeping Facilities**. The tool should have additional capacities to help with the introduction and management of all the necessary data (namely) including the management of different technological files, different types of circuits (e.g., operational amplifiers, phase-locked-loops, etc.), different performance measurements for the circuit type, with different design parameters, with different available components or different topologies, and so on.
  - **Encapsulate (Hiding) Details** – Some tools interact with external programs and so it makes sense that the interface with these additional tools can be made in an automatically way hiding unused options. Also, the complicated aspects of the tool or the algorithm should be encapsulated in order to be accepted and used by the largest number of users.

2.3 Proposed Approach

The goal of this approach is to create an innovative circuit-level optimization kernel that can deliver a sized circuit from a given circuit specifications. It is a difficult and critical step in the design flow since most analog designs require a custom optimized design and the design problem is typically under-constrained with many degree of freedom and with many (often conflicting) performance requirements to be taken into account. Typical optimization algorithms used are simulated annealing and genetic or evolutionary algorithms. The proposed approach, first, models analog design knowledge using soft computing techniques, than, enhances a stochastic optimization kernel by embedding the design knowledge model. The proposed approach uses common IC design environments, and electrical circuit as the evaluation engine, (also an experimental equation based approach is included) and is validated for well known design examples.



2.4 Conclusions

Automated design of analog circuits, also referred to as analog circuit synthesis, has been the subject of active scientific research for many years now. This chapter has covered some of the most significant tools and methodologies to analog IC design automation. Here, a set of general properties that allow us to characterize each approach have been identified and a better insight related to advantages and limitations have been presented.

Actually, the use of design management platforms, like the Cadence® Virtuoso platform, with a set of integrated CAD tools and database facilities to deal with the design transformations from the system-level to the physical implementation, can significantly speed up the design process and enhance the productivity of analog/mixed-signal integrated circuit (IC) design teams. These design management platforms are a valuable help in analog IC design but they are still far behind the development stage of design automation tools already available for digital design. Furthermore, in order to be competitive in the near future new design automation tools should also be fully compatible with future computer-aided design frameworks and with open standards.

3 FUZZY SYSTEMS TO ENHANCE GENETIC ALGORITHM KERNEL

The proposed fuzzy-genetic optimization approach, rises on a genetic algorithm (GA) standard kernel implementation. Therefore, the implemented GA - Fuzzy approach, which will serve as a benchmark to evaluate the new approach, will be described in this section.

Genetic algorithms, evolutionary computation have a number of qualities in common, including the fact that all evolve solutions, all utilize some kind of selection based on survival of the fittest, and all invoke some sort of evolutionary manipulation such as crossover or mutation.

Fuzzy-logic based systems have been specially proposed to deal with “uncertain” information and have proved to be very efficient in capturing human expertise [19].

Designing complex analog and mixed-signal circuits is a very difficult and cumbersome task that requires extensive design expertise and effort [32].

That’s why the association of fuzzy logic with evolutionary algorithms is considered a viable solution, for increasing the analog and mixed signal design productivity.

The basic concepts are common to every algorithm approach to problem solving. Figure 3-1 illustrates a generic model based design phases. In this work an effort was made to follow a standard system development approach, where the fist step consisted on the requirments and specification definition phase, that can be seen in part as the collection of every necessary requirements for the system to answer and reach the desiered solution, supported by a strong state of the art study.

The design phase, the system archecture and modules division were defined and structured. The implementation phase consisted on the software coding and circuit testing according with the defined design, but in an iteractive way, redesigning sometimes when it shows necessary to adapt and optimize the implementation and finally the test and verification phase, experiment a set of circuit test bench to have metrics, and validate the developed system.

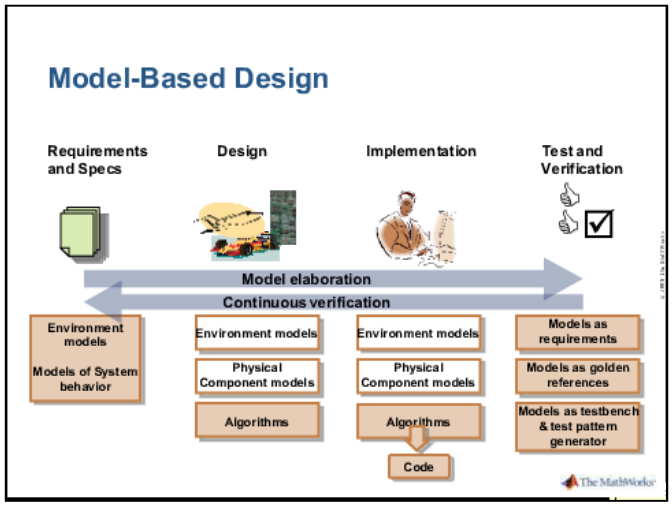


Figure 3-1 Generic Model System

3.1 Fuzzy Model: Modelling Kernel

Since it was introduced by Zadeth in 1965, fuzzy logic has demonstrated to be an adequate tool for emulating the approximate reasoning mechanisms used by the human brain. Inference systems based on fuzzy logic share certain features with other intelligence paradigms. A fuzzy system acquires and represents its knowledge base symbolically, while knowledge is processed numerically. The first feature permits the use of fast and simple mechanisms for representing and acquiring knowledge, structured in rules, and the second allows the employment of fast numerical algorithms and the direct implementation of the system by a microelectronic circuit. The ability of fuzzy logic to describe a complex system by means of simple and intuitive set of behavioral rules has motivated an increasing interest for applying it in fields such as industrial control, non-linear system modeling, and decision –making systems[31].

In this approach first, the search space, a R^n space, is limited on each dimension by the range associated to each optimization variable, which may be discrete or continues. Naturally, each variable will be represented by a gene and the complete set of genes (variables) compose the chromosome, which represents a point inside the n-dimensional search space. Then, the implementation of each GA steps were intentionally made simple, once, the aim of this study is to evaluate the ability of the fuzzy model, representing design knowledge, to improve the optimization kernel.

Moreover, as will be seen later, the proposed approach applies to any other GA implementation. Therefore, the GA steps are implemented as follows: the initial population generation is random, the paring operator selects the pairs at random from the upper half of the population, the crossover operator uses a single point, the mutation operator randomly selects the genes to be mutated. Finally, the cost function measures the absolute distance of each chromosome, solution, to the desired performance goals.

3.2 Fuzzy Model Implementation

The fuzzy model is defined in common 5 steps approach, as follows:

- membership functions
- fuzzy rules
- Inference algorithm
- defuzzification
- solution normalization

The membership function is applied for all input variables (in our case, the input variables are the genes that compose the chromosome) and can be either triangular or Gaussian, decided by the user at this level. The membership function converts an element x in crispy values into fuzzy a standard value $\mu(x)$ set between [0, 1].

The fuzzy rules are defined based on the circuit behavior and their general form is given by if-then rules, as the following: “If input is low then output is max”, where low and max are membership function terms. For simple cases these rules can easily be described manually, however, for more complex structures it is clearly impossible to infer such information in a reasonable time. Here, a five step procedure, described in table 4.1-1, is considered to automatically generate the fuzzy rules based on a Design of Experiments DOE approach to determine the optimal samples set. DOE techniques provide a mathematical basis to select a limited but “optimal” set of sample points needed to fit a black-box model. Well-known and often-used sampling schemes range from full and fractional factorial design, over Placket-Burman and Taguchi schemes, to Latin hypercube and even random design.

The fuzzy inference algorithm is applied to activate a subset of rules from the complete set of fuzzy rules, this in agreement with the fuzzy values. There are many operators to implement the inference mechanism, e.g, MIN, MAX, SUM and PROD or a hybrid solution with more than one. Here, the inference is implemented based on a MAX-MIN strategy. As

mentioned before, the input variables are converted from crispy values to fuzzy values. Therefore, it is needed to defuzzify the result of the fuzzy inference.

Table 3-1 Procedure for Determining the Fuzzy Rules

Step	Description
1	Convert the design space from b^k to b^{k-p} fractional factorial design space.
2	Determine the new combination of sample points for the fractional factorial design.
3	Determine the performance outputs for the selected sample points.
4	Evaluate the main effects of the control parameters (input variables) on each output.
5	Generate the fuzzy rules.

A defuzzification is a process to get a best possibility representation of the distribution of an inferred fuzzy action. Several methods can also be applied here, e.g., “center-of-gravity”, “center-ofarea”, “first-of-maxima”, “last-of-maxima”, “middle-of-maxima”, “center-of-area for singletons”, etc. Here, the center-of-area approach (1) was selected.

$$x_0 = \frac{\sum_{j=1}^{n} \mu(x_j) \times x_j}{\sum_{j=1}^{n} \mu(x_j)} \qquad (1)$$

At last, the result returned by the defuzzification method is always a rated value. Thus, we must analyze this result to take a final decision.

3.3 Fuzzy-Genetic Algorithm

The previously described fuzzy model was integrated in a GA flow in order to introduce design knowledge during the search process and improve the efficiency of the optimization kernel.

In the present case the fuzzy model will be used to support the GA’s mutation operator, as illustrated in fiure 3-2. As mentioned before, the mutation replaces the value of a gene with a randomly-generated value from the domain defined for each input variable. But sometimes, those changes may move the individual to an undesired or unpromising search space region, especially when large domains are considered. The inclusion of the fuzzy model allows a fine control of the new gene randomly-generated by indicating a more promising orientation for the new gene value.

The change on the mutation operator based on the fuzzy model is shown in Figure 3-2, for the case of a mutation in variable 2. In step 4, the results for Gain and Frequency are obtained by applying the defuzzification, considering all the input variables in the selected chromosome. The result is achieved using a triangular shape, three levels of subsets and is applying the MAX-MIN inference method.

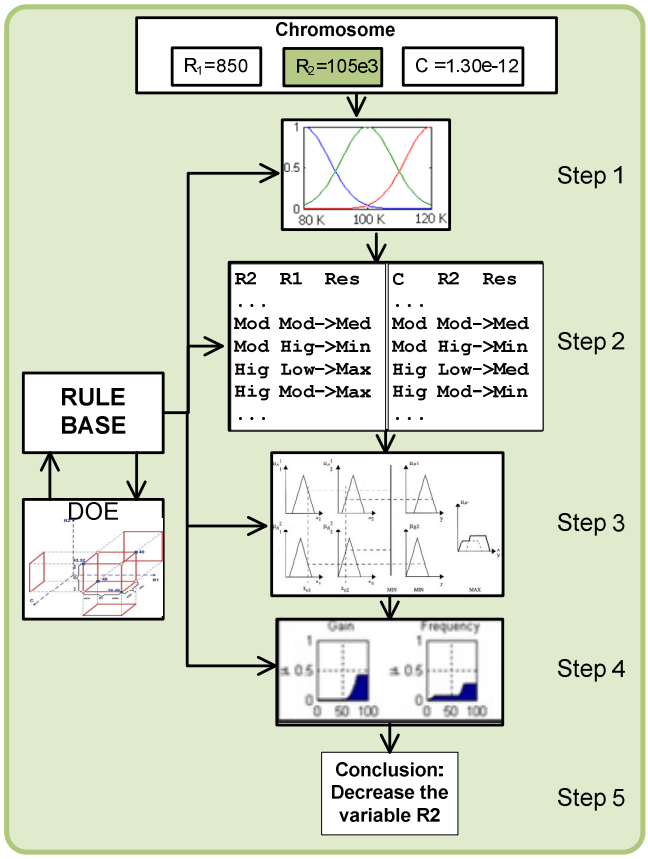


Figure 3-2 Fuzzy-Genetic Optimization Kernel (FUGA).

Under the center-of-area method, the result from defuzzification gives a 64.76% and 50.94% output for the Gain and Frequency, respectively. As the results show, the Frequency's defuzzification presents a more stable output then Gain's result.

The final step of fuzzy based mutation operator would be to analyze the defuzzification results and return one of three possible alternatives:

- Increase the optimization variable value;
- decrease the optimization variable value;
- Generate a new random value in a small range around the present value.

when the conclusion process chooses the third alternative, it means that the present value (cell proposed to mutation) of input variable is already near to the best goal. For that reason, a small change around the current value must be defined and a new random value is calculated. It guarantees that the fitness of that chromosome would not have large variations. Otherwise, when the conclusion process chooses the other alternatives, the increase or decrease of the cell must be done regarding the limits of the gene domain.

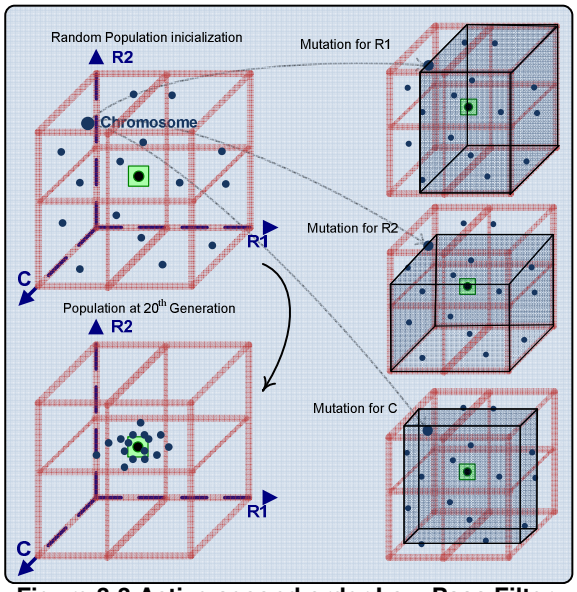


Figure 3-3 Active second order Low-Pass Filter.

For the working example, and considering the chromosome presented in Figure 3-3, a graphical illustration of the population convergence is shown in Figure 3-3. Moreover, the qualitative information obtained after the defuzzification allows the following conclusion, to be applied by the mutation operator: a mutation for R1 will result on an increment of the present value, a mutation for R2 will result on a decrement of the present value and, finally, for variable C the fuzzy model concludes that the present value is near the main goal, once the output from the defuzzification is stable, so, in this case, a random mutation will be considered, allowing only a small change around the present value.

3.4 Conclusions

In this section, the fuzzy model conceptes and structure was described, then the process to automatically generate the model was discussed and, finally, the proposed fuzzy model integration on the GA flow example was given to help understand how the model will influence and guide the genetic algortihm kernel.

4 THE SYSTEM ARCHITECTURAL DESIGN

4.1 General Description

This chapter describes the overall design for the optimization kernel implemented architecture, where the external interfaces, internal modules, and the interaction between each element, defines the necessary process for the system to reach the desired solution, in a reasonable time and effective steps.

The software implementation was performed in a UNIX-like environment (Linux). The programming language chosen was C, the electric circuit simulator is spectre and python for user interface.

4.2 Kernel Implementation Overview

The optimization kernel has a structure that allows the independent development of each external interface, and also allows different approaches introduction of these external elements in the system without changes over the optimization kernel internal work flow.

The Figure 4-1 illustrates a generic view over all the elements that interact with the optimization kernel.

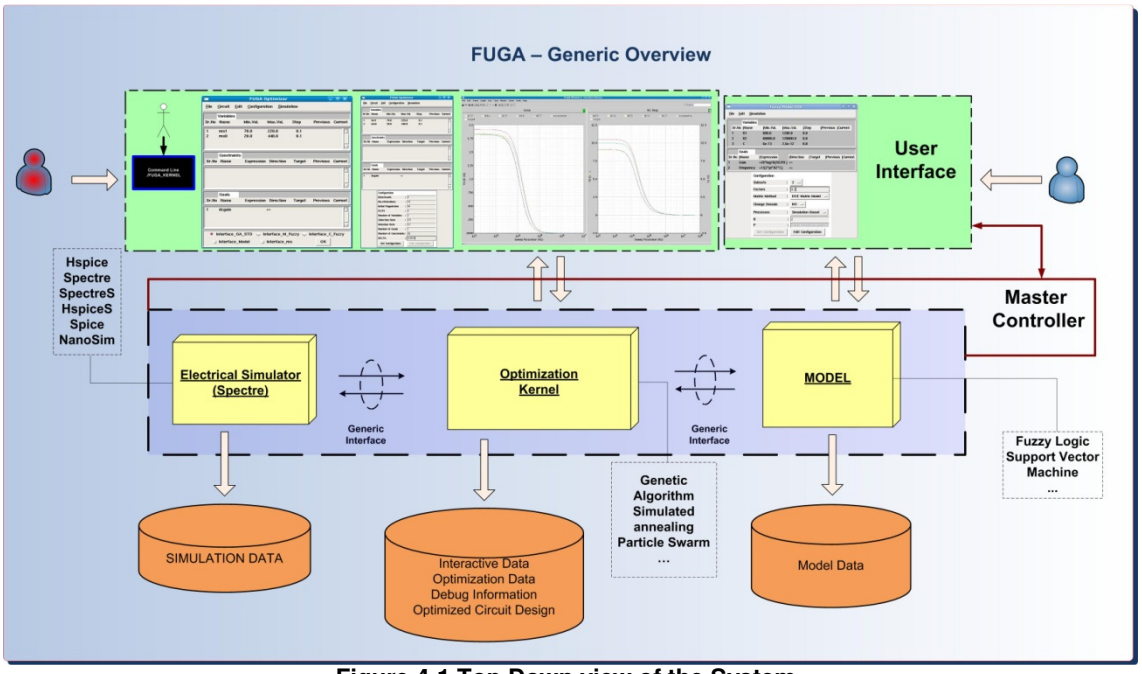


Figure 4-1 Top Down view of the System

4.3 User Interface

In module **User interface**, we first use Cadence interface features to complement the design work flow, and these features helps to design the circuit schematic with Virtuoso Schematic. The simulation using Cadence Analog Environment and correspondent measurements using the inbuilt Cadence Calculator provides a large number of circuit measurement functions. This environment also allows the graphic results visualization, and more important for the implemented Kernel functionalities the circuit "*Netlist*". Also the Cadence own Optimizer tool can be used and comparison between the obtained results can be made, in

the Results section some examples are described using the Cadence Optimizer tool and the developed Optimizer.

User interface module has a set of input files needed by the optimization kernel and some external elements as spectre circuit simulator and the model module.

These input files have information about the genetic algorithm configuration, the optimization variables ranges and steps, the circuit constraints and specification; The circuit "Netlist" generated with Cadence Analog Environment feature, written manually or generated with other tool as an example the Spice Reader, but it has to have spectre syntax, in this approach where we use spectre circuit simulator, the measurement file constructed with a specific spectre measurement description language – " SpectreMDL ", which is a scripting language that we can use to control the Spectre Analog simulator and the Wave Scan display, and the parameters file generated by the Kernel where the optimization variables values to be simulated are stored; The information about the model module configuration are stored in specifics files that are used as input for the model and the model generated or updated files are used in kernel flow when the model option is active. In section Detailed Design a complete explanation for all the folder structure and the information in each input file are described more deeply.

The Kernel can be used as a standalone tool, with a graphical interface, developed using python, with this interface all the algorithm configuration, model configuration and optimization variables parameters can be introduced by the user as illustrated in Figure 4-2. This user interface than generates the necessary input files for the kernel input.

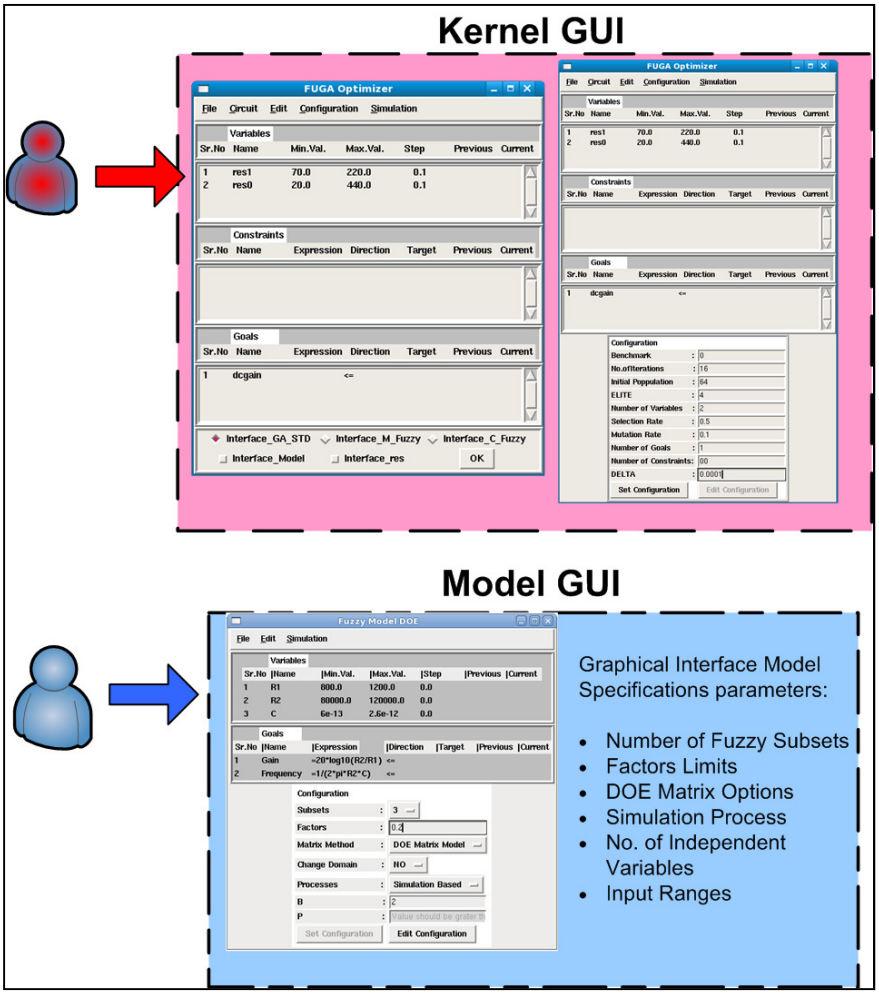


Figure 4-2 User Interface Module

4.4 Internal Modules Detailed Design

In this chapter it's explained in detailed every internal module block, in a top down view. Figure 4-3 illustrates the main internal modules to give an image of what it's going to be discussed in this chapter and some generic interaction between them. In figure 4.4 the INIT module has attached some inputs files where the algorithm configurations are stored.

The internal modules illustrated in Figure 4-3 are a set of ".h" and its correspondent ".c" divided in this distinct way, to handle the complexity of the system, where one module can be implemented with different algorithms. For example for modules like sort, pairing, mating and mutation there are several algorithms proposed to sort the population, or pairing algorithms etc.

In these modules we can find different implementations and this separation gives the developer the freedom to add alternative implementations and compare the results in a clear view. The modules that handle the fuzzy module are also separated, and also the functions responsible to handle the interaction with the electric circuit simulator are gather in a particular module that will be explained in details in this chapter. Figure 4-4 shows the repository organization, some internal files and folders. The fuzzy results folder has the output results when using the optimization kernel in fuzzy mode; the GA results folder has the output results when kernel is running with genetic algorithm standard approach.

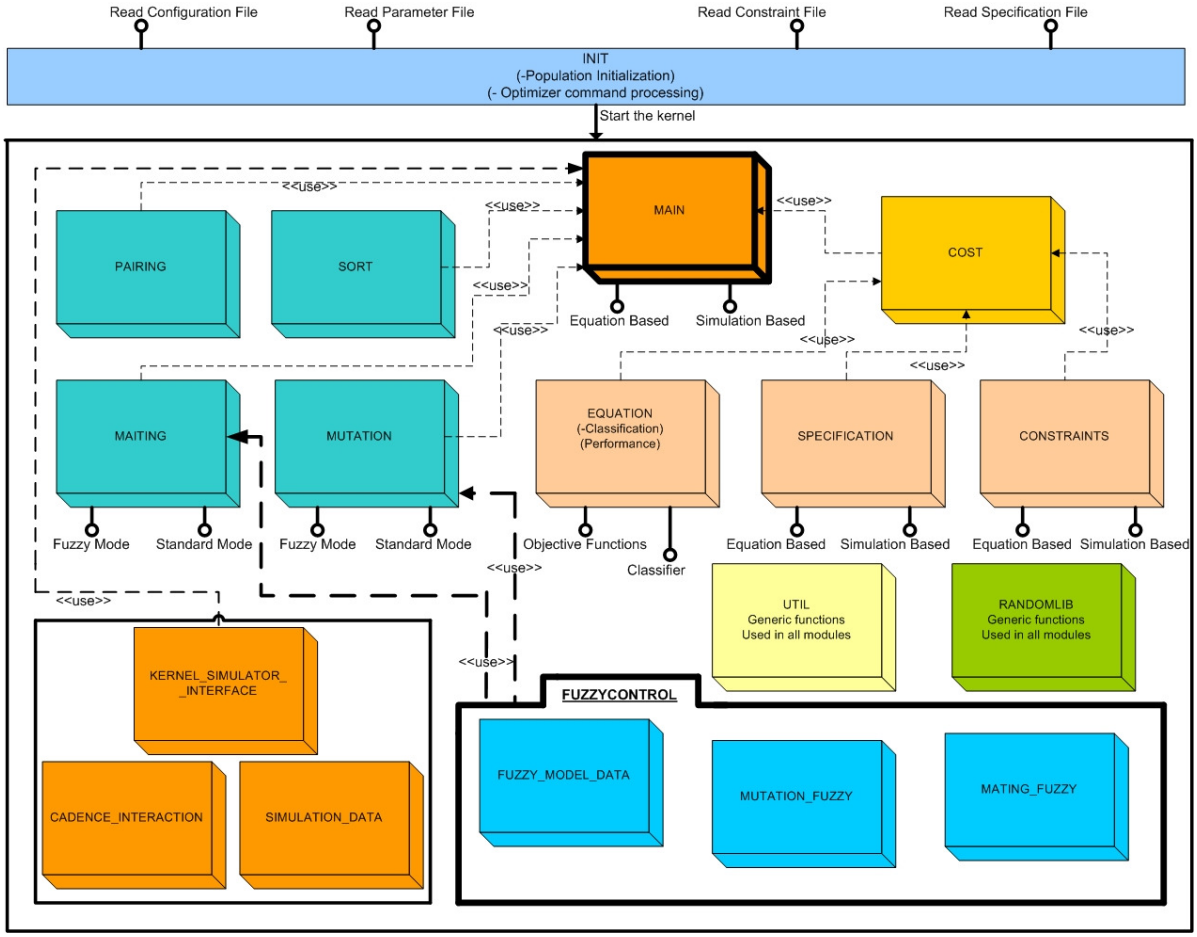


Figure 4-3 Internal Modules Detailed Design

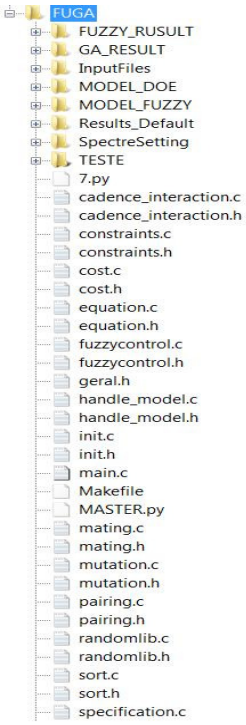


Figure 4-4 Source Code Repository view

4.4.1 Data Structure

The system source code is divided by several modules, organized in different files with different scopes as illustrated in Figure 4-3. The header file “geral.h” in source code holds the generic data structure that contains those variables needed in more than one module, so they are visible and can be shared everywhere necessary. The genetic algorithm population is declared as a global variable given the fact that this population is used for almost all the system modules for different purposes, like in mating process, cost calculation, mutation process etc…

The population has to be sorted with a given criteria and it is the data structure that holds all the chromosomes that represents possible solutions for the optimization problem. Figure 4-5 shows the population structure where we can see that the population is a list of pointers where each element points to a chromosome that have a set of correspondent variables that characterizes this population individual, and the variable **genes** represents one of the optimization problem solution, with the list of circuit components values.

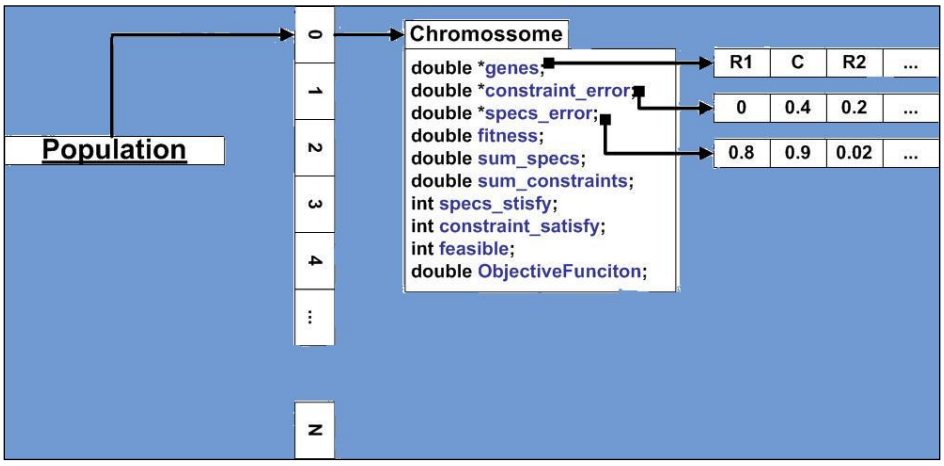


Figure 4-5 Population Data Structure

The remaining variables holds the values necessary to analyse in the optimization flow, if the given gene values satisfy the specifications, constrains, if it's a feasible solution and the correspondent fitness value that is updated in module "*cost*". The number of individual in the population is set by the user in a configuration file, so it's fixed at the beginning of each new optimization process.

The **genes** allowed range of variation, and the correspondent name for example R1, C … are read from the parameter file and stored in a structure that hold all the genes names, maximum values, minimum values and the configured step of variation.

The **parents** are stored in a structure that has the list of mothers and list of fathers, to be mixed in the mating process; these are represented as integer because they will have only the index number that corresponds to an individual in the population that will act as mother or father defined in the mating algorithm.

To store the **circuit specifications** introduced by the designer like the dc gain and frequency for example, it's allocated a structure with a set of the entire specification goals ID that contains the specification name and correspondent values. To store the **circuit constraints** we have a similar allocated structure with the constraint name and the correspondent values. Figure 4-6 illustrates the kind of structure used to store these values. The allocation is limited by an integer global number read from the configuration file, the specifications goal limit N is SPECS_OUT and the constraints goals limits N is NumConstraint, and these two structures for specifications and constraints are also filled at the beginning of the process as it will be explained in section 5.4.3. These two structures are global since they are going to be accessed from different modules.

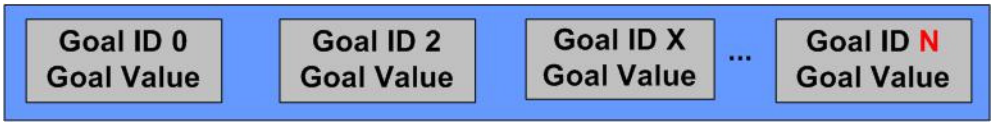


Figure 4-6 Goals Data Structure

After the electric simulator finishes the circuit simulation, several output files is generated and the results needed for the optimization is processed and filtered. **The simulation result data** used after being extracted are stored in the structure MesureData. Figure 4-7 shows one element of this structure, what we have is N elements with this illustrated structure where N stands for the number of individual in the population, so the output data is stored and correctly indentified with the correspondent index on the population.

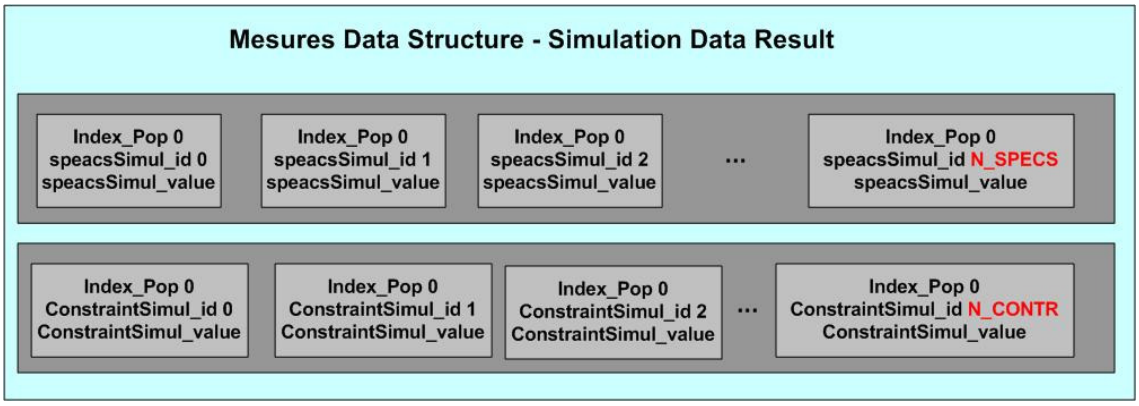


Figure 4-7 Simulation Result Data

For the **fuzzy model data** inside the optimization kernel there is a structure that has the gene index value, one integer that indicates the increment or decrement of the gene value, and the mutation percentage to be included, this structure stores the output of the module model fuzzy that given one individual it returns these information to guide the mutation algorithm.

These are the main global structures used in several modules in the optimization kernel, then each module has other structures internal and sometimes shared between some, but they will be reported later in next sections. Some constants values read from the input files are also store and kept in global variables like the Selection Rate, the Mutation Rate, the number of optimization run, and some auxiliary variables used in more than one module.

4.4.2 Kernel Input and Output Data

This section explains the kernel external interface. The optimization kernel interacts with all the external elements in the system exchanging data with them, for example the electric circuit simulator, where this iteration is controlled by the Master element, the input data given directly by the user in common line or by the user graphic interface, the data exchange with the module fuzzy model, and finally the kernel output report that allows to follow the optimization evolution and all the data it produces, and these data are also used to validate the optimization and verify that the problem specification are satisfied.

Some data examples are given and described in order to understand what the contents of these data means, what is necessary to give to the optimization kernel to have a successful simulation, and what the kernel outputs in return.

The Optimization Kernel can be lunched in common line with its arguments as it is explained in the user manual in Appendix A and shown here:

./KERNEL –cfg configuration_file.cfg -par parameters_file.par
–spc specsification_constraint.spc [OPTIONS] [OUTPUT FILES]

To successful lunch the kernel program it's mandatory that a configuration file and a parameter file is given as input arguments. Otherwise the kernel produces an error and it's not possible to start the optimization.

The configuration file has the genetic algorithm main configuration to initialize the optimization operators, and these inputs have a major impact on the algorithm behaviour, consequently the convergence. Figure 4-8 illustrates an example for the configuration file with the comments, which explains each entry value (the comments are represented by a line starting with the character "#").

```
#
# This File has the configuration parameters for the Genetic Algorithm (GA)
# kernel
#
# The parameter are orginized in the following order:
#
# 0 Benchmark considered                          BENCHMARK
# 1 Number of GA cycles                           nRUNS
# 2 Total individual in the population            Pop_inicial
# 3 A selected number of the best individual      ELITE
# 4 The number of the optimization variables      NumVar_Genes
# 5 The probability of the selction process       Selection_Rate
# 6 The probability of the mutation process       Mutation_Rate
# 7 The number of optimization specifications     SPECS_OUT
# 8 The number of optimization constraints        NumConstraint
# 9 Parameter that improves the sort acuracy      DELTA
#

0
40
16
2
2
0.5
0.1
1
1
0.0001
```

Figure 4-8 Configuration File

The parameter file has the optimization variables names our “genes”, maximum and minimum values, the step for the random initialization, and the information explaining each value in the parameter input file in comments as shown in Figure 4-9 (lines starting with “#”), and if using the graphical interface these data are also placed in the parameter input window separator.

More than these input files; kernel receives the following optional flags:

– The fuzzy mutation flag [- fm] is specified as an optional input argument. This input flag enables the kernel to set the variable that determines what type of mutation algorithm is going to be used in the mutation process. When this flag is present in the input arguments the mutation algorithm used is the one guided by the fuzzy model, if this parameter is not specified as an input argument the mutation operator will have a standard behaviour based on random generated values.

```
#
# This file has the optimization variables parameters
#
# File orginization:
#
# variable name
# minimum variable value
# maximum variable value
# step
#
# ... Repeats for n = number of the optimization variable
#

res0
70
440
0.1

res1
2
16
0.1
```

Figure 4-9 Parameters File

– The Fuzzy crossover flag [- fc] an optional input flag also, indicates the kernel to use in the pairing process the fuzzy model guidance, and if this option is not specified the crossover operator will have a standard behaviour.

– The optional flag [- CIRCUIT_SIM] defines that the electrical circuit simulator is to be used in the optimization process as the evaluation engine. The optimization kernel can operate in two distinct modes, the equation based mode and the simulation based mode. When this option is specified the Kernel operates in the simulation based mode, otherwise it will operate in equation based mode.

– The optional flag [- model] allows the optimization kernel to lunch the Design Of Experiment Model DOE. To use genetic algorithm Kernel combined with fuzzy model, this option must be specified at least for the first time when the optimization process begin, so the external model module DOE will generate the necessary outputs with the model conclusions of how the input optimization variable influence the circuit output and the fuzzy logic rules that will be the inputs needed by fuzzy model module to guide the algorithm [Appendix A].

– The optional flag [- res] determines the optimization result data folder name and place. This option can be specified if there is a need to specify the folder where the optimization data and debug result are going to be placed; otherwise the results are placed in a default folder [Results_Default] in the directory that the program was called.

Now is time to explain the kernel output data. The optimization kernel outputs two main types of data, the output report data and the interaction data with the external elements. The output report data have all the information regarding the GA population evolution through all the generation, and reports the algorithm data in each optimization step. The output folder has several output files with report data, the output file **PopulationData.txt** allows the user to make sure that the configuration introduced to the kernel is the one uploaded and in use, as can be checked in this example Figure 4-10, where at the beginning of this file it's printed out the genetic algorithm configuration being used, like the population size, and the algorithm operators like mutation rate, the parameters range etc…

```
+-----------------------------------+
|===================================|
|            EVOLUTION              |
|         -================-        |
|          POPULATION DATA          |
|            Optimizer              |
|                                   |
|===================================|
+-----------------------------------+

The optimization algorithm configuration:
Benchmark equation= 0
 Number of generation = 40
 Population size = 16
 Number of Elite Chomossome= 2
 Number of variable genes = 2
 Selection Rate = 0.500000
 Mutation Rate = 0.100000
 Number of Specification = 1
 Number of Constraint = 1

The optimization parameters:
i=0 name=res0, min=7.000000e+01 and max=4.400000e+02
i=1 name=res1, min=2.000000e+00 and max=1.600000e+01
***********Welcome to EVOLUTION initial population without COST********
***************Chromossome= 0 **********************
Sum of specification = 0.000000e+00
The Obejective to optimize= 0.000000e+00
Sum of the constraint = 0.000000e+00
fitness = 0.000000e+00
number specs satisfied= 0
number of constraint sat. = 0
Chomossome feasibility = 0
    genes[0]=1.130000e+02
    genes[1]=1.550000e+01
    specification_error[0]= 0.000000e+00
     constraint[0] = 0.000000e+00
 ***************Chromossome= 1 **********************
Sum of specification = 0.000000e+00
The Obejective to optimize= 0.000000e+00
Sum of the constraint = 0.000000e+00
```

Figure 4-10 Population Data File Head

More than the configuration validation, this file PopulationData.txt allows the user to follow the evolution of all the individuals' in the population after each step, because all the population data is dumped into this file. Figure 4-11 is an example of this file during the mating process, where can be seen which are the chromosomes chosen to be the parents and it's possible to indentify the new offspring and the population state after this process.

```
 ***************Chromossome= 15 ***********************
Sum of specification = 5.686275e-01
The Obejective to optimize= 5.686275e-01
Sum of the constraint = 0.000000e+00
fitness = 5.686275e-01
number specs satisfied= 0
number of constraint sat. = 1
Chomossome feasibility = 0
    genes[0]=1.130000e+02
    genes[1]=1.550000e+01
    specification_error[0]= 5.686275e-01
     constraint[0] = 0.000000e+00
 ************EvolutionIII runn 0********************
mother_ind. = 1, father_ind. = 2
mother_ind. = 5, father_ind. = 7
mother_ind. = 1, father_ind. = 4
mother_ind. = 2, father_ind. = 4
j< crossover_point=2

pop[keep=8 + k=0]->genes[j=0]= pop[pa->mom[i=0] =1]->genes[j] = 197.300000;
pop[keep=8 + k=0 +1]->genes[j=0]= pop[pa->dad[i=0] =2]->genes[j] = 414.500000;
pop[keep=8 + k=0]->genes[j=1]= pop[pa->mom[i=0] =1]->genes[j] = 2.800000;
pop[keep=8 + k=0 +1]->genes[j=1]= pop[pa->dad[i=0] =2]->genes[j] = 7.100000;
pop[keep=8 + k=0]->genes[j=2]= pop[pa->mom[i=0] =1]->genes[j] = 0.000000;
pop[keep=8 + k=0 +1]->genes[j=2]= pop[pa->dad[i=0] =2]->genes[j] = 0.000000;
j= crossover_point=2 && j<NumVar_Genes
```

Figure 4-11 Mating Output Report

In this file as explained before, it's also possible to see which are the chromosomes selected for the mutation process and the correspondent values as printed in this example of Figure 4-12.

```
  ******************Mutation START**************************
For Chromossome 3 the random mutation value is 198.850152 in gene 0
For Chromossome 8 the random mutation value is 72.599009 in gene 0
For Chromossome 7 the random mutation value is 76.220626 in gene 0
 ********After Mutation ***************************
***************Chromossome= 0 ***********************
Sum of specification = 7.325000e-03
The Obejective to optimize= 7.325000e-03
Sum of the constraint = 0.000000e+00
fitness = 7.325000e-03
number specs satisfied= 0
number of constraint sat. = 1
Chomossome feasibility = 0
    genes[0]=2.707376e+02
    genes[1]=2.800000e+00
    specification_error[0]= 7.325000e-03
     constraint[0] = 0.000000e+00
```

Figure 4-12 Mutation Output Report

The data in the PopulationData.txt file is organized in the following way, first it presents the particularity of the optimization process step, and after it presents all the chromosome data for the entire population, so we can monitor the population state after each step, and check what happened in each optimization part.

The output file **GenesEvolution.txt** placed in the report output folder has the optimization variables evolution; in this file the user can monitor the genes values evolution for the best

chromosome in the population in all the generations. This data also allows the generation of graphic views for the genes values variation. Figure 4-14 is an example of the data in this file.

```
1  res0 res1
2  Generation= 0 4.186000e+02 5.400000e+00
3  Generation= 1 1.973000e+02 2.414479e+00
4  Generation= 2 1.973000e+02 2.414479e+00
5  Generation= 3 1.973000e+02 2.414479e+00
6  Generation= 4 2.636757e+02 2.800000e+00
```

Figure 4-13 Genes Evolution

Another file inside the result output folder is the file **graph_default.txt** that has the best chromosome fitness value evolution through all the generation. This information also allows monitoring the algorithm performance and helps to know if we reach the desired solution, the convergence or if the algorithm stagnated at some point.

```
0 5.531500e-02
1 4.386500e-02
2 4.386500e-02
3 4.386500e-02
4 1.306500e-02
5 1.306500e-02
6 1.306500e-02
7 1.306500e-02
8 1.306500e-02
9 7.325000e-03
10 7.325000e-03
11 7.325000e-03
12 7.325000e-03
13 7.325000e-03
14 7.325000e-03
15 7.325000e-03
16 7.325000e-03
```

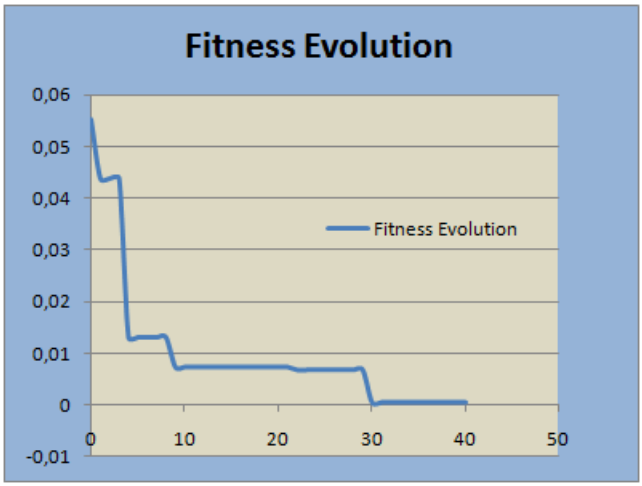


Figure 4-14 Best Chromosome Fitness Evolution

In this same result folder can also be found the two more files, the file PopulationSorted.txt that has just the fitness values and the errors for all the individual in the population for debug purpose and the DEBUG_FUZZY.txt also allows us to monitor the values that the module model fuzzy returns to the genetic algorithm kernel.

The exchange of data between the GA optimization kernel and the module model fuzzy is handled directly by function return values, which will be explained in detailed in section 5.6, given the fact that the model is integrated with the optimization kernel as an additional library included in the source code.

Before each electric simulation needed in the optimization flow the optimization kernel generates an output file with the data necessary for the electric simulation. This file is one of the inputs for the electric simulator spectre in this case. This input file is called **InputParameter.scs** and is placed in folder SpectreSetting in the source code structure, which has all the electric circuit simulation related files. Figure 4-15 is the example for this file that is generated according with MDL rules. After some search for a better way to introduce this data to the spectre simulator, this was the best choice found in the manual .This file InputParameter.scs is included in the circuit netlist, the circuit.scs. It has the entire population genes, as illustrated in Figure 4-15, the "_id_" column identifies the index of the chromosome followed by the genes value. Using this MDL option "data paramset" it was possible to give the spectre simulator all the population to simulate as one big set of consecutives simulations, and with spectre internal mechanism it identifies each line as being one simulation, it iterates over all the individual and produces the results. So to understand this example will be 15 different simulation given once to spectre which will output the result after finishing all the simulation but this will be explained later next section.

```
simulator lang=spectre
data paramset {
_id_ res0 res1
0 2.888896e+02 2.882481e+00
1 2.888896e+02 2.882481e+00
2 3.486898e+02 2.882481e+00
3 2.888896e+02 2.882481e+00
4 2.888896e+02 2.882481e+00
5 2.888896e+02 4.163935e+00
6 2.888896e+02 2.882481e+00
7 2.888896e+02 2.882481e+00
8 2.888896e+02 2.882481e+00
9 2.888896e+02 2.882481e+00
10 2.888896e+02 2.882481e+00
11 2.888896e+02 2.882481e+00
12 2.888896e+02 2.882481e+00
13 2.888896e+02 2.882481e+00
14 2.888896e+02 2.882481e+00
15 2.068607e+02 2.882481e+00
}
```

Figure 4-15 Input Parameters Data

The second type of kernel data output can be monitored looking at this file ComunicationKernelMaster.txt that the kernel generates after reading the data from the Master Controller pipe output that is the element handling the electric simulator output results, this file shows the data in the input buffer, and their location in the simulation data result structure.

```
num_measures= 16
begin= 0
end= 16
[KERNEL]::measure_buf[i=0] = 1.725490e+01
[KERNEL]::measure_buf[i=1] = 3.412470e+01
[KERNEL]::measure_buf[i=2] = 2.807590e+01
[KERNEL]::measure_buf[i=3] = 3.778740e+01
[KERNEL]::measure_buf[i=4] = 2.744980e+01
[KERNEL]::measure_buf[i=5] = 2.407520e+01
[KERNEL]::measure_buf[i=6] = 3.695880e+01
[KERNEL]::measure_buf[i=7] = 3.059730e+01
[KERNEL]::measure_buf[i=8] = 2.984860e+01
[KERNEL]::measure_buf[i=9] = 2.864230e+01
[KERNEL]::measure_buf[i=10] = 3.220920e+01
[KERNEL]::measure_buf[i=11] = 1.918210e+01
[KERNEL]::measure_buf[i=12] = 2.494510e+01
[KERNEL]::measure_buf[i=13] = 2.594710e+01
[KERNEL]::measure_buf[i=14] = 2.607140e+01
[KERNEL]::measure_buf[i=15] = 3.532480e+01
[KERNEL]::ResSimulation[j=0]->SimulResSpc[0].speacsSimul_value = 1.725490e+01
[KERNEL]::ResSimulation[j=1]->SimulResSpc[0].speacsSimul_value = 3.412470e+01
[KERNEL]::ResSimulation[j=2]->SimulResSpc[0].speacsSimul_value = 2.807590e+01
[KERNEL]::ResSimulation[j=3]->SimulResSpc[0].speacsSimul_value = 3.778740e+01
[KERNEL]::ResSimulation[j=4]->SimulResSpc[0].speacsSimul_value = 2.744980e+01
[KERNEL]::ResSimulation[j=5]->SimulResSpc[0].speacsSimul_value = 2.407520e+01
[KERNEL]::ResSimulation[j=6]->SimulResSpc[0].speacsSimul_value = 3.695880e+01
[KERNEL]::ResSimulation[j=7]->SimulResSpc[0].speacsSimul_value = 3.059730e+01
[KERNEL]::ResSimulation[j=8]->SimulResSpc[0].speacsSimul_value = 2.984860e+01
[KERNEL]::ResSimulation[j=9]->SimulResSpc[0].speacsSimul_value = 2.864230e+01
[KERNEL]::ResSimulation[j=10]->SimulResSpc[0].speacsSimul_value = 3.220920e+01
[KERNEL]::ResSimulation[j=11]->SimulResSpc[0].speacsSimul_value = 1.918210e+01
[KERNEL]::ResSimulation[j=12]->SimulResSpc[0].speacsSimul_value = 2.494510e+01
[KERNEL]::ResSimulation[j=13]->SimulResSpc[0].speacsSimul_value = 2.594710e+01
[KERNEL]::ResSimulation[j=14]->SimulResSpc[0].speacsSimul_value = 2.607140e+01
[KERNEL]::ResSimulation[j=15]->SimulResSpc[0].speacsSimul_value = 3.532480e+01
num_call_sim = 1
num_measures= 16
begin= 0
end= 16
[KERNEL]::measure_buf[i=0] = 3.778740e+01
[KERNEL]::measure_buf[i=1] = 3.695880e+01
```

Figure 4-16 Communication Data between Master Controler and Kernel

4.4.3 Boot Module

The boot module is responsible for processing the optimization kernel input parameters, upload the optimization kernel input files and initialize the system main data structure. The boot module functions are placed in source code repository, in files “init.c” and “init.h”.

The function that process the optimization kernel input arguments will report an error to the standard output if the mandatory two input files are not provided. Otherwise it will call the functions that read and process these files, the algorithm configuration file, and the parameters files. This function also allows the system to set the variables that defines what kind of optimization process is going to be performed according with the input arguments, by setting the variables that determines if the mutation and mating process is to be standard or with the fuzzy model guidance, and if is to use the electric simulator as the evaluation engine.

After processing the input arguments, the input files, and the algorithm configuration variables are all set, the maximum number of generations to run, the initial population size, the number of optimization variables called genes and the respective genes maximum values etc… It will be possible to do the memory allocation for all the data structures. It was decided to place all functions that handle the data structure memory allocation (i.e. population data structure, specifications and constraint data structure …), and initialization in this module, although they are called in other modules.

All the algorithm population genes are initialized in this module in function “init_pop_rand(Chromossome ** population)”. This initialization uses uniform random values, according with the minimum value, maximum value and the step value defined in the input file parameters for each gene, to guarantee that the random genes values are between a defined range, and has a uniform distribution with a given step. The formula that generates these values is defined as:

$$R = \left\lfloor \frac{Gmax - Gmin}{step} \right\rfloor * rand * step \quad (2)$$

Where this equation is implemented directly with this line of C code:

```
rand_result = floor( ((gene_par[j].max - gene_par[j].min )/step ) *rnd ) * step;
```

4.4.4 Additional Utilities and Random Library Module

This small module was used to place all the system encapsulated common generic functions, with its respective error handling. Here can be found generic functions to open files, print the population’s structure giving all the details for the chromosomes status, the function to retrieve the cost function for each individual in the population, write the kernel menu in the standard output etc… This module in the source code is denominated as “util.c” and “util.h”.

The **random number generator** module is separated in files “randomlib.c” and “randomlib.h” this used approach appeared in "Toward a Universal Random Number Generator" by George Marsaglia and Arif Zaman from Florida State University Report: FSU-SCRI-87-50. It was later modified by F. James and published in "A Review of Pseudo- random Number Generators". This is claimed to be the best known random number generator available. It passes ALL of the tests for random number generators and has a period of 2^144, is completely portable (gives bit identical results on all machines with at least 24-bit mantissas in the floating point representation). This module provides functions that enables to generate random integer value in a given range, random float value etc…

4.4.5 Genetic Algorithm Operators

Genetic Algorithms can be viewed as a general-purpose search method, optimization method, or a learning mechanism. Genetic Algorithms maintain a set of candidate solutions. The set is called population and candidate solutions for the target problem are called individuals or chromosomes. In GA, an individual is often represented by a fixed-length string of genes, where each gene is a small part of a candidate solution. The model proposed will represent real values into the genes of each individual. Their basic mechanisms are similar of biological evolution: reproduction between individuals, mutation of genes through a selected individual and “survival of the fittest”.

The behaviour for evolution of the population, in the standard GA applied to analog circuits/systems optimization proposed, can be described as follows:

1. Initialize a random population of chromosomes (population size = n).
2. Evaluate the fitness of each individual in the population.
3. If the stop condition is satisfied, stop and return the best individual in the population. If not, continue.
4. Select the best for example n/2 pairs of individuals from the population. Create the new remaining n/2 individuals through the crossover operation and replace the worst n/2 individuals in the population. Note that, individuals can be select several times.
5. Apply the mutation operator to a small random selected individual. Replace the old individual by the new one. All individuals can be select and be subject to mutation, even the best individual in population.
6. Go to Step 2.

In the next section it will be explained the implemented modified genetic fuzzy optimization kernel.

4.4.6 Main Module

This section describes the Main module process developed for the system. Figure 4-17 illustrates the principal interactions and in this module all the main function of each internal module are called and integrated following the implemented genetic – fuzzy algorithm approach.

It has been adopted a use of a debug flag definition to determine the lines of code that is to run always or to be run only when is required more debug information. For example the “Show_Pop” is the function that prints to PopulationData.txt file, all the Population data details with current status of all the chromosomes in the population and it’s used as shown in this example:

```
#ifdef DEBUG

fprintf(fp_status_pop,"**********Welcome to EVOLUTION initial population without COST********\n");

Show_Pop(fp_status_pop, population );

#endif
```

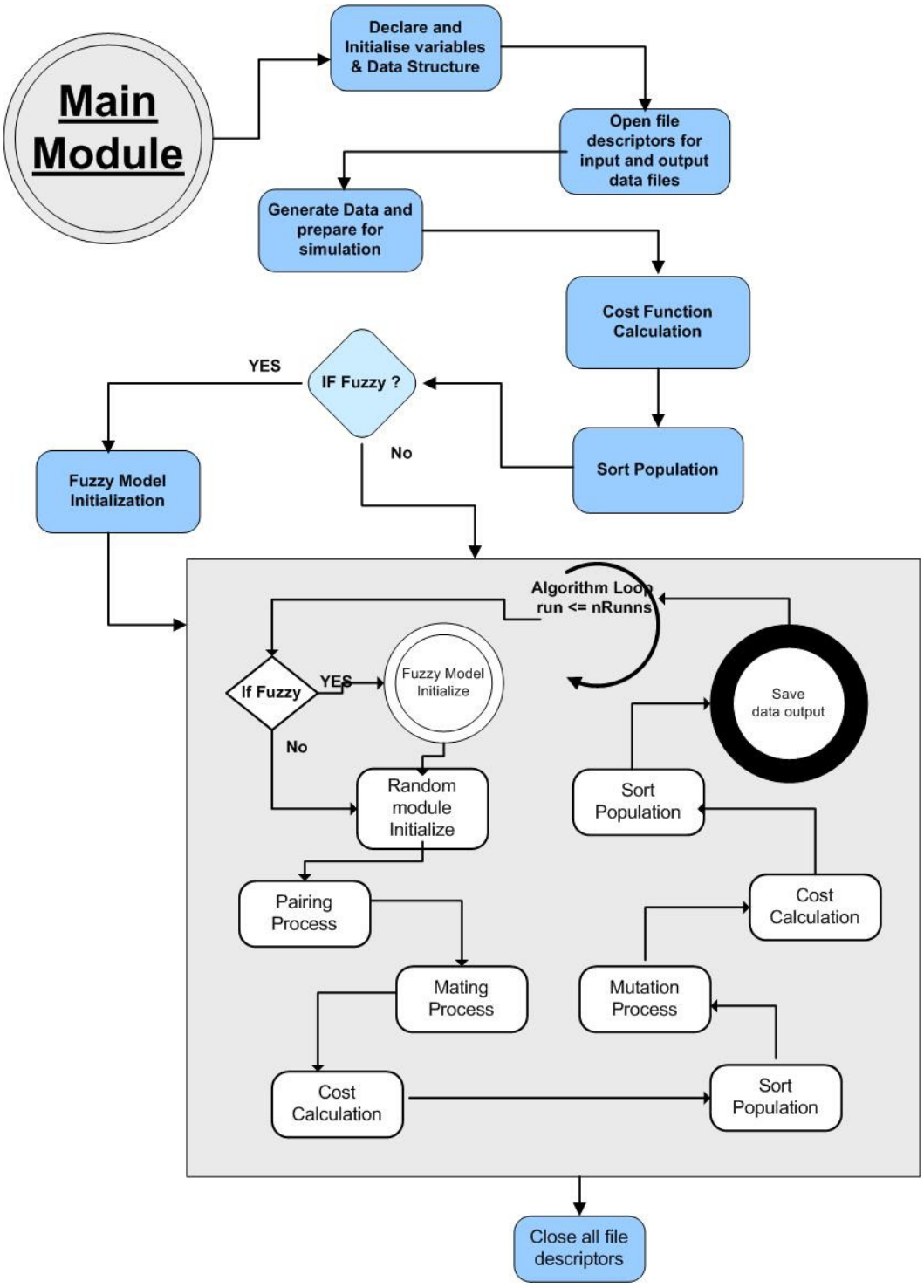


Figure 4-17 Main Module

The first main step is the declaration and initialization phase, initialization of variables in use, for example the fuzzy buffer to store the values before insert in Fuzzy module functions, statistic data for number of time is called the electric simulator etc… Some file descriptors are open at the beginning also, the file descriptors for PopulationData,txt file, the file descriptor for the ComunicationKernelMaster.txt output file, for DEBUG_FUZZY,txt output and some other filtered result from the population data.

In this phase the boot module functions are called the configuration input files and others as explained before in boot module section are uploaded and all the main internal data structure (population, constraints, specification) are allocated and initialized.

The “keep” variable value is the number of top best individual that survives for mating and the bottom PoppulationNumber – Keep are discarded to make room for the new offspring, equation 3 and 4 are performed at this phase in the main module. Letting only a few chromosomes survive to the next generation limits the available genes in the offspring. Keeping too many chromosomes allows bad performers a chance to contribute theirs traits to the next generation. For this reasons is often keep 50% ($Selection_{Rate}$ =0.5) in the natural selection process. The equation input values are set by the boot module.

$$Keep = \lceil Selection_{Rate} * Pop_{Initial} \rceil \quad (3)$$

$$Pairs = \left\lceil \frac{Keep}{2} \right\rceil \quad (4)$$

The next step is the cost function calculation for the entire population, so first if is the kernel simulation mode is called the function “DumpGenesSimulation (population, Pop_inicial, ALLPOPULATION)“ that is placed in module “cadence_interaction” and is responsible to generate the file InputParameter.scs necessary to the electric simulation, this file has the population genes to be simulated as explained before. When the data are all set, the Cost function is called from module “Cost” to compute the chromosome fitness and related data. After this step a debug section is included that allows to see the population after this phase, this is helpful during development and test phase.

The chromosomes after cost calculation are sorted, so the Sort_Population function from “Sort” module is called. The sort method will be discussed later, so at this point in the main module only the main function from sort module is called and the decision of each method to use is taken inside the sort module according with the configuration input values already uploaded. Also after this step debug lines are include supervising the population.

A check to the global variable Mutation_Mode is made, this value is set in the boot module, and if it has a true value the Fuzzy_init function is called to initialize the model fuzzy. The Fuzzy module is a kernel external module, and is included in this main so its functions are visible.

After all the initialization phase, the algorithm loop begins, and the stop criteria is the number of generation reaches the maximum defined in the configuration file, or the user commands to stop the simulation.

Inside the algorithm cycle search for the optimum solution, there are several steps performed:

- If is to use fuzzy model, function Fuzzy_interface is called. And the debug fuzzy file is updated if the debug mode is enabled.

- Next the init_Random(runn) is called to initialize the random number generator and the seed used is the current runn number.

- Memory allocation is done for parents to be used in the selection process, this function is from init module, and after the Rand_Pairing(mother_father); is called from “paring” module. The first approximation used is random pairing, but more methods can be tested. There is a debug option to show the generated parents.

- After the paring process is concluded, the mating process starts, and the parents are extracted from the population and the offspring are generated and placed in the population

as will be explained in the mating section. Also after this step, if the debug option is enabled, it is possible to see the population after mating with the new offspring. The memory free for the allocated parents is done after this step.

- The dump genes function is called now with a different behaviour it will only dump the new offspring to minimize the number of electric simulator call that slows down the process when kernel is in simulation mode. After the cost function is called to compute the fitness for the population new individuals. A new population sort is performed, and the debug when enabled the Show_pop function updates the PopulationOutputData.txt file.

- Mutation function happens now, where the only function called in the main is mutation(fp_status_pop, population); and inside mutation module the decision of which method to use is taken. This process is performed. If debug is enable the population after mutation is uploaded to the output files, and also the model fuzzy debug information and status are updated.

- After mutation the Cost function module is called the chromosomes mutated are updated, and a new sort in the population is made.

- The final step the best chromosome with it's details are extracted from the population, and saved in output files to be represented graphically, the fitness evolution and the genes value evolution.

After the stop criterion is met the loop ends and all the initially open file descriptors are closed.

4.4.7 **Sort Techniques**

Two sort techniques were implemented in this system. The well known Quick Sort and a promising sort algorithm found in the literature. Depending on the user choice, expressed in the configuration file, it's used one method or the other, and the results can be compared to see which one shows better performance. Some examples are shown in this section, along with the detailed explanation about the implemented techniques.

Quicksort is a sorting algorithm that on average, makes O(nlogn) (big O notation) comparisons to sort n items. In the worst case, it makes O(n2) comparisons, though this behaviour is usually rare. Quicksort is often faster in practice than other O(nlogn) algorithms[wikipedia.org]. Quicksort can be implemented as an in-place sort, requiring only O(log n) additional space. Quicksort (also known as "partition-exchange sort") is a comparison sort and, in space efficient implementations, is not a stable sort. The implement quick sort in this module was made encapsulating the qsort() function from <stdlib.h> library. An additional function was necessary that compares two chromosome elements, using the fitness value as the comparison metric.

```
The comparison function must return an integer less than, equal to, or
     greater than zero if the first argument is considered to be
respectively
     less than, equal to, or greater than the second.
```

Compare routine: **compare** (const void* individ_A,const void* individ_B)

if ((*(Chromossome**)individ_A)->fitness < (*(Chromossome**)individ_B)->fitness){

```
        return -1;

} else if ( (*(Chromossome**)individ_A)->fitness == (*(Chromossome**)individ_B)->fitness ){

        return 0;

} else{

        return 1;

}
```

The second approach proposed in A Memetic Approach to the Automatic Synthesis of High Performance Analog Integrated Circuits [33] was implemented and tested. It proves to achieve remarkable results.
The algorithm is described here as:

Given two candidates in the population, there may be, at most, three situations:
(1) Both solutions are feasible;
(2) Both solutions are infeasible;
(3) One solution is feasible, but the other is not.

Accordingly, the selection rules are:
(1) Given two feasible solutions, select the one with a better objective function value;
(2) Given two infeasible solutions, select the solution with smaller constraint violation;
(3) If one solution is feasible and the other is not, select the feasible solution.

The drawback of this technique is that some candidates with very good performances are missing in the search process. To illustrate and understand this phenomenon, and why this technique was adopted, let us consider the performance space in Fig. 5.4-15. The region in blue is the feasible region. We can see that the left point is infeasible, while the middle and the right points are feasible. Suppose the global optimum is the middle point. According to the method described above, the right point will be selected when it competes with the left point. However, the fact is that the left point may be more useful than the right one, because it is much closer to the optimal point than the right one. In order to protect solutions such as the left point, and combined with one-to-one selection in differential evolution (DE), the above rule is modified by following a methodology inspired on simulated annealing.[33]

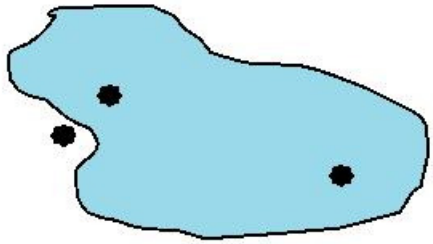


Figure 4-18 Illustrative solution space.[33]

For a candidate x and its child x', suppose one is feasible and the other is not. If the violation of constraints of the infeasible solution is less than δ and it has a better objective function value, then it is accepted with a non-zero probability. This is analogous to a simulated annealing SA procedure: if the child is worse than the parent, SA does not reject it, but it is accepted with a probability. This DELTA δ value is inserted by the user in the configuration input file, described as the parameter that improves the sort accuracy. In the kernel the implemented function can be found in module sort defined as "EvolSort_Population(Chromossome ** pop, int start, int end)".

When the Sort_Population function is called in the main module, the sort algorithm used will be the one selected by the user expressed in the variable Sort_Mode that is 0 for Qsort_Population function and 1 for EvolSort_Population.

4.4.8 **Pairing**

The main function in this Pairing module: Finds a list of the two parents randomly and it's defined as: **Rand_Pairing**(Parents *mom_dad). For the pairing strategy, the selection of the potential mate is chosen from the list of candidates that better complement the already selected parent in terms of satisfied constraints. Mating the parent with the one that better fulfills the faulty constraints of both parents can potentially increase the probability of achieving a child that satisfies more constraints than the parents do. Figure 4-19 shows the algorithm flow. When the variable repeat in the Figure 4-19 is set to 1 at the end, the cycle continues until all the pairs are found.

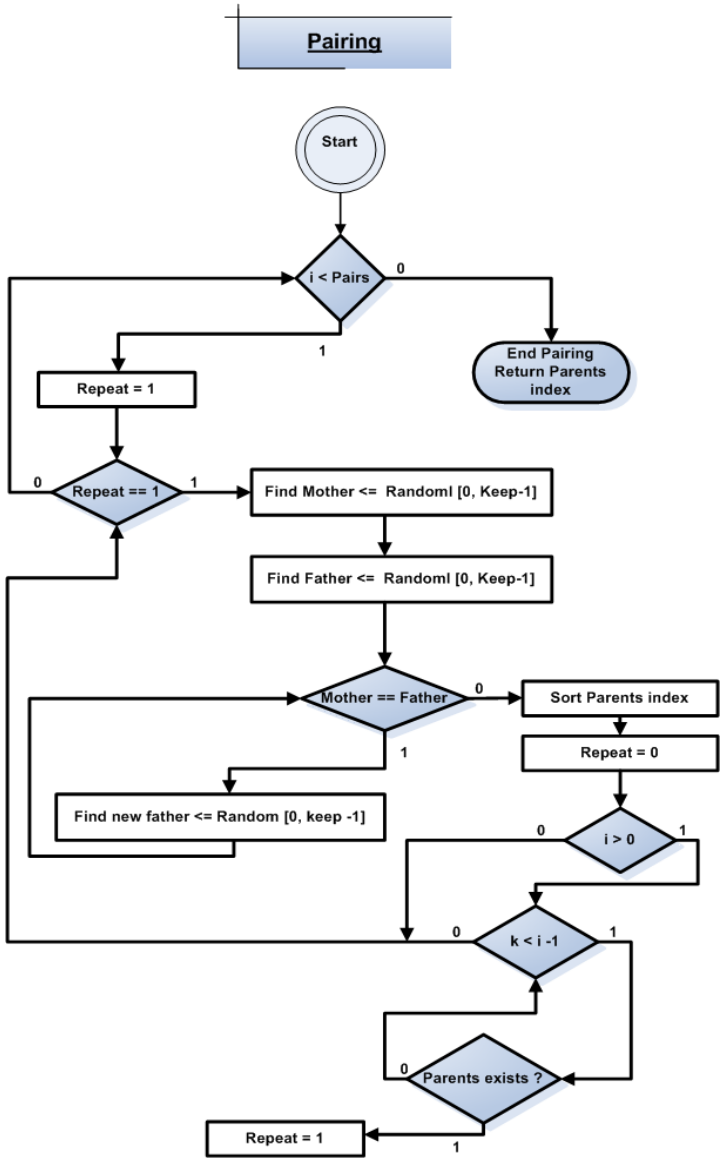


Figure 4-19 Paring implemented algorithm

4.4.9 **Mating**

Mating is the creation of one or more offspring from the parents selected in the pairing process. This module has the process that generates 2 offspring, at a time for each pairs of parents. It includes the new offspring in the population, and returns the population with best keep individual and theirs offspring.

The Figure 4-20 illustrates the implemented standard mating algorithm where in this approach involves two parents that produce two offspring. Many different approaches have been tried for

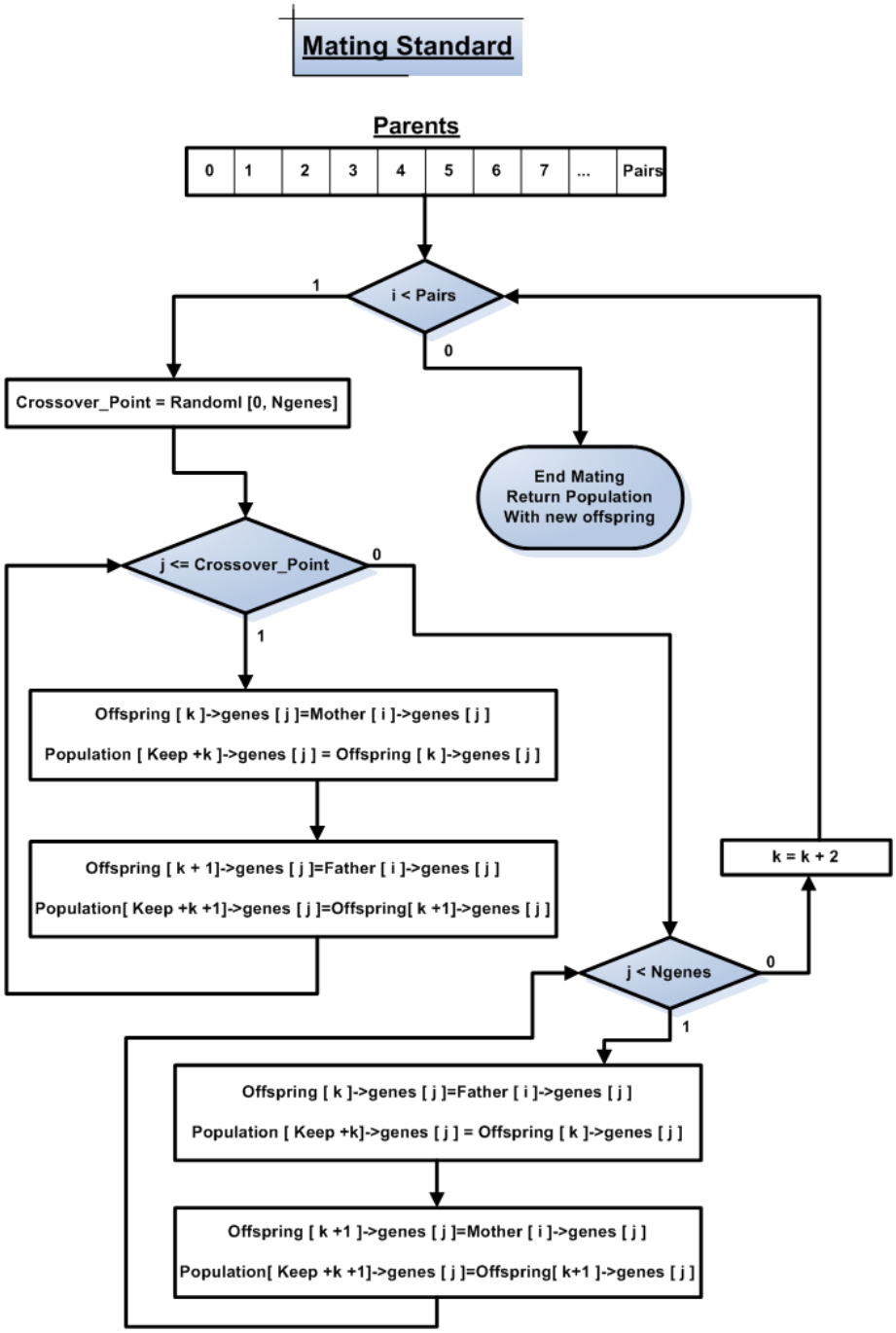


Figure 4-20 Mating Process

crossing over in GAs. The standard strategy implemented chooses one point in the chromosome to mark as the crossover points. Then the variables on both sides of this point are swapped between the two parents. This kernel has the structure and function to select the fuzzy guided mating process, MatingFuzzyValues(pop, pa, fuzzy_data); and the desired approach to be used during the optimization process is selected according with the Mating_Mode configured value.

4.4.10 **Mutation**

This section describes in detail the implemented mutation module. If no mutation process exists, or even another strategy, the GA can converge too quickly into one region of the cost surface. If this area is in the region of the global minimum, that is good solution. However, some functions have many local minima. If we do nothing to solve this tendency to converge quickly, we could end up in a local rather than a global minimum. To avoid this problem of overly fast convergence, we force the routine to explore other areas of the cost surface by randomly introducing changes, or mutations, in some of the variables. Mutation is one of the primary methods to maintaining diversity among feasible solutions.

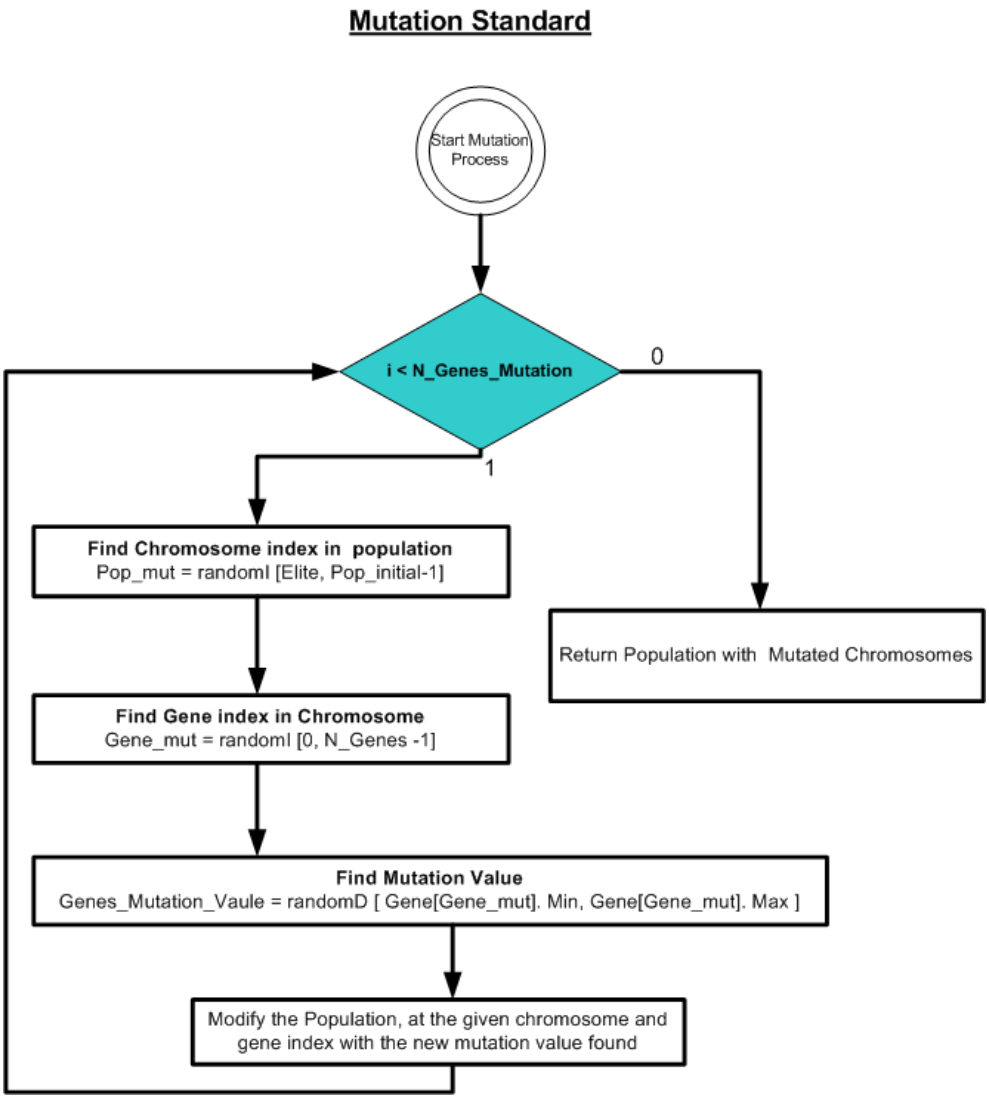


Figure 4-21 Standard Mutation

Mutation is an operator that acts on a single individual at a time. Unlike crossover, which recombines genetic material between two parents, mutation replaces the value of a gene with a randomly-generated value. Similarly to crossover, mutation is usually a blind operator, in the sense that the gene to be mutated and its new value are randomly chosen without any attempt to minimize the fitness of the new individual in the standard approach. In order to escape from a local minimum, a kind of jump operation is needed. So, by using the mutation operator, we can get some individuals different then they were and exploit more efficiently in the entire output domain.

Both mutation approach are implemented and placed in "mutation.c" and "mutation.h" source code. The approach in use is selected according with Mutation_Mode configured value.

For the mutation guided with fuzzy system model, it was implemented the function MutationFuzzy(FILE* fp, Chromossome **pop), where the mutation value to be set in the chosen chromosome genes to be mutated is the value indicated by "Fuzzy_interface(population[pop_mut]->genes, gene_mut)" function, from module Model Fuzzy.

The fuzzy model introduces a level of knowledge in the GA kernel at mutation level, preventing it to be a complete blind process, allowing a guidance based on the fuzzy model output generated according with circuit under optimization equations. Controlling parameters of genetic algorithms with fuzzy systems and evaluate the population with objective of achieve the solution more efficiently. In the proposed method the fuzzy logic will be performing only in the mutation value, for chosen chromosomes. As we said before, the mutation replaces the value of a gene with a randomly-generated value between the domains defined to input variables. But sometimes, those changes will increase the fitness of individuals, especial if we are talking about a large domain of input variables. With fuzzy logic we would control the attribution of the new gene randomly-generated value and indicates a better orientation for the new value gene.

So the difference between the standard mutation strategy and the fuzzy mutation strategy is that the MutationFuzzy function instead of a random number for the mutation value, it uses a value returned by the fuzzy model. It returns this value, given the population, the index of the chromosome to be mutated pop_mut and the index gene_mut of the gene to be mutated, these index are randomly chosen, the same as in standard mutation.

The call function is made in the following line of code:

Mutation_values_Fuzzy = Fuzzy_interface(population[pop_mut]->genes, gene_mut);

Figure 4-21 illustrates the cycle performed in the mutation process. The cycle is controlled by N_Genes_Mutation. This variable is obtained following equations (5) and (6).

$$Mutated_{pop} = Pop_{initial} - ELITE \quad (5)$$

$$\mathrm{N_{Genes_{Mutation}}} = \lceil \mathrm{Mutated_{Pop}} * \mathrm{Mutation_{Rate}} * \mathrm{NumVarGenes} \rceil \quad (6)$$

The equations arguments are: $Pop_{Initial}$ is the total number of individuals in the population, the ELITE is the number of the best chromosome in the population not to suffer mutation and its defined in the configuration file. The $Mutation_{Rate}$ is defined in input configuration file, and finally the NumVarGenes is the number of genes in a given chromosome.

4.4.11 **Evaluation Engine**

The principal evaluation engine for the simulation based approach used in this system is SPECTRE. The modules involved in the evaluation engine process are: cost module (files cost.c and cost.h), specification module (files specification.c specification.h), constraint module (files constraint.c and constraint.h), the cadence interaction module (files cadence_interation.c

and cadence_interaction.h), and Master.py. In this section the role of each one of this module, is going to be explained in order to understand the implemented solution.

For the equation based approach some examples were made, given the fact that the principal objectives of this thesis is to focus on the evaluation engine using the simulation based approach. The module involved in this process is equation module (files equation.c and equeation.h), constraint module, specification and cost module.

The synthesis tool handles the Analog circuit design as a constraint optimization problem. In this equation (7), the function f (x) is the objective to be minimized (or maximized) and h(x) is equality constraints, and g(x) corresponds to user-defined specifications, delimiting the feasible region.

$$\begin{aligned} &\min_x f(x) \\ &\quad g(x) \geq 0 \\ \text{Subject to} \quad &\quad h(x) = 0 \\ &\quad X_L < x < X_H \end{aligned} \tag{7}$$

The constrained optimization problem is transformed into an unconstrained one by minimizing the following function:

$$f'(x) = f(x) + \sum_{i=1}^{n} w_i \langle g_i(x) \rangle \tag{8}$$

The parameters w_i are the penalty coefficients.
The implemented cost function equation that evaluates a given solution is given by the generic expression (8) and in particular case expressed by equation (9) where $w_i = 1$, representing the sum of the normalized distances to each performance goal. G_{xexp} is the simulated value and G_{yth} represents the goal value.

$$Cost = \left|\frac{G_{0\text{exp}} - G_{0th}}{G_{0th}}\right| + \left|\frac{G_{1\text{exp}} - G_{1th}}{G_{1th}}\right| + \cdots + \left|\frac{G_{Popth} - G_{Popth}}{G_{Popth}}\right| \tag{9}$$

$$Cost(x) = \sum_{j=0}^{N\ SPECS} wp_j * Fit_j(x) + \sum_{k=0}^{N\ CONTR} wc_k * Cosnt_k(x) \tag{10}$$

$$Fit_j(x) = \frac{|P_j - goal(j)|}{goal(j)} \tag{11}$$

$$Const_k(x) = \frac{|C_k - goal(k)|}{goal(k)} \tag{12}$$

Equation 10, 11 and 12, illustrates the mathematical formulation behind the implemented cost function. In equation 11 the $Fit_j(x)$ is a set of normalized objective functions derived from performance specifications to be optimized, implemented in specification module. In equation 12 the $Const_k(x)$ is a set of normalized user defined penalty terms acting as constraint function, implemented in constraint module. In these two last equations j is the number of objective functions, and k is the number of constraint functions. These are used to translate the achieved design constraints and specifications in a cost function. The designer setups the relative importance, for each competing specification, adjusting the individual scalar weights wp_j and wc_k.

The design goal is usually to minimize/maximize one objective function (e.g., minimize power consumption), subject to some constraints (e.g., slew rate larger than a certain value). In this system the Cost is the objective function. The synthesis is achieved by embedding a performance evaluator within an iterative optimization procedure. Simulation-based method, rely on electrical simulations to evaluate the circuit performances and the simulation-based methods yield superior accuracy, generality, and ease of use, compared with equation based.

The implemented approach calls function: Cost_Function(Chromossome ** pop, int NumPop, CostType_e cost_type); This method given a population, it calculates the fitness value, for a set of individual. A swhich allows us to select the kernel mode, based on the configured KERNEL_MODE variable set in the configuration file.

case 0: EquationCostFunction(pop);
case 1: SimCostFunction(pop, NumPop, cost_type)

4.4.11.1 Equation Based Approach

In this mode, given a population the equation module calculates the fitness value for every individual in the population.

The module equation has the functions for handling the equation base approach. The main function is the ObjectiveFunction that is a swhich with some tested BENCHMARK circuit directly implemented, and generic function to be optimized, and test the algorithm performance. This objective function returns what we call the fitness, when it's given as an input one chromosome.

Here it's given some implemented and tested benchmark examples:

1. Low Pass filter equation
   /*fitness function for the low pass filter*/

   objective_function = abs(res_specs[0] - A0)/A0 + abs(res_specs[1] - F0)/F0 ;

2. objective_function= AlotLocalMinimun(X); Represented in Figure 4-22.

```
/*
 * This function calculate one of the benchmark
 * function that its also presented
 * with a lot of local minimum a good challenge
 */

z1 = sqrt(1 + pow(X[0],2) + pow(X[1],2));
z2 = sqrt(1 + pow((X[0]-1),2) + pow((X[1]+1),2) );
z3 = sqrt(1 + pow((X[0]+1),2) + pow((X[1]+1),2));
z4 = sqrt(1 + pow((X[0]-1),2) + pow((X[1]-1),2));
z5 = sqrt(1 + pow((X[0]+1),2) + pow((X[1]-1),2));

function = 5 - sin(4*z1)/z1 - sin(2.5*z2)/z2 - sin(3*z3)/z3 - sin(2*z4)/z4 -sin(4*z5)/z5;
```

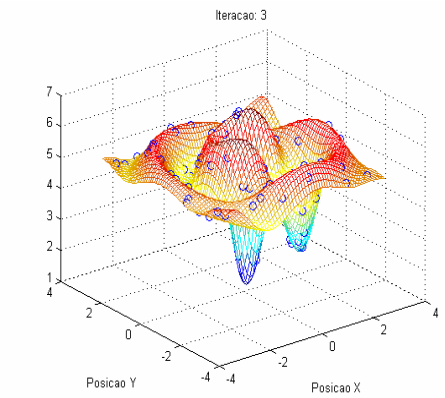


Figure 4-22 Representation of function f(x), and some individuals

The implemented objective function is defined as ObjectiveFunction(double *res_specs, Chromossome *pop). The input argument "res_specs" holds the calculated specification error. The constraint error calculation is made in its correspondent module. The specification calculus is made in module specification by function SpecsEquation(Chromossome *pop). To help on these the calculations process several functions had to be implemented.

These help functions, evaluates if the obtained experimental chromosome variable value are in this following conditions:

- Condition -> f(experimental) >= given_value; bigger then the given goal – function CheckBiger(double experimental, double given_value);
- Condition -> f(experimental) < given_value ; lower then the given goal - function CheckLower(double experimental, double given_value);
- Condition-> min < experimental < max ; between a given range of given goals - getFitBand(double var, double low, double up) and CheckBound(double experimental, double given_max, double given_min);
- Validates if experimental = given value CheckEqual(double experimental, double given_value).

The equation based cost calculations process is performed in Cost module when function EquationCostFunction(Chromossome ** pop) is called. This function given a population, it calculates the fitness value for every individual. The first step is the population data structure

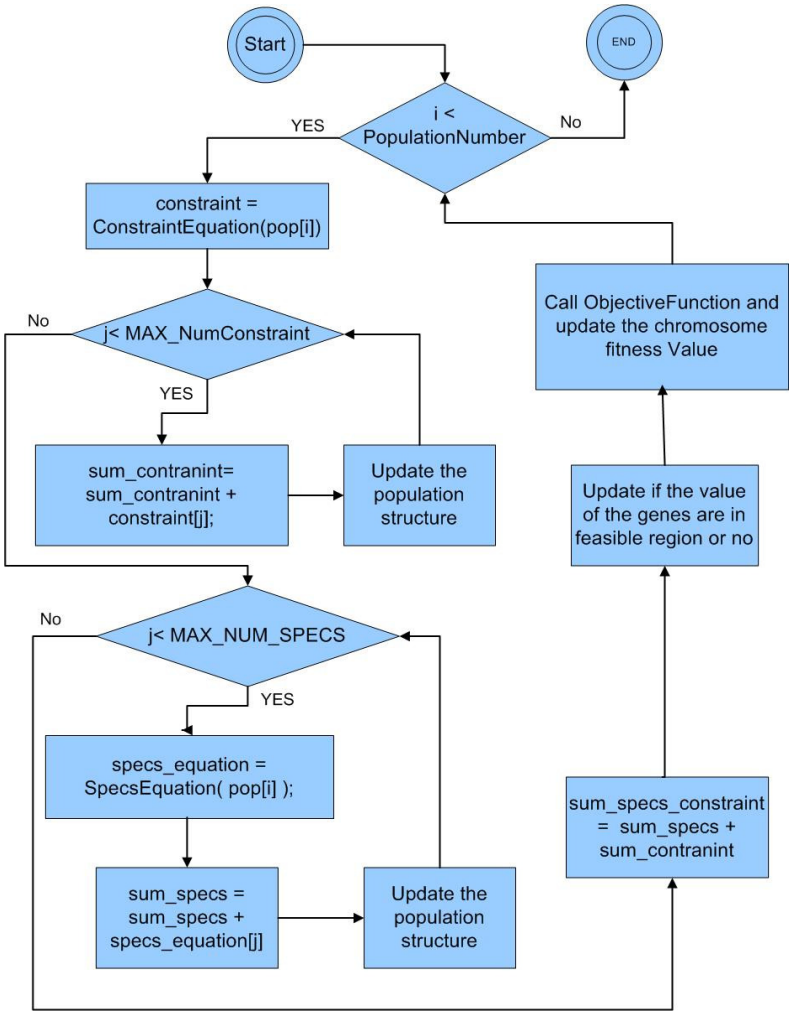


Figure 4-23 Equation Based Cost Calculation

initialization and cleaning, Figure 4-23 shows in detail the equation based cost calculation implemented flow. This cycle iterates over the entire population chromosomes. The Constraint equation and specification equation calculus are performed in their correspondent module.

4.4.11.2 Simulation Based Approach

Figure 4-24 presents a generic view for the simulation based cost function process flow. The details of the internal cycle that iterates over all chromosomes and compute its fitness data,

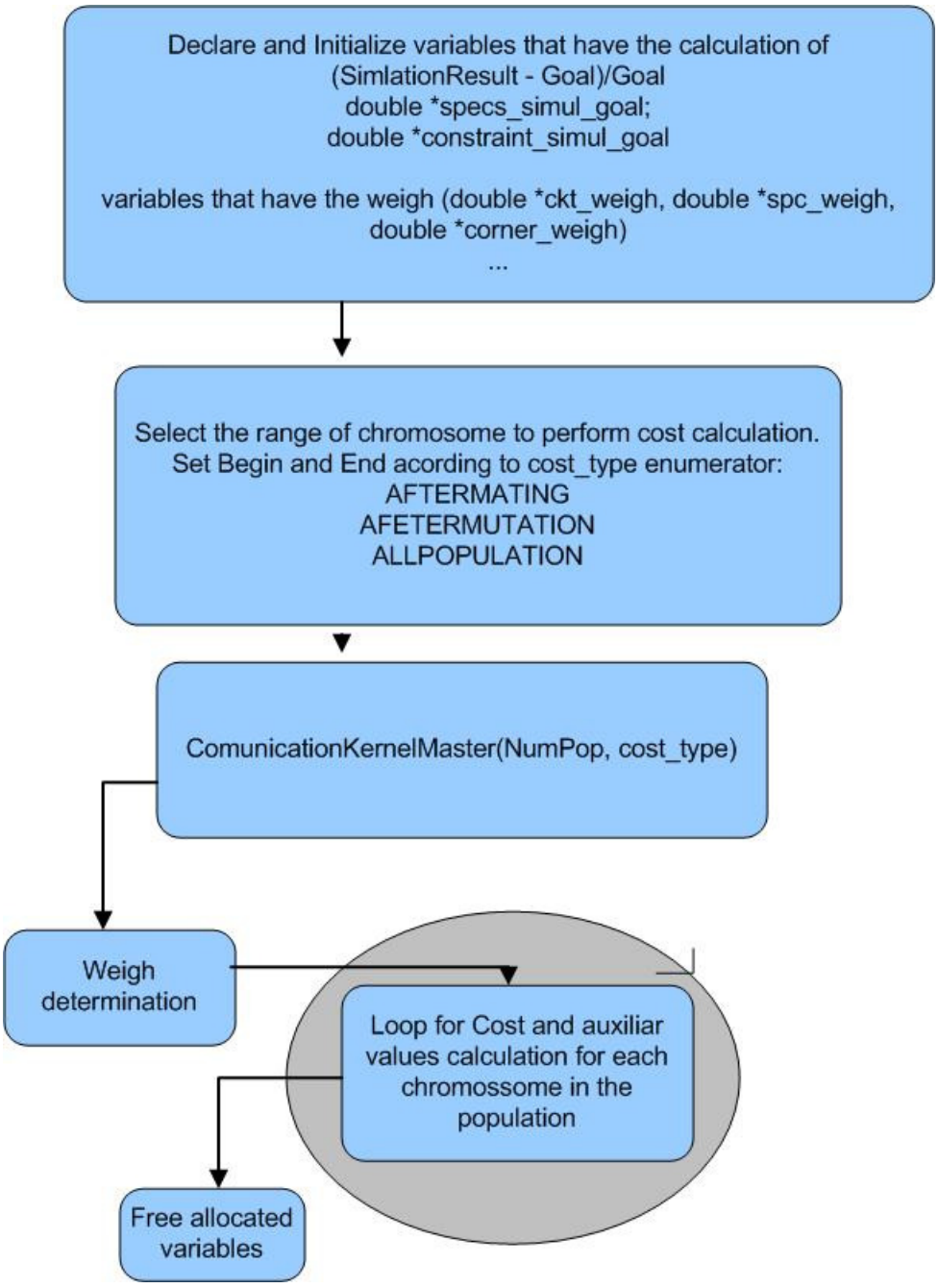


Figure 4-24 Simulation Based Cost Function calculation

specification and constraint error are given in Figure 4-25 to illustrate in a more clear way.

In the simulation based cost function approach a complexity is added to the process flow, due to the communication between the kernel and the electric simulator spectre. It was decided to use a master controller module that its function is to handle this complexity, the communication between the optimization kernel and the electric simulator, the data transfer and the synchronization between these two programs.

The process flow illustrated in Figure 4-25 is what equation 10 formulates.

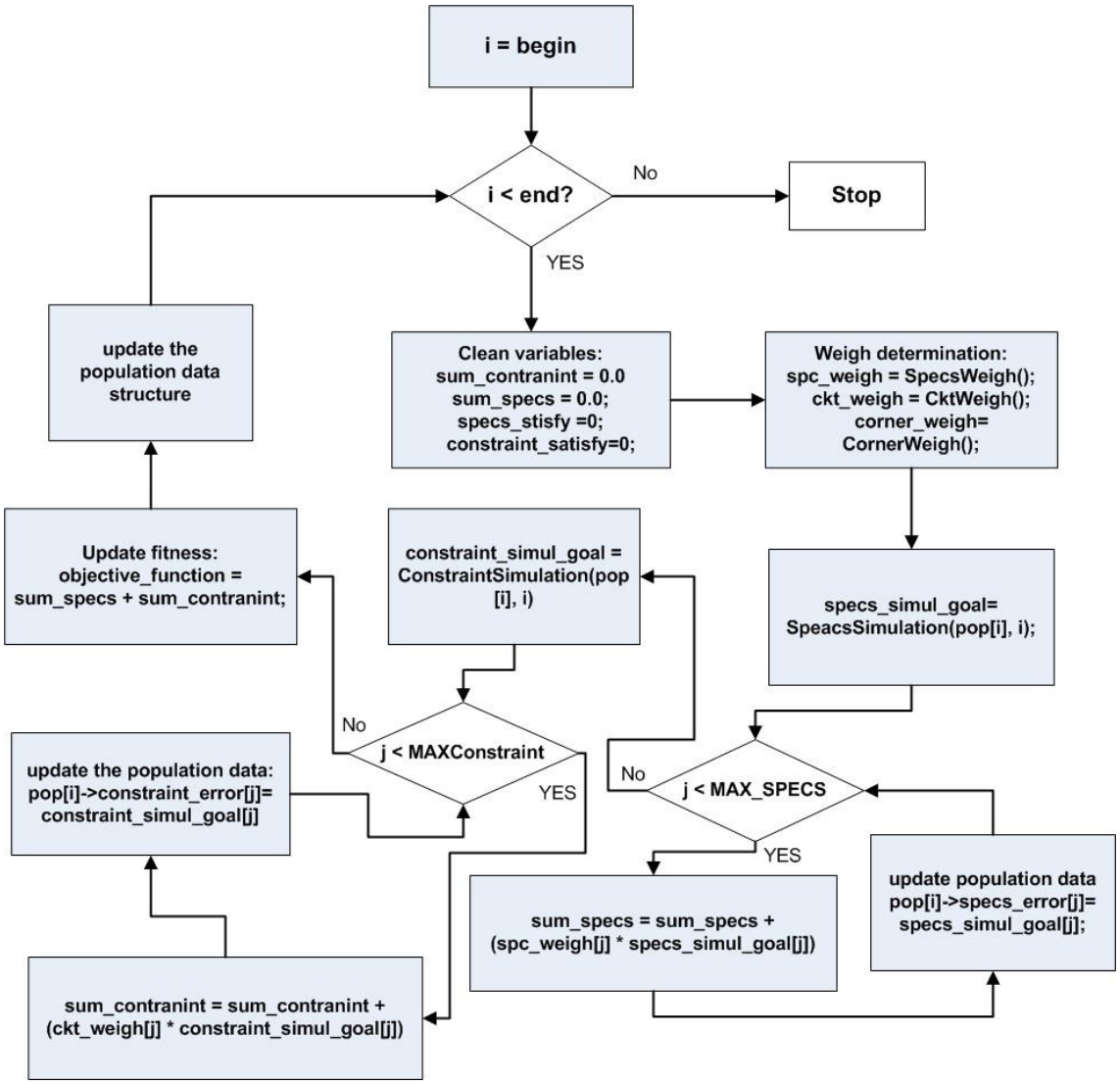


Figure 4-25 Internal cycle for simulation based cost function

In the optimization kernel side the function responsible to handle the communication with the Master controller module is the ComunicationKernelMaster (int NumPop, CostType_e cost_type) function. This function is placed in module cadence_interaction.

The principal steps taken in this function, is illustrated in Figure 4-26. The Master controller is the module that initiates the optimization kernel, and keep open a pipe connection, listening to it's output, and when it receives the begin order it performs the necessary operations to start the electric simulator spectre, given the right input options and files, and after process the spectre outputs files and send back the results to the optimization kernel, it waits for consecutive order to lunch a new electric circuit simulation or to stop the process.

This process is explained in more detail in the following sections. This communication kernel master function, waits for the results send by the master controller and after receiving it continues the optimization process in the kernel side.

4.5 Electrical Circuit Simulator Interaction

The optimization kernel interface to electrical simulator is going to be explained in detail in this section.

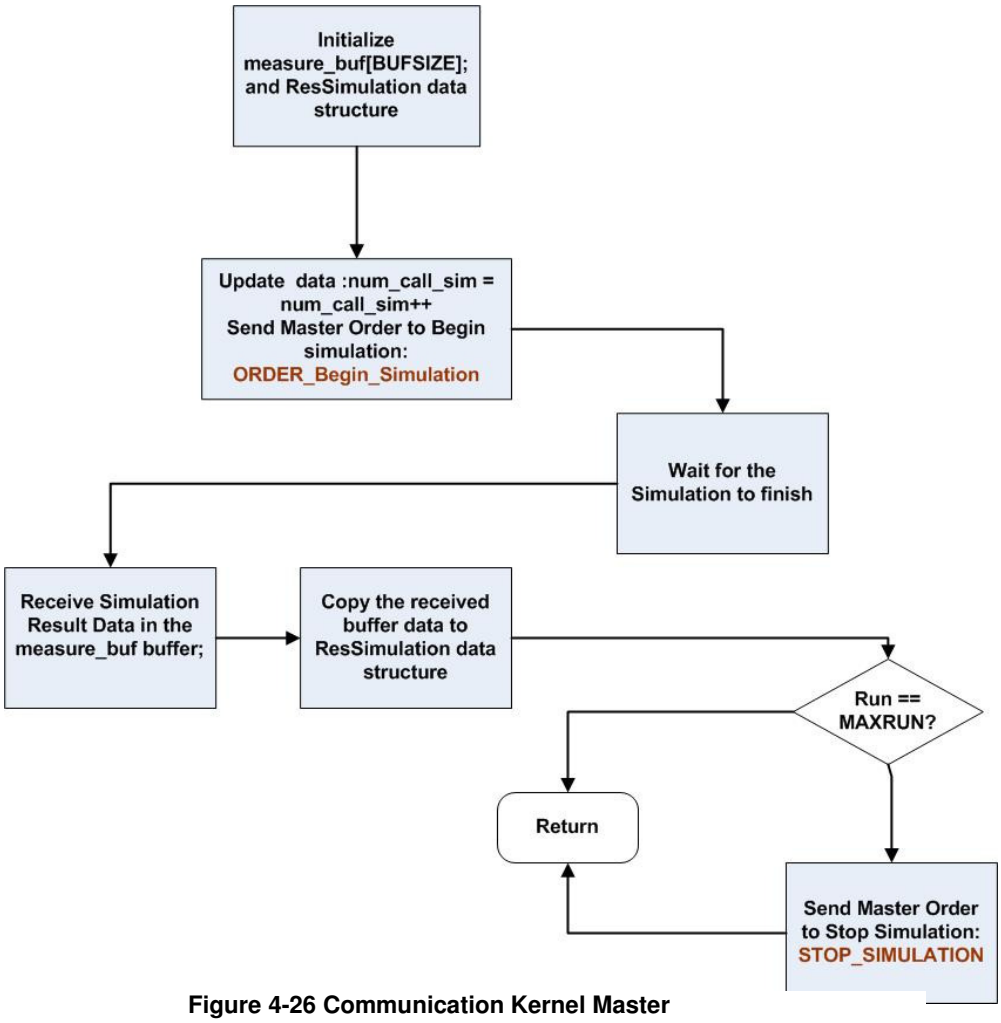


Figure 4-26 Communication Kernel Master

The electric circuit simulator of choice in this system is Spectre circuit simulator from cadence. A modern circuit simulator that uses direct methods to simulate analog and digital circuits at the differential equation level. The basic capabilities of the Spectre circuit simulator are similar in function and application to SPICE, but the Spectre circuit simulator is not descended from SPICE. The Spectre and SPICE simulators use the same basic algorithms—such as implicit integration methods, Newton-Raphson, and direct matrix solution—but every algorithm is newly implemented. The Spectre circuit simulator has many improvements over SPICE. The Spectre circuit simulator can simulate larger circuits than other simulators because its convergence algorithms are effective with large circuits. It has also Improved Accuracy, is designed to improve simulation speed, Improved Reliability etc…[34] Spectre has this feature SpectreMDL (MDL is measurement description language). This feature was used, to simulate the circuits and have the results with a better organization. SpectreMDL uses a control file to facilitate the

making of measurements, performing multiple simulations such as corners analysis; automatically stopping simulations as soon as all necessary data exists for the requested measurenements etc…

The optimization kernel and the model engine, needs to interacts with the electric simulator, spectre. That's why a communication protocol was developed to implement this communication. In order to control, manage the communication, and the data exchange a new module was necessary, the Master module. Figure 4-27 shows a simple and very high level view for the principal big blocks involved in this protocol.

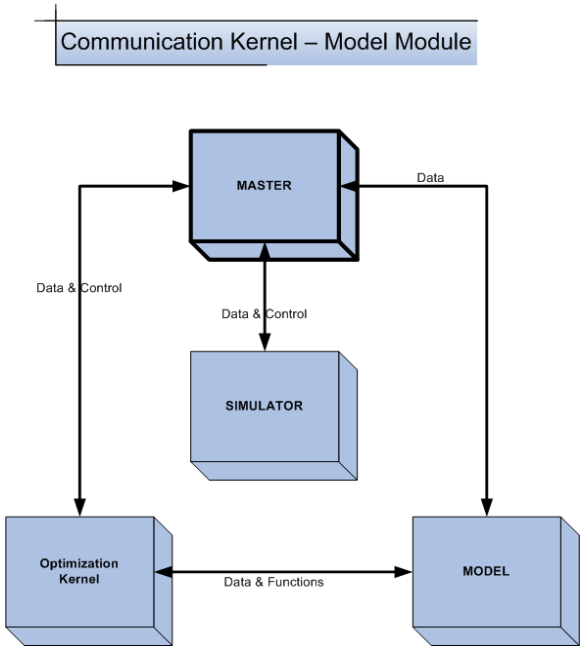


Figure 4-27 Communication with electric simulator

This Figure 4-28 shows the complete internal communication protocol performed in this system. This picture intends to give a top view of how the main elements interact, and the sequence of this interaction.

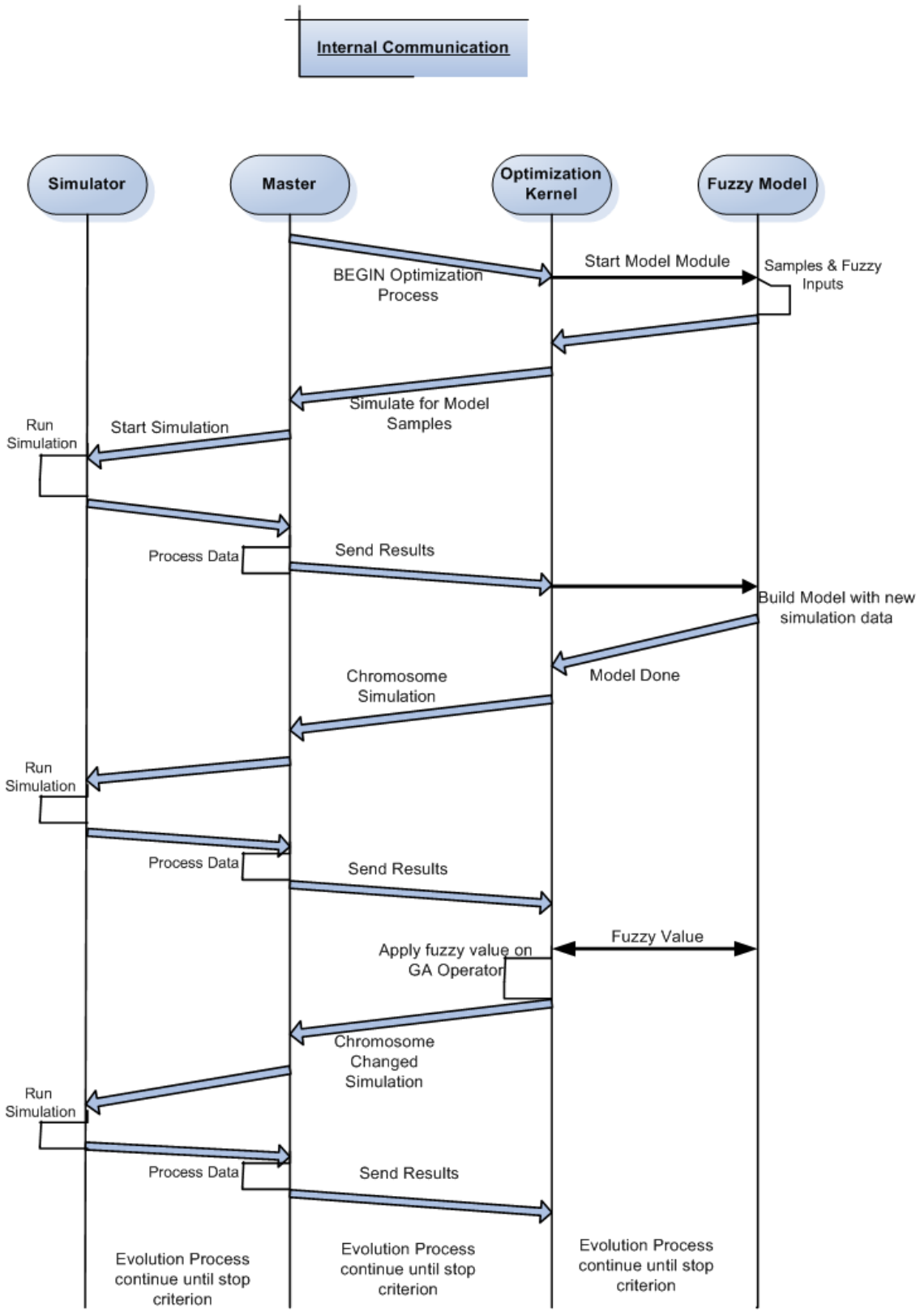


Figure 4-28 Internal Communication

4.5.1 Master Module

The Master Module the principal actor in the communication process between the main modules in the system as explained before was developed using python. This option was taken because it shows considerable advantages for the processing of data with internal regular expression modules “re” module, parsing of text files many library to support, and it provides robust library for multiple process handling, the “subprocess” module. Also it was a choice because for the user interface python has several useful graphical modules to be used like Tkinter, tkMessageBox, tkSimpleDialog etc...

This module lunch the optimization kernel with different input argument, as defined by the user (used python library “os” function popen4). The Master access the spectreMDL necessary inputs files in SpetreSetting folder (illustrated in Figure 4-29) and when it receives the kernel order it lunches the simulation.

After the simulation finishes spectreMDL outputs a file.measure that has the results organized in a standard form, defined in the control file file.mdl, and the function HandleMeasureFile that uses regular expression and an internal implemented algorithm, parses the file, and send the results to the optimization kernel.

When Master lunches the simulator, it uses the option “log” that gives an indication to the simulator to generate an log output file with all the simulation information, and also it uses the input option “batch” for the simulator to run in batch mode.

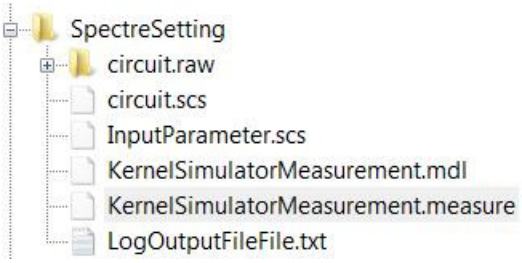


Figure 4-29 Spectre input and output files

4.5.2 Simulation Data

Figure 4-30 is an example of the output file from SpectreMDL after a simulation, with the measures results for the DCgain for 9 chromosomes. The chromosome identifier is defined in the input file *InputParameter.scs*, and the simulator generates this output putting the identifier (_id_) for each given chromosome. These values are parsed by the Master module.

for each given chromosome. These values are parsed by the Master module.

```
Exported variables from PSF results directory:  ./SpectreSetting/circ

date                :  11:36:40 PM, Sat Jan 27, 2011
design              :  // Generated for: spectre
simulator           :  spectre


Swept Measurements :
Measurement Name    :  dcdata
Analysis Type       :  dc
dcgain                  _id_ @ 0
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 1
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 2
                        res0 @ 348.69
                        res1 @ 2.88248      =  41.6524
dcgain                  _id_ @ 3
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 4
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 5
                        res0 @ 288.89
                        res1 @ 4.16394      =  36.8239
dcgain                  _id_ @ 6
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 7
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 8
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
dcgain                  _id_ @ 9
                        res0 @ 288.89
                        res1 @ 2.88248      =  40.0184
```

Figure 4-30 Spectre Simulation output

4.6 Main Interaction during Optimization

This section presents briefly the main interaction during optimization, the principal input and output files and main communications, all illustrated in Figure 4-31.

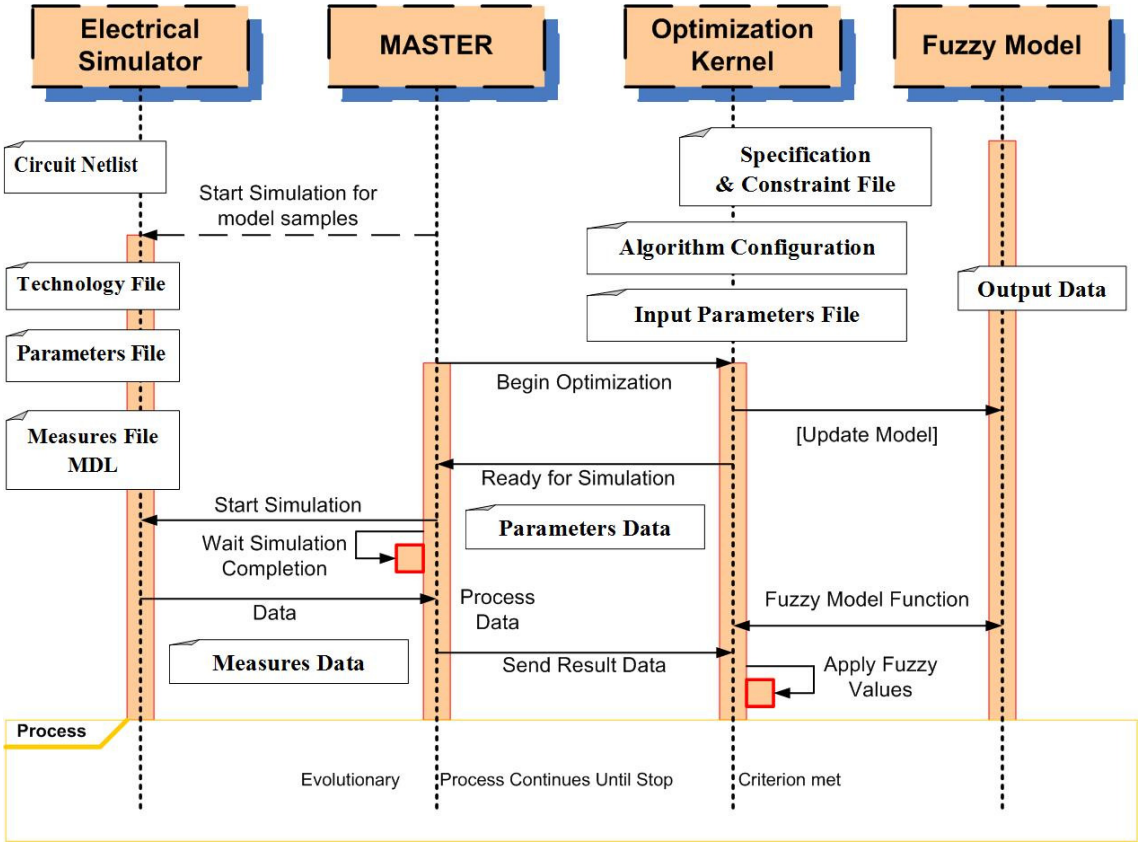


Figure 4-31 System Main Interaction

4.7 Conclusions

This chapter discussed the design architecture, methodology and design implementation of every module in the system. A detailed explanation on the decision made during the system development was given, supported by some mathematical theory and state of the art implementations. Figure 4-32 is a resume of the system, illustrating the main modules, their output and input files.

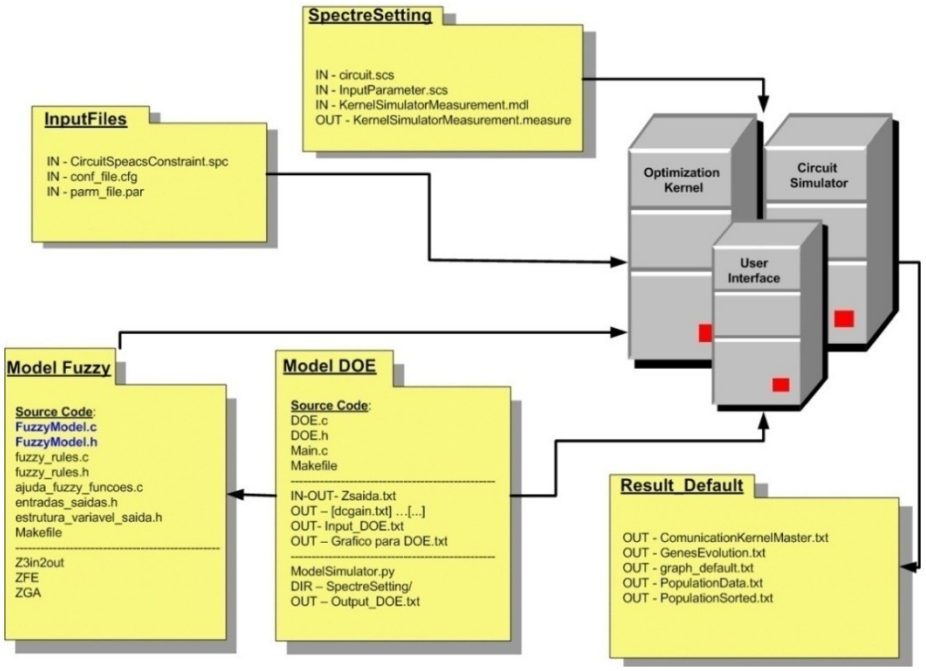


Figure 4-32 System Top View

5 ANALYSIS OF THE OBTAINED TESTS RESULTS

This chapter presents the achieved result to validate the Fuzzy-Genetic optimization kernel approach.

Some case studies of Analog circuit synthesis are discussed, showing the viability of the proposed approach. The results presented were all produced by a C implementation, and using as the evaluation engine the equation based approach, and simulation-based approach to determine the circuit performance.

5.1 FUGA with Equation Based Evaluation

This section presents an optimization processes using equation based as the evaluation engine. The results achieved with the standard genetic algorithm kernel mode versus fuzzy-genetic algorithm where the fuzzy rules are introduced in the mutation step of the GA are described. The proposed optimization approach is independent from the circuit, so, a well known circuit structure was selected in order to clearly illustrate the achieved performance gains. The circuit in Figure 5-1 a second order low pass filter with the transfer function in equation 13 and the equivalent representation in equation 14 in terms of the DC Gain, the cut frequency (ω0) and the quality factor (Q), is designed with the goal described in equations 15 and considering the passive elements sizes (R1 to R4; C1, C2) as optimization variables.

$$T(s) = \frac{\frac{1}{R_1 R_3 C_1 C_2}}{s^2 + s\left(\frac{1}{R_2 C_1} + \frac{1}{R_4 C_2}\right) + \frac{1}{R_2 R_4 C_1 C_2}} \tag{13}$$

$$T(s) = \frac{a_0}{s^2 + s\frac{\omega_0}{Q} + \omega_0^{\,2}} \tag{14}$$

$$\begin{aligned} A_{0\_th} &= 70dB \pm 0.1 \\ \omega_{0\_th} &= 3.6E6\, rad\, s^{-1} \pm 1E3 \\ Q_{0\_th} &= 0.48 \pm 0.02 \end{aligned} \tag{15}$$

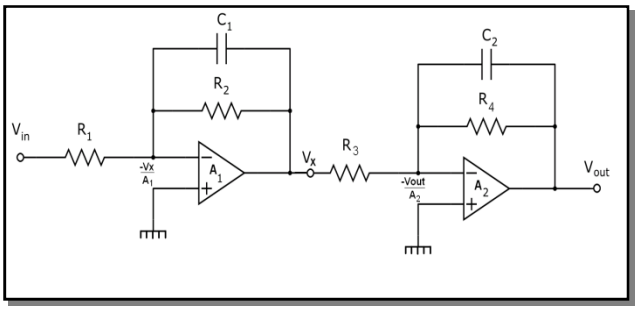
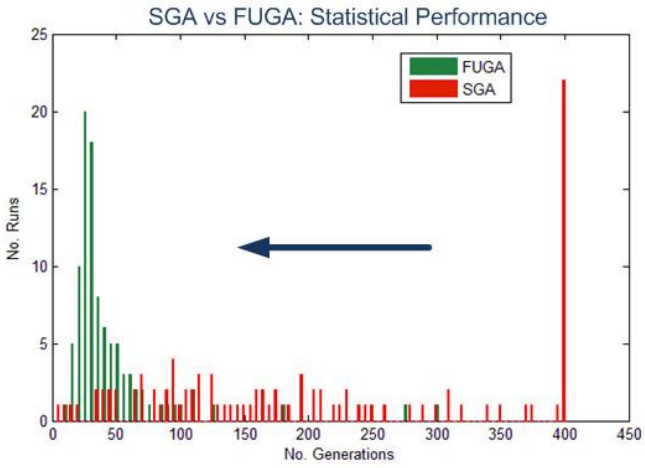


Figure 5-1 Active second order Low-Pass Filter

In Figure 5-2 and Table 5-1 the results clearly show that with FUGA approach the system algorithm performance is much better than with standard genetic algorithm.



Example: 2º Order Low-Pass

Figure 5-2 Result comparison

The results achieved presented in Table 5-1 shows that the success rate increased, the number of required generations is reduced, and the cpu time is higher for the fuzzy-genetic approach, due to additional computation required to determine the mutation value.

Table 5-1 2º Order Low-Pass Equation Based Results

	GA-Fuzzy	*Standard GA*
Nº Runs	100	100
Average (GA Gen. Num)	43.51	207.73
Std (GA Gen. Num)	48.26	132.17
Success Rate [%]	99	77
Cost Function Calls	289 090	1 233 990
CPU Time [seg]	8.71	1.38

Figure 5-3 illustrates the active filter bode diagram, achieved in the first generation and so one, proving that the algorithm converges to the solution.

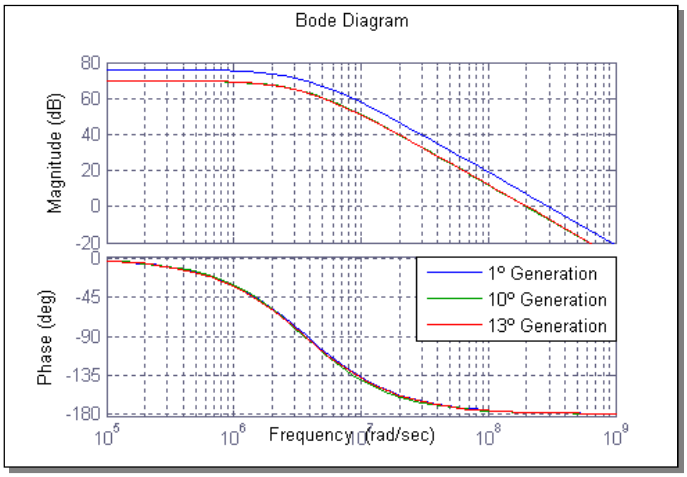


Figure 5-3 Bode Diagram's evolution for GA-Fuzzy

5.2 FUGA with Simulation Based Evaluation

The main objective in this section is to discuss and present a case study for the circuit synthesis problem using the system simulation based approach. The synthesis processes presented is characterized, and all the inputs and outputs illustrated.

5.2.1 Filter Benchmark Circuits

The schematic on Figure 5-4 shows the low pass filter used to present as one of the examples to test FUGA in simulation based mode. This example was intentionally made simple, aiming to make easier the understanding of all the system process steps, the results outputs achieved. This schematic was developed in cadence environment, which allowed the circuit.scs file generation used by the electric simulator spectre, in the evaluation process. The generated netlist, the file circuit.scs can be checked in appendix F, that was used as one of the inputs for spectre.

This section describes detailed information on the circuit specifications for low pass filter, the algorithm configuration and the output results for the optimization process using kernel in mode GA- Fuzzy and Standard GA. Table 5-2 presents the class of circuits used in this test and the number of optimization variable considered for this example.

Table 5-2 Class of circuit

Ident.	*Name*	*No.Devices*	*Opt.Var*
FilterLowPass	Low Pass Filter	4	2

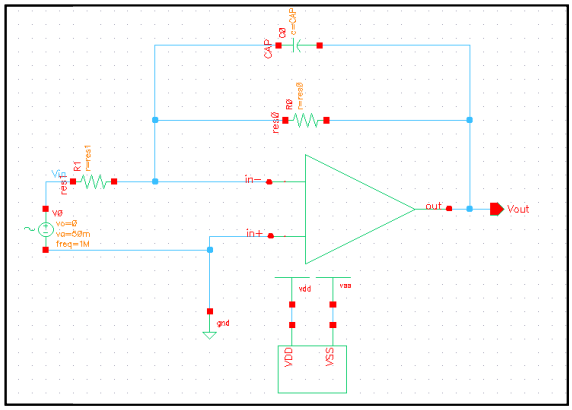


Figure 5-4 Low Pass Schematic in Cadence

The optimization parameters range configured for this examples are placed in Table 5-3.

Table 5-3 Parameters range

Id	*Res0[Ω]*	*Res1 [Ω]*	*Step*
FilterLowPass	[70, 440]	[2, 16]	0.1

The specification and requirements under test are placed in Table 5-4.

Table 5-4 Specification Requirements

Id	*Gain*	*GBW*	*Phase*	*Power*
FilterLowPass	> 40 dB	>20 MHz	60º<Ph<90º	Min (mW)

For this example the initial population size (=16), elite size (=2), mutation rate (10%), selection rate (50%) and a normal distribution method for generating the initial population. The stop criterion was here defined as a maximum number of runs (40). This example was intentional made with a low number of individual in the population so the chromosomes evolution can be followed in the different output files, for different optimization process steps. The algorithm was initialized with the following default configuration parameters listed in Table 5-5.

Table 5-5 Algorithm Configuration

Algorithm Setup	*GA-STD*	*GA-Fuzzy*
Number of GA cycles	40	40
Elite Number	2	2
Population Size	16	16
Selection Rate	50%	50%
Mutation Rate	10%	10%
Kernel type	GA Standard	GA Fuzzy
Kernel mode	Simulation Based	Simulation Based
Optimization variables	2	2
Sort Accuracy (δ)	0.0001	0.0001

The results for the low pass filter in Table 5-6 shows significant improvements when compared to the standard approach, particularly, the success rate is always superior, the number of required generations and the number of cost function evaluations is considerable reduced. This proposed approach can be generalized once it keeps or improves the benefits as both the number of optimization variables and performance specs increase.

Table 5-6 Results GA Standard vs. GA-FUZZY

Metrics	*GA-Fuzzy*	*Standard GA*
The Best chromosome's cost after 40 generation	6.000e-05	4.600e-04
The Worst chromosome's cost after 40 generation	3.500e-02	7.940e-02
Average fitness	2.000e-03	9.918e-03
Electric Simulation Calls	83.000	83.000

The best solutions found after 40 runs are presented in Table 5-7 representing the best Chromosome.

Table 5-7 Results Found

	GA-Fuzzy		*Standard GA*	
Optimization Variable	Res0	Res1	Res0	Res1
Results	2.889e+02	2.882e+00	2.795e+02	2.794e+00

Figure 5-5 illustrates the output data results for all the 40 runs, where it's clear that the GA-FUZZY has faster convergence than GA standard approach.

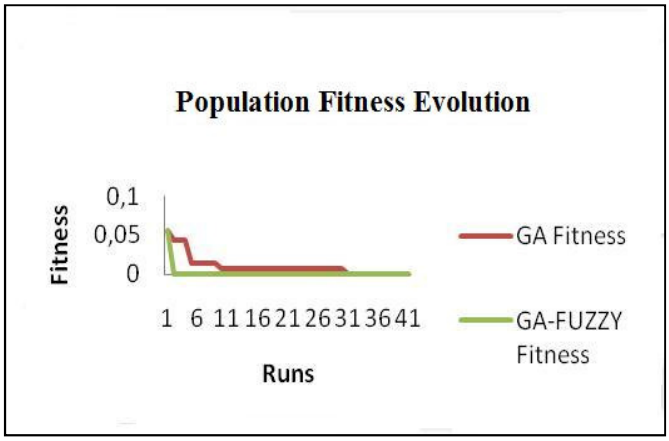


Figure 5-5 Fitness Evolution

Figure 5-6 illustrates the optimization variables values evolution, and the results achieve with GA-FUZZY approach stabilizes early compared with GA-STD.

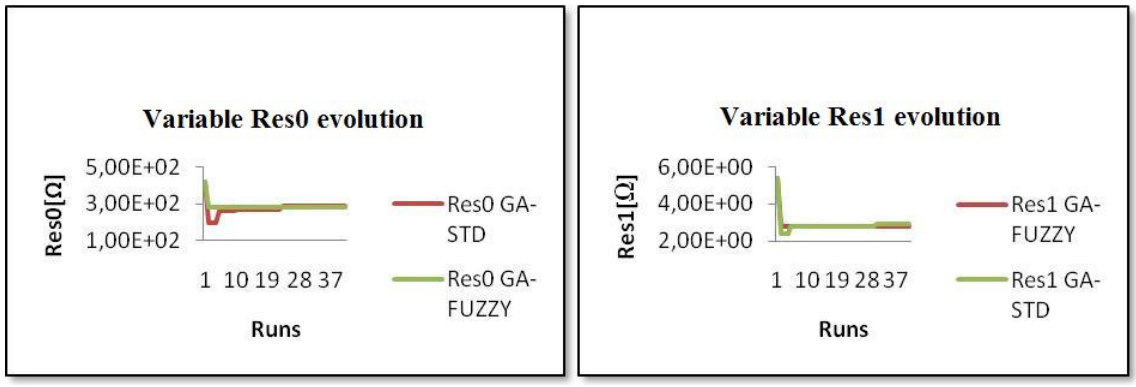


Figure 5-6 Res0 and Res1 results evolution

5.2.2 Operational Amplifier Case Study

The design of an operational amplifier (op amp) can be divided into two distinct design related activities that are for the most part independent of one another. The first of these activities involves choosing or creating the basic structure of the operational amplifier a diagram that describes the interconnection of all the transistors. In most cases, this structure does not change throughout the remaining portion of the design, but sometimes certain characteristics of the chosen design must be changed by modifying the structure.

Once the structure has been selected, the designer must select dc currents and begin to size transistors and design the compensation circuit. Most of the work involved in completing a design is associated with this, the second activity of the design process. Devices must be properly scaled in order to meet all of the ac and dc requirements imposed on the op amp computer circuit simulations, based on hand calculations that are used extensively to aid designer in this phase. Figure 5-7 shows the steps taken in consideration in the operational amplifier design, which is one of the most important building blocks in Analog circuit design.

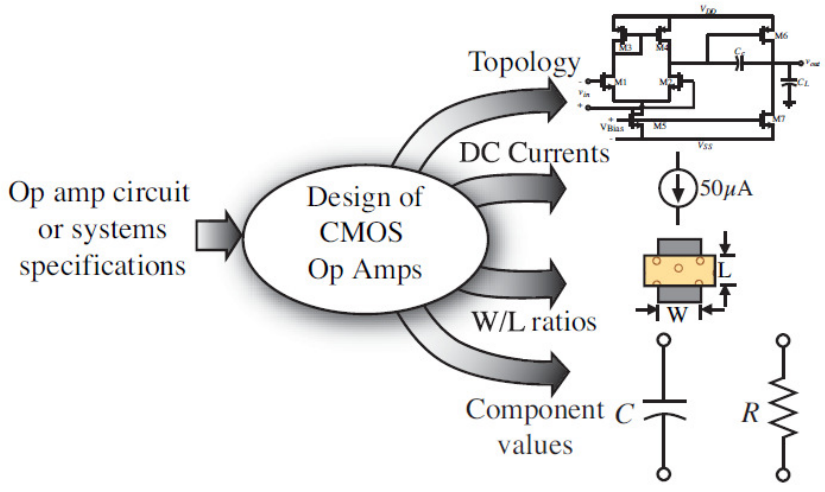


Figure 5-7 Operational Amplifier Design outputs [35]

Before the actual design of an op amp can begin, though, one must set out all of the requirements and boundary conditions that will be used to guide the design. The boundary conditions that must be considered are: Process specification (VT, K', Cox, etc.); Supply voltage and range; Supply current and range and Operating temperature and range.

The Requirements: Gain; Gain bandwidth; Settling time; Slew rate; Input common-mode range, ICMR; Common-mode rejection ratio, CMRR; Power-supply rejection ratio, PSRR; Output-voltage swing; Output resistance; Offset; Noise; and Layout area. In the following subsection a real example will be described that takes in consideration all the aspects described.

5.2.2.1 Two Stage Op-Amp

This section describes the design procedure for the Two Stage Unbuffered CMOS operational amplifier illustrated in Figure 5-8. The experimental results obtained with the systems using standard genetic algorithm approach and using fuzzy model are going to be compared and discussed.

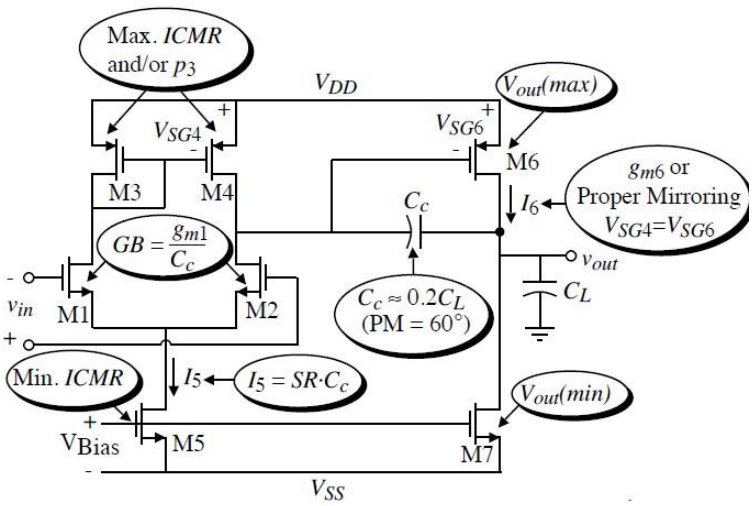


Figure 5-8 Two Stage Ampop design relationship [35]

The determined gain and power dissipation specification are given by this equation in Figure 5-9:

$$A_v = \frac{2g_{m2}g_{m6}}{I_5(l_2 + l_4)I_6(l_6 + l_7)} \qquad P_{diss} = (I_5 + I_6)(V_{DD} + |V_{SS}|)$$

Figure 5-9 Gain and power dissipation specification [35]

Table 5-8 shows the 9 optimization variables used given to the optimization kernel and to the model fuzzy module as the input parameters, with its minimum, maximum and step values.

Table 5-8 Optimization Parameters Range

Id	*W_m1m2*	*L_m1m2*	*W_m3m4*	*L_m3m4*	*Wm6*	*L_m6*	*Wm7*	*L_m7*	*Cc*
Two Stage Opamp	[1E-6, 20E-6, 1E-6]	[1E-6, 20E-6, 1E-6]	[1E-6, 20E-6, 1E-6]	[1E-6, 20E-6, 1E-6]	[1E-6, 20E-6, 1E-6]	[1E-6, 10E-6, 1E-6]	[1E-6, 20E-6, 1E-6]	[1E-6, 10E-6, 1E-6]	[1E-12, 10E-12, 1E-12]

The designed amplifier has to meet the specification in Table 5-9.

Table 5-9 Performance Parameter Specification

Id	*Gain*	*GBW*	*Phase*	*Power*

TwoStageOpamp	> 80 dB	>20 MHz	60º<Ph<90º	Min (mW)

The same configuration is set to both approaches, in order to compare the results under the same conditions.

Table 5-10 Optimization Algorithm Configuration Parameters

Algorithm Setup	***GA-STD***	***GA-Fuzzy***
Number of GA cycles	90	90
Elite Number	8	8
Population Size	40	40
Selection Rate	50%	50%
Mutation Rate	20%	20%
Kernel type	GA Standard	GA Fuzzy
Kernel mode	Simulation Based	Simulation Based
Optimization variables	9	9
Sort Acuracy (δ)	0.0001	0.0001

For the working example the first step is to call the DOE module. It is produced the information on the contribution of each circuit parameter to achieve the desired circuit performance. This information is used in the fuzzy model module. Figure 5-10 shows the contents of the input file for the fuzzy model, where optimization variables information are described

```
Number of optimization Variables
9
Optimization Variables Ranges
w_m1m2: [1.000000e-06;2.000000e-05]
l_m1m2: [1.000000e-06;2.000000e-05]
w_m3m4: [1.000000e-06;2.000000e-05]
l_m3m4: [1.000000e-06;2.000000e-05]
w_m6: [1.000000e-06;2.000000e-05]
l_m6: [1.000000e-06;1.000000e-05]
w_m7: [1.000000e-06;2.000000e-05]
l_m7: [1.000000e-06;1.000000e-05]
_Cc: [1.000000e-12;1.000000e-11]
Number of Output Variables
1
Output Variables Name
dcgain
For dcgain the variables that influences:
w_m1m2: Negativamente
l_m1m2: Negativamente
w_m3m4: Positivamente
l_m3m4: Negativamente
w_m6: Positivamente
l_m6: Positivamente
w_m7: Positivamente
l_m7: Positivamente
```

Figure 5-10 Fuzzy model input - output data file content

The fuzzy rules are defined based on the circuit behaviour and their general form is given by if-then rules, Figure 5-11 shows a part of the generated rules for this case study.

Construção das Regras Difusas
Subsets = 3
Variável de Saída: dcgain

w_m1m2	l_m1m2	w_m3m4	l_m3m4	w_m6	l_m6	w_m7	l_m7	Resultado
Low	Low	Low	Low	Low	Low	Low	Low	Med
Low	Low	Low	Low	Low	Low	Low	Mod	Max
Low	Low	Low	Low	Low	Low	Low	Hig	Max
Low	Low	Low	Low	Low	Low	Mod	Low	Max
Low	Low	Low	Low	Low	Low	Mod	Mod	Max
Low	Low	Low	Low	Low	Low	Mod	Hig	Max
Low	Low	Low	Low	Low	Low	Hig	Low	Max
Low	Low	Low	Low	Low	Low	Hig	Mod	Max
Low	Low	Low	Low	Low	Low	Hig	Hig	Max
Low	Low	Low	Low	Low	Mod	Low	Low	Max
Low	Low	Low	Low	Low	Mod	Low	Mod	Max
Low	Low	Low	Low	Low	Mod	Low	Hig	Max
Low	Low	Low	Low	Low	Mod	Mod	Low	Max
Low	Low	Low	Low	Low	Mod	Mod	Mod	Max
Low	Low	Low	Low	Low	Mod	Mod	Hig	Max
Low	Low	Low	Low	Low	Mod	Hig	Low	Max
Low	Low	Low	Low	Low	Mod	Hig	Mod	Max
Low	Low	Low	Low	Low	Mod	Hig	Hig	Max
Low	Low	Low	Low	Low	Hig	Low	Low	Max

Figure 5-11 Fuzzy rule generated data

Figure 5-12 presents the schematic used to test the operational amplifier.

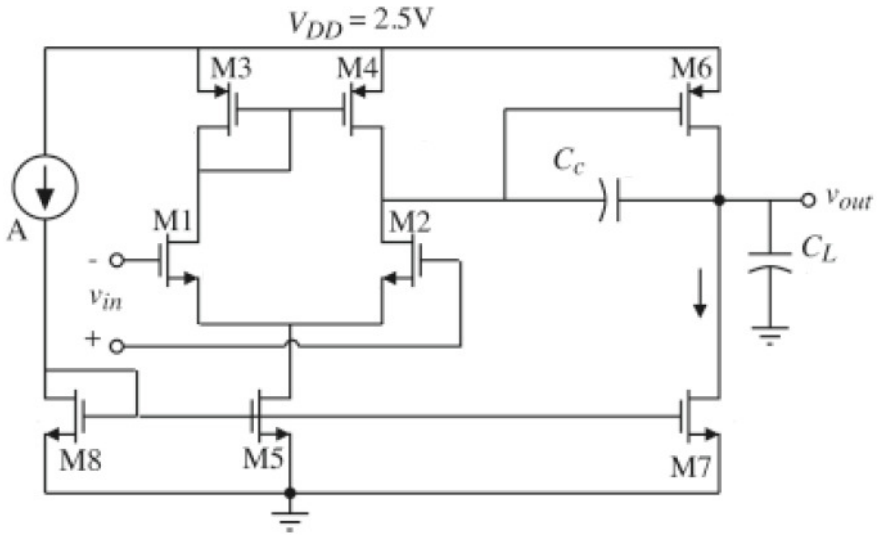


Figure 5-12 Two Stage Ampop Test schematic

The first analysis made involves the open-loop configuration shown in Figure 5-13 .

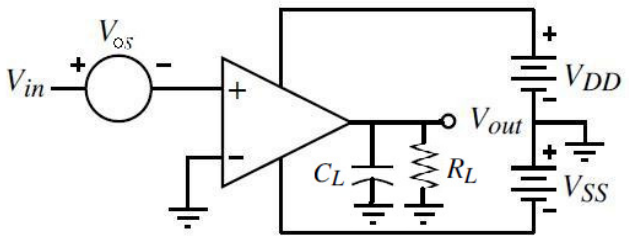


Figure 5-13 Open-loop mode with offset compensation

A coarse sweep of Vin is made from -5 to 5 to find the value of Vin where the output makes the transition from Vss to Vdd. The output – voltage versus Vin transfer function results obtained with wavescan (cadence graphical feature) is shown in Figure 5-14.

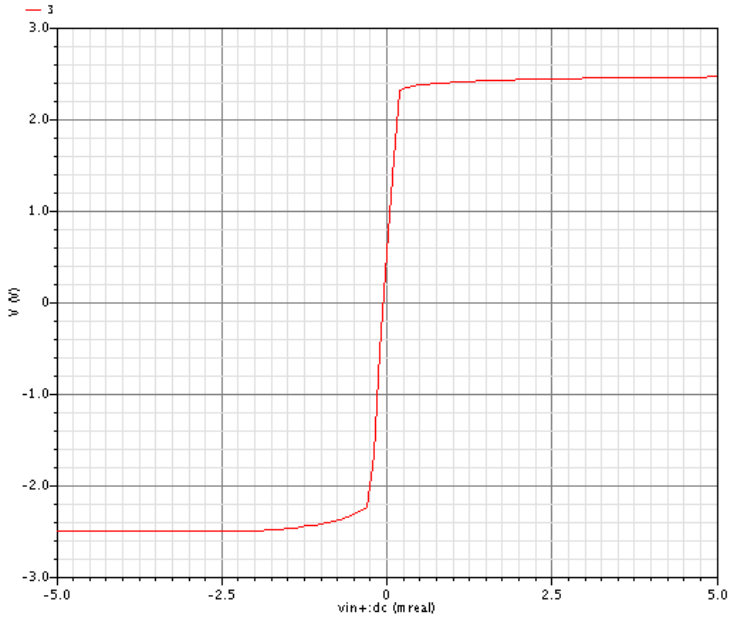


Figure 5-14 Open Loop transfer characteristic

Table 5-11 presents the results achieved with FUGA system performing both approach GA-STD and GA-Fuzzy. Table 5-12 has the solution found, with both approach.

Table 5-11 Comparison the Algorithms

Metrics	*GA-Fuzzy*	*Standard GA*
The Best chromosome's cost after 90 generation	2.00e-04	1.25e-06
The Worst chromosome's cost after 90 generation	3.97e-01	9.63e-01
Average Best Individual fitness	1.50e-2	7.26e-3
Electric Simulation Calls	183	183

Table 5-12 Final Transistor Dimensions

Opt. Variable	*W_m1m2*	*L_m1m2*	*W_m3m4*	*L_m3m4*	*Wm6*	*L_m6*	*Wm7*	*L_m7*	*Cc*
GA-STD Results	6.63e-6	1.00e-6	1.83e-5	3.00e-6	1.13e-5	2.72e-6	1.16e-5	6.00e-6	1.12e-12
FUZZY Results	1.56e-5	1.96e-6	1.04e-5	2.00e-6	1.17e-6	4.00e-6	1.16e-6	8.00e-6	1.22e-12

One great advantage of using spectremdl language features is that it allows the simulation of all the population at once and the results are placed in a standard form making the parsing of all the data less complex. Figure 5-15 shows the graphical output obtained with wavescan after the simulation for this case study, where the 40 chromosome open-loop transfer function magnitude response and the open-loop transfer function phase response are represented. This way is possible to have a graphical view over the output data.

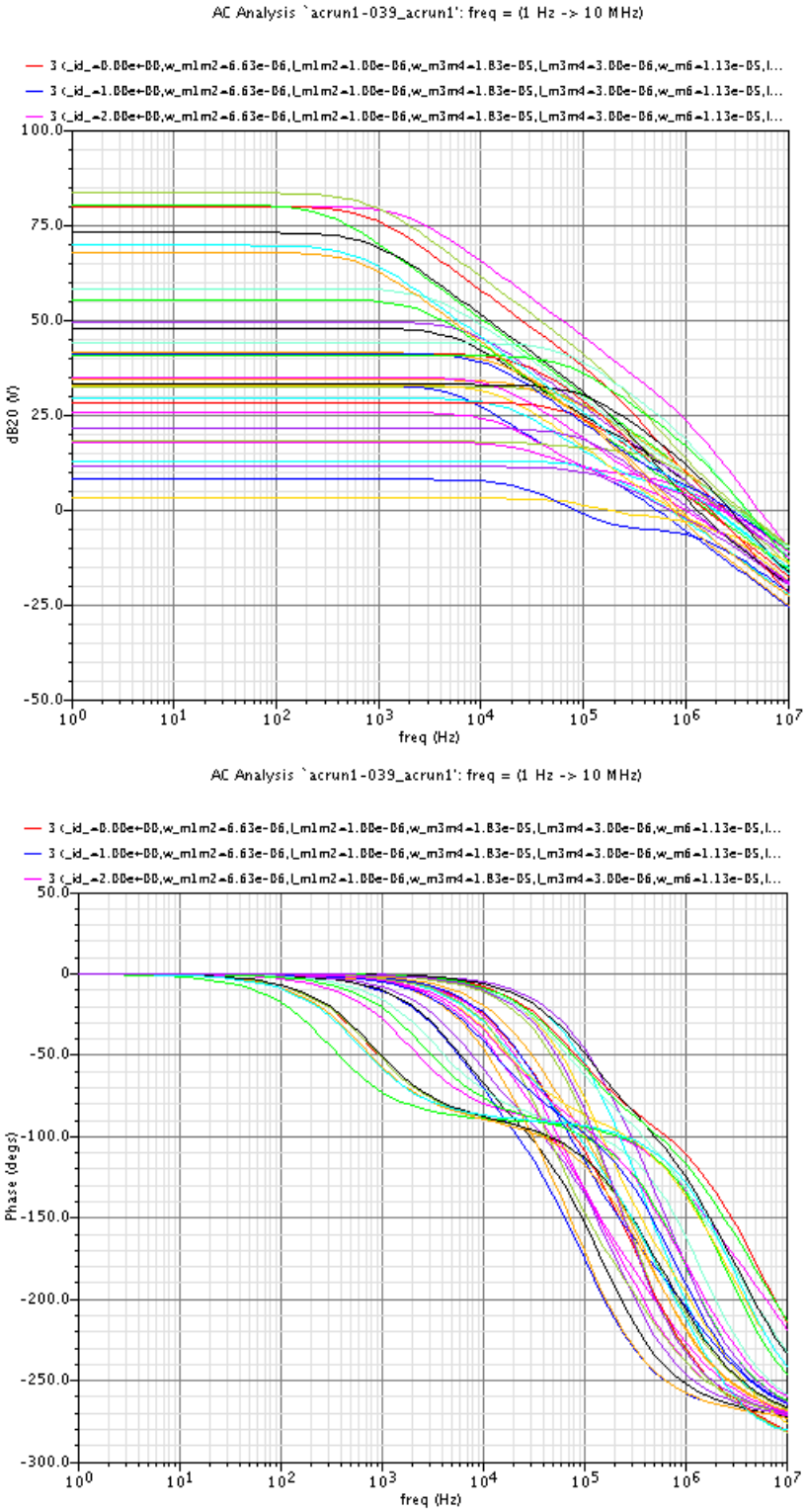


Figure 5-15 Transfer function for all the chromosome

The fitness evolution in this case study shows that the GA-Standard has a quicker convergence until run 11 as shown in Figure 5-16, but after this in run 12 and beyond the performance of GA-Fuzzy approach is a lot better, and with GA-STD the system never (after 90 runs) reached the desired gain > 80db, and with GA-Fuzzy approach this value is reached.

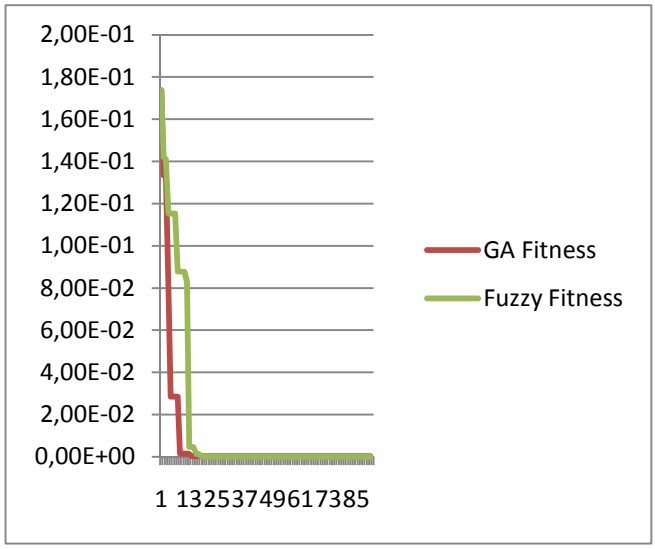


Figure 5-16 CMOS Two Stage Ampop Fitness Evolution

Figure 5-17 has the representation using a logarithm scale, to show clearly the two approaches performance, and FUGA in fuzzy mode, introduces considerable gains, in terms of performance, by using knowledge, represented by a set of fuzzy rules, to persuade the genetic algorithm to search in more promising directions.

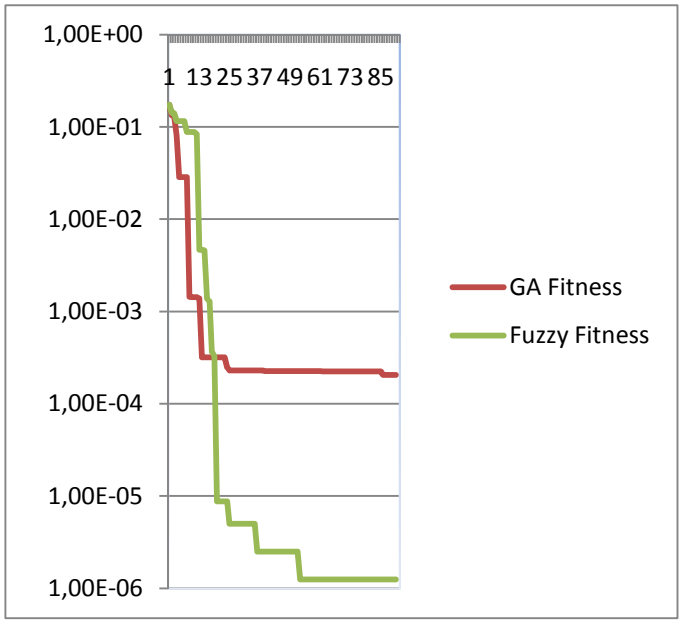


Figure 5-17 CMOS Two Stage Opamp Fitness Evolution Logarithm Scale

Figure 5-18 shows some of the 9 optimization variables values evolution through all the generations. This example also shows the scalability of the approach allowing its application to more complex circuit structures, with a higher number of optimization variables, given the fact that the systems operates independently, and in a generic process when handling the variables.

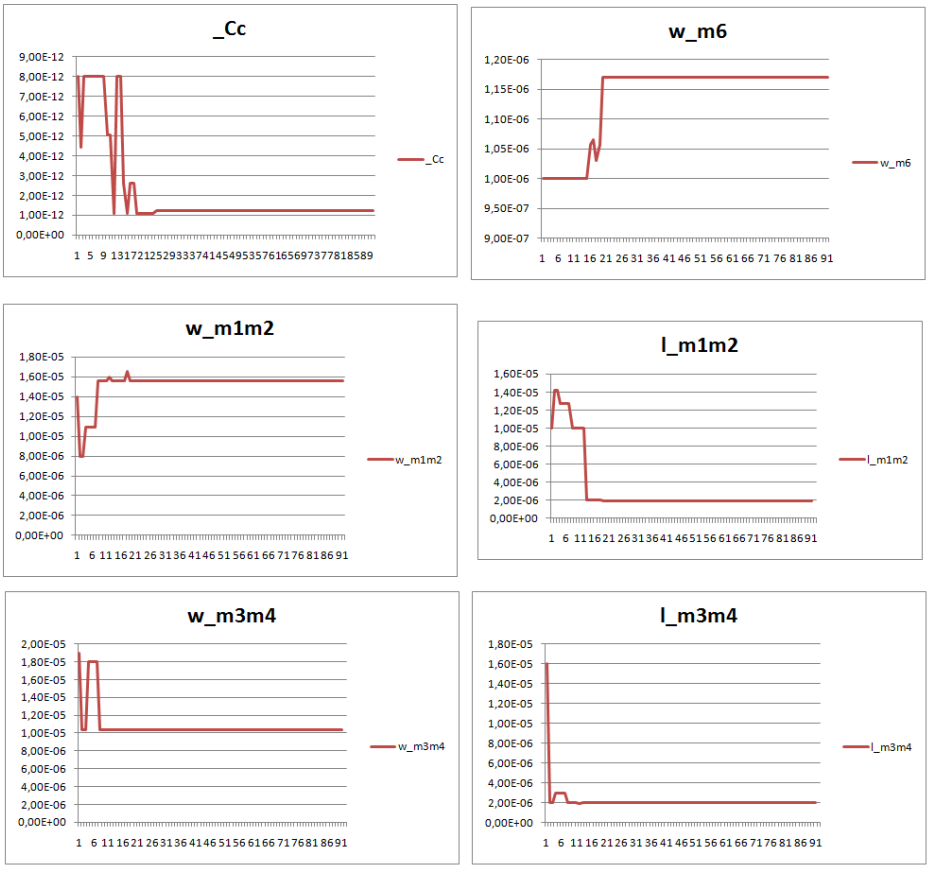


Figure 5-18 Genes Evolution

5.2.2.2 GA-SVM Optimization Kernel Results

The objective of this section is to use the two stage Opamp circuit example to compare the performance and the efficiency of the proposed Fuzzy genetic approach with the GA-STD, and an optimization approach based on a learning scheme using Support Vectors Machines (SVMs) combined with evolutionary strategies. This case study GA-SVM, and GA-SVM-SIM was defined in [36]. The GA-SVM optimization-based approach is demonstrated for the design of some Analog circuits using HSPICE as the evaluation engine, where the GA-SVM algorithm implements an optimization engine whose evaluated engine is made exclusively by the SVM performance model and the GA-SVM-SIM differs slightly, allowing also the coexistence with a circuit simulator.

The component sizes and ranges for this example are using a 0.35 µm CMOS technology process with a supply voltage of 5V, and the desired specification gain is (>60db): All transistor lengths were fixed at 1µm, CL was fixed at 10pF and Ibias fixed at 30pA. The circuit was optimized according to the parameters ranges of Table 5-13.

Table 5-13 Parameters Range

Opt. Parameters	***Limits***	***Step Size***
WI from M1=M2=M5	1.0e-6 - 200.0e-6	1.0e-6
W2 from M3=M4	1.0e-6 - 200.0e-6	1.0e-6
W3 from M7=M8	1.0e-6 - 200.0e-6	1.0e-6
W4 from M6	1.0e-4 - 350.0e-4	1.0e-6

The design results concerning device sizes are included in Table 5-14. Figure 5-19 shows the best solution achieved.

Table 5-14 Results Achieved

Opt.Par.	*GA-STD*	*GA-MOD*	*GA-SVM*	*GA-SVM-SIM*	*GA-FUZZY*
w1	3,00E-6	7,00E-06	5,00E-06	6,00E-06	1,40E-5
w2	1,74E-4	2,80E-05	1,50E-05	2,30E-05	1,11E-4
w3	9,77E-5	2,30E-05	2,00E-05	2,10E-05	6,37E-5
w4	2,23E-3	3,28E-04	1,70E-04	2,85E-04	4,87E-4

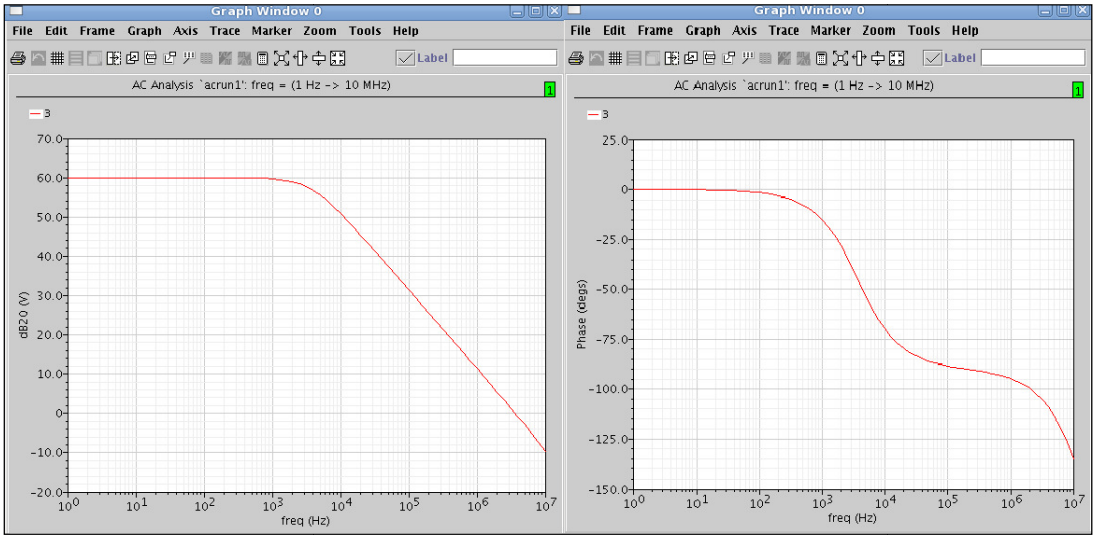


Figure 5-19 Two Stage Amplifier Vout representation

Finally, the behaviour of the algorithms is presented in Figure 5-20 and Figure 5-21, where Fuzzy shows better results for this example.

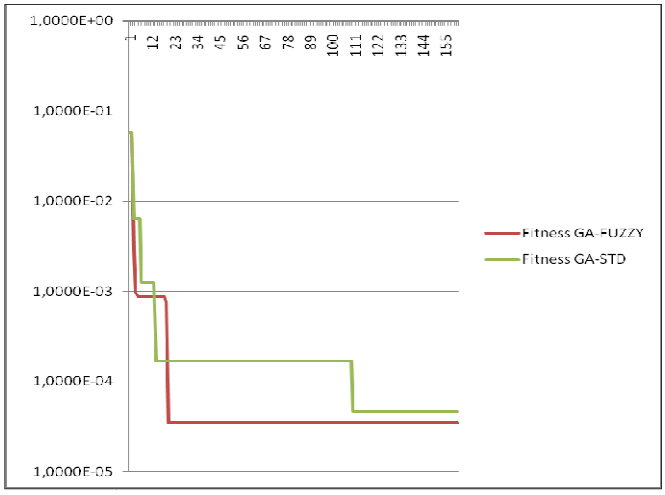


Figure 5-20 GA-STD vs. GA-FUZZY Results

The achieved results show significant gains in efficiency over the GA-STD for FUGA approach and GA-SVM-SIM, between this two last approaches the performance in this case study shows similar behaviours, in terms of fitness evolution.

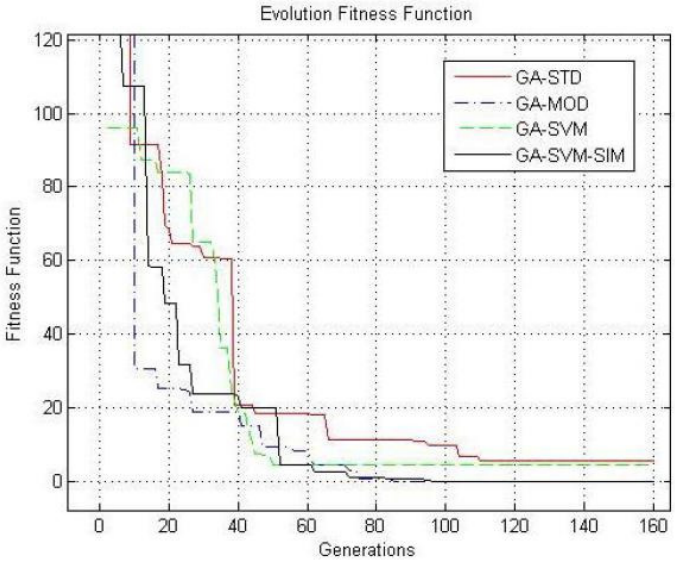


Figure 5-21 Evolution of the Cost Function

5.2.2.3 Operational Amplifier using Bipolar

Another interesting example is an operational amplifier with bipolar transistors, used also as a design problem to be optimized. The circuit netlist used to test this example is present in APPEDIX G.

Table 5-15 Bipamp Optimization Parameters Range

Id	*valR0*	*valR2*	*valR4*	*valR6*	*valR9*	*areQ15*	*areaQ5*	*areaQ24*	*areaQ7*
BipAMP [min,max,step]	[1, 200,1]	[1, 200,1]]	[1, 200,1]]	[1, 200,1]]	[1, 20e+3,1]	[1, 10,0.4]	[1, 10,0.4]	[1, 10,0.2]	[1, 10,1]

The stop criteria used in this example was the number of generation (=90), present in Table 5-16.

Table 5-16 Bipamp Optimization Algorithm Configuration Parameters

Algorithm Setup	***GA-STD***	***GA-Fuzzy***
Number of GA cycles	90	90
Elite Number	2	2
Population Size	16	16
Selection Rate	50%	50%
Mutation Rate	10%	10%
Kernel type	GA Standard	GA Fuzzy
Kernel mode	Simulation Based	Simulation Based
Optimization variables	9	9
Sort Acuracy (δ)	0.0001	0.0001

For this example it's clear the performance for the GA-Fuzzy is a lot better than the standard approach, as can be checked in Figure 5-22.

Table 5-17 Bipamp Specification

Id	*Gain*
BipAMP	> 40 dB

Table 5-18 Final variables Dimensions

Optimization Variable	*valR0*	*valR2*	*valR4*	*valR6*	*valR9*	*areQ15*	*areaQ5*	*areaQ24*	*areaQ7*
GA-STD Results	191.0	193.0	196.0	71.0	9917.0	9.0	9.4	2.4	2.6
FUZZY Results	136.0	189.0	151.0	171	9923.0	27.7	2.4	1.9	32.7

Although the system runs for 90 generation, analysing the achieved results, it's confirm that with the FUGA approach in fuzzy mode the optimal solution is reached around generation 10. And in Figure 5-22 Fitness Evolution, the initial conditions for the GA-STD was better than the fuzzy initial conditions, that in both cases depends on the uniform random generated genes values, but FUGA in FUZZY mode with the circuit design knowledge had a quicker convergence than the GA-STD.

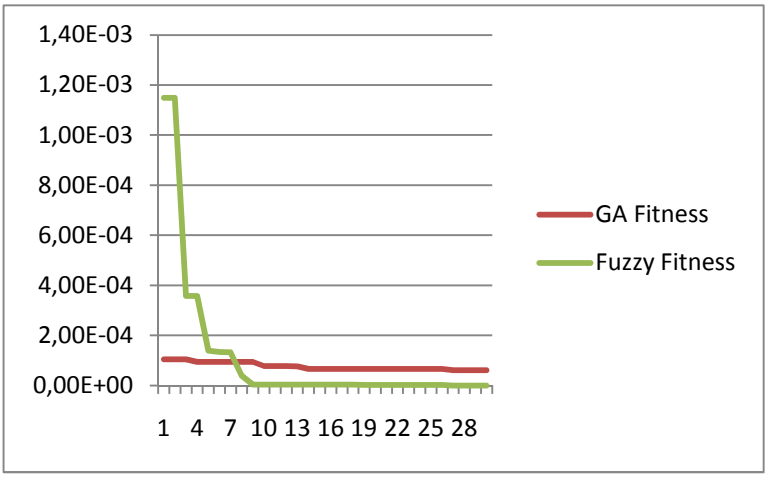


Figure 5-22 Fitness Evolution

Figure 5-23 shows the gene values evolution, where is clear that the big variation in the genes values happens until the generation 10.

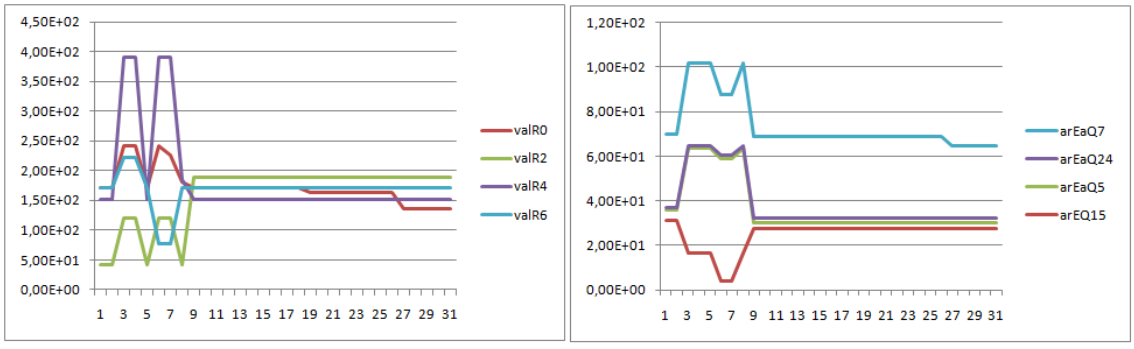


Figure 5-23 Genes Evolution

Figure 5-24 shows operational amplifier with bipolar transistors best solution achieved after the spectremdl simulation.

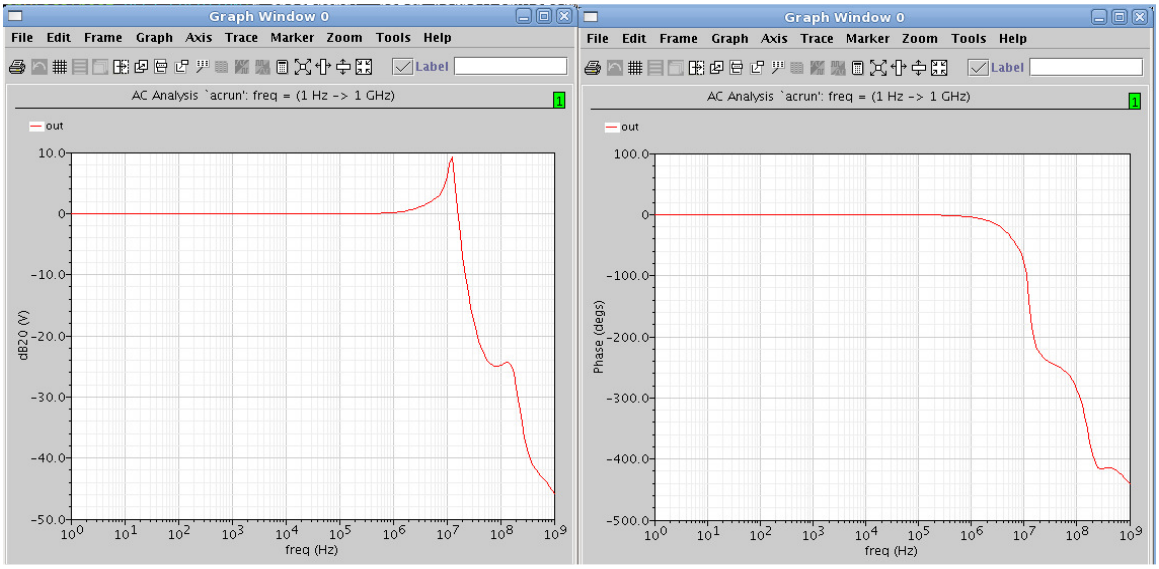


Figure 5-24 Bipolar Operational Amplifier Vout representation

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

An innovative Analog Circuit Design optimization Kernel based on a Genetic Algorithm - Fuzzy Models & Design Of Experiment was presented. The proposed approach "FUGA" achieves good results compared with traditional standard GA approach.

The proposed Fuzzy-Genetic approach, to implement an optimization kernel for circuit level design, introduces considerable gains as illustrates the results (figure 6.1.1), in terms of performance, by using design knowledge, represented by a set of fuzzy rules, in order to persuade the genetic algorithm to search in more promising directions.

Preliminary results show a large reduction in terms of the number of fitness function evaluations and, also, the number of generation needed to reach the desired solution. The additional time required to reach the solution is not relevant when considering the time needed to perform the additional number of electrical simulation in the case of standard GA approach. Finally, the automatic generation of the design fuzzy rules represents an additional advantage of the proposed solution when optimizing highly complex structures.

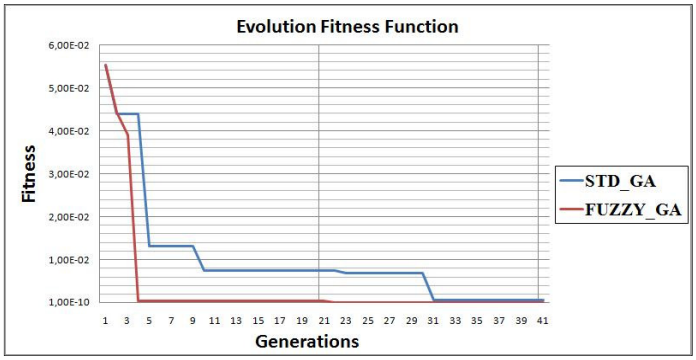


Figure 6-1 Fitness Evolution for Low Pass circuit

6.2 Future Work

The synthesis system can be improved in many different aspects. The corner analysis can be explore in more detail, and more circuit can be tested and compare the results between fuzzy and standard, also compare with cadence and other available state of the art implementation. The integration of the system inside the cadence environment is another interesting work to be made, using skill language to integrate. Finally different GA operator's algorithm can be tried to compare and improve the approach.

There are many aspects that can be proposed as a future work to continue this project, given the fact that developing a complete system for Analog and mixed signal design automation at synthesis level is a complex engineering problem that is under research and with some commercial application, that requires a strong development, integration and test process, and it will always demand more work to be done, due to technological advances.

It's valuable to produce more exhaustive tests, using complex circuits to validate and optimize the new GA-Fuzzy Kernel approach.

Implement and explore the Fuzzy Model in different GA operators. Research and develop new techniques to improve the Kernel performance.

Include in FUGA approach, optimization strategies to handle Corner Analysis and Monte Carlo Analysis.

Use parallel programming to fully exploit increasingly parallel hardware and achieve good runtime scaling. Use Skill programming language to develop a larger interaction with different Cadence functionalities.

Research and develop DFM methodology and techniques to address the nanotechnologies challenges.

APPENDIX A Kernel Manual

A.1 Optimization Kernel User Manual

NAME:

KERNEL – System that optimizes the synthesis process in a circuit design project by generating automatically a fully optimized circuit from a designer specifications input and interacts with a model engine, a circuit simulator engine, a graphic interface and a commercial tool (CADENCE) for circuit design.

SYNOPSIS

./KERNEL –cfg configuration_file.cfg -par parameters_file.par
–spc specsification_constraint.spc [OPTIONS] [OUTPUT FILES]

DESCRIPTION

The Optimization Kernel is the principal module, where the genetic algorithm searches for an optimal solution to the circuit synthesis process, under many constraints to reach a set of objectives specified by the circuit designer. To find the solution the optimization kernel uses a standard genetic algorithm flow and also has an option that combines the genetic algorithm with model engines like Design of experiment model combined with Fuzzy model that guides the algorithm to a quicker convergence.

It's possible to specify a list of optional input argument to the program that will be described in this manual. This kernel can function as a standalone tool with a specific option but some options requires the interaction with the interface to handle the output data, or the communication interaction with external programs.

The configuration file must be introduced and it has the algorithm configuration necessary to initialize the optimization operators, and these parameters will have a major impact on the algorithm behavior, consequently the convergence, so they must be adjusted according with the designer needs. Inside the input file there is information as comment statements, explaining the objectives of each value. If using the graphical interface there is placed on the configuration window separator, the information about each input for the configuration value.

The parameter file must be introduced also, and it has mainly the optimization variables name, maximum and minimum values, the step for the random initialization, and finally information explaining the parameter input file, and if using the graphical interface these information are also placed in the parameter input window separator.

OPTIONS

- fm, fuzzy mutation

In the genetic algorithm optimization flow the mutation operator will be guided by fuzzy model rules and values, when this option is specified as an input argument. This guidance is based on the design of experiment conclusions of how the input optimization variables influences the circuit behaviour, if this

parameter is not specified as an input argument the mutation operator will have a standard behaviour based on random generated values.

- fc, fuzzy crossover

In the genetic algorithm optimization flow the crossover operator will be guided with a routine based on the information given by the fuzzy model values, based on design of experiment conclusions, if this option is not specified the crossover operator will have a standard behaviour.

- CIRCUIT_SIM electrical circuit simulator

The optimization kernel can operate in two distinct modes, the equation based mode and the simulation based mode. When this option is specified the Kernel operates in the simulation based mode, otherwise it will operate in equation based mode.
In simulation based mode there must be used an external module that handles the communication interface, between the optimization kernel and the electrical circuit simulator. This external interface module has to call the kernel program, and after receiving a Begin Simulation Order and the input data, it will start the electrical circuit simulation, and after finishing the electrical circuit simulation it has to send to the kernel input the simulation results, with the result data organized in the same order indicated by the kernel, this interaction will flow inside the optimization loop, until the kernel send the stop simulation order to the communication interface module, and that happens when a final solution is reached or the number of generation is reached.

- model Design Of Experiment Model

This option allows the optimization kernel to lunch the DOE- Design of experiment module.
To use genetic algorithm Kernel combined with fuzzy model, this option must be specified at least for the first time when the optimization process begin, so the external model module DOE will generate the necessary outputs with the model conclusions of how the input optimization variable influence the circuit output and the fuzzy logic rules that will be the inputs needed by fuzzy model module to guide the algorithm.

- res Optimization Result folder

This option can be specified if there is a need to specify the folder where the optimization data and debug result are going to be placed; otherwise the results are placed in a default folder [Results_Default].

APPENDIX B System Description

B.1 Hardware and System Software Description

CPU Intel CORE 2 QUAD CORE
CPU Intel Core 2 Quad core, Q6600 BOX2GB
Fedora core 7 Operating System
1 - Transcend 1 GB DDR2 800MHz
2 - Transcend 1 GB DDR2 800MHz

B.2 Installation Manual

When running the Optimization Kernel to use Circuit Simulator, it's mandatory to make sure that the user sources the file cshrc.cadence in cadence installation directory in order to set all the need environmental variables.
In this system this file is placed in a different partition:
/home/cadmgr/config/
The system has to have installed the following python modules:
Tkinter
Gnuplot
Numeric
Numpy

APPENDIX C Equation Based Functions

C.1 Benchmark Functions

Test Problem 1 (G_6 in Michalewicz and Schoenauer [1996])

$$\text{Minimize } G_6(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$

$$\text{subject to: } \begin{aligned} &(x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0 \\ &-(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \geq 0 \end{aligned}$$

$$\text{with } 13 \leq x_1 \leq 100, 0 \leq x_2 \leq 100.$$

The solution is

$$x^* = (14.095, 0.84296),$$

$$G_6(x^*) = -6961.81381.$$

$$G_7(x) = x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2$$
$$+2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

$$\text{subject to } \begin{aligned} &105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0 \\ &-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0 \\ &-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \\ &-x_1^2 - 2(x_2 - 2)^2 + 2x_1 x_2 - 14x_5 + 6x_6 \geq 0 \\ &8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \\ &-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0 \\ &3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0 \\ &-0.5(x_1 - 8)^2 - 2(x_2 - 4) - 3x_5^2 + x_6 + 30 \geq 0 \end{aligned}$$

with $-10 \leq x_i \leq 10, i = 1, \cdots 10$.

The solution is

$$x^* = (2.171996, 2.363683, 8.773926, 5.095984,$$
$$0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$$

$$G_7(x^*) = 24.3062091.$$

Test Problem 3 (G_9 in Michalewicz and Schoenauer [1996])

Minimize

$$G_9(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

subject to

$$2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127 \le 0$$

$$7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 \le$$

$$23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 \le 0$$

$$4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0$$

with $-10 \le x_i \le 10, i = 1, \cdots 7$.

The solution is

$$x^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$$

$$G_9(x^*) = 680.6300573$$

APPENDIX D State OF ART TABLE

Tool	Year	Description	Techniques	Obs.
KOAN/ANAGRAM	1991	Macro-cell Place and Route; uses pre-defined small module generators data-base; synthesizes an optimized layout configuration from a given Spice Net-list with symmetry, matching and tech. specs.	Optimization based with Simulated Annealing.	The chosen library constitutes a limit of this method since an enormous number of pre-designed layout blocks is required
Layla	1995	It takes into account symmetry constraints, performance degradation due to interconnect parasitic and device mismatches and combines this with geometrical optimization techniques (devices merges, abutment, etc.)	Optimization based with Simulated Annealing.	A performance-driven methodology where all performance constraints are satisfied. Optimize the layout quantifying the performance degradation.
A SKILLTM -based Library for Retargetble Embedded Analog Cores	2001	Automatic generation and reusability of physical layouts of analog and nixed-signal blocks based on high-functionality pCells that are fully independent of technologies.	Knowledge-Based	Parameterized cells (pCells) are organized hierarchically.
ALDAC	2002	Module Generator	Simulated Annealing	
IPRAIL	2004	Retargeting is achieved using a automatically extracted template and using a circuit optimizer to size the cells. Uses either a rule or a performance driven approach. Uses optimization based with knowledge-based	Linear Programming and graph short path on the relational template extracted from the source layout	(+) General approach. (-) Larger run-time required.
ALLADIN	2004	The layout generation is based on relatively complex sub-circuits. Designers can construct layouts of parameterizable modules in a technology and application independent way. The placement and routing of modules are performed automatically under the constraints defined by designers.	Three phase Place and Route:1 – GASA e half-peri routing; 2 – VFSRA e global routing (fine tuning); 3 – Detailed Routing	Design platform for analog circuits, based on a user managed device generators library.
LAYGEN	2005	Expert knowledge is used to guide an evolutionary algorithm during the automatic generation of the layout The designer provides a high level layout description where position and interconnections are predefined. This template contains placement and routing constrains and is independent from technology. Deal with hierarchically templates for more complex circuits.	Knowledge-based with evolutionary computation techniques. Uses a geometric template	(+) Speeds up retargeting operations or technology migration (-) Works better when changes in circuit parameters result in small adjustments. for the target technology

APPENDIX E Overview of analog sizing tools [29]

Features / Tools	Date	Evaluation Class	Algorithms / Techniques	Equation / Design Plan	Implementation Language	Robust Design	Circuit Complexity	Computation Time Effort	Setup-Time	Interactive Design	Bookkeeping	Encapsulation	Advantages or Particular Properties
IDAC	1987	Knowledge Based (KB) Simplified Equations	Design plan + Post Optimization	Symbolic analyzer + manually	—	—	5-20 devices	A few seconds (VAX780)	Long time to add a new circuit more than 4 months. Employs worst-case design.	—	—	—	Able to size a library of analog schematics as a function of technology and building-block specs.
DELIGHT SPICE	1988	Circuit Simulator	Feasible Directions	—	—	✓	20 Par 28 devices	18h (Masscomp MC 500)	Long time run. Use SPICE. Problems with simulator convergence.	—	—	—	Uses a large library of opt. algorithms
OASYS	1989	KB Simplified Equations	Design Plan + Backtracking	Manually and specific to each class	—	✓	19 Par 17 devices OpAmp	3 sec. (VAXstation II/GPX)	Long time to add an new circuit. Approx. 6 months, including circuit analysis	—	—	—	Employ task decomposition Simplifies design reuse. Determine feasible and infeasible surfaces
BLADES	1989	KB Lookup Tables and Rules	Artificial Intelligence	Manually	—	—	34 devices OpAmp	A few seconds to design and 20 min to complete simulation	Long time to create and maintain look-up tables. Uses divide and conquer strategy to partition blocks written by "if-then clauses"	—	—	—	Uses a divide and conquer. Ability to perform circuit design, generate test files and invoke circuit simulator automatically
OPASYN	1990	Simplified Equations	Grid + Steepest Descent	Manually. Simple analytical model	**C and Lisp**	✓	7 Par, 60 devices OpAmp	5 min. (VAX 8800)	2 weeks for known circuits and a few days for simpler circuits.	—	—	—	Interface with Berkeley CAD environment. Includes layout simul.
MAULIK	1991	Simplified Equations + BSIM	Branch and Bound	Manually	—	—	39 Par, 19 devices	1 min. DEC 3100	6 months, including circuit analysis	—	—	—	Topology selection + device sizing
STAIC	1992	Simplified Equations	Two Step Optimization	Manually	**C++**	—	22 device OpAmp	3 min. MIPS 2000	Entering new descriptions take long time.	—	—	—	Presents an analog description language with 4 levels
FRIDGE	1994	Circuit Simulator	SA with SPICE +local Method	—	—	x	10 Par OpAmp	45 min. Limited to a few thousand evaluations.	Effort to add a circuit in about one hour.	—	—	—	Has only been demonstrated for problems of a small number of opt. var.
ISAID	1995	Simplified Equations + Qualitative Reasoning	Qualitative Reasoning + Post Optimization	Manually	—		8 Par 13 devices OpAmp	—	—	—	—	—	Replace exact performance relations with quantitative relations.

Features / Tools	Date	Evaluation Class	Algorithms / Techniques	Equation / Design Plan	Implementation Language	Robust Design	Circuit Complexity	Computation Time Effort	Setup-Time	Interactive Design	Bookkeeping	Encapsulation	Advantages or Particular Properties
SD-OPT	1995	Equations + Behavioral Simulations	SA	Manually	—	x	Forth-order sigma-delta modulator	1.5 weeks	Exhaustive analysis Required. High cost to implement new structure	x	x	x	Design for switched-capacitor delta-sigma modulators
FASY	1995	Simulation Based	Fuzzy + SA + Gradient+ NN	SPICE as the evaluation engine.	—	x	9 devices Ampop	6 hours (96 MIPs workstation)	—	x	x	—	Hybrid SA for course and gradient for fine simulations and fuzzy logic to aid in topology selection.
ASTRX/OBLX	1996	AWE + Equations	SA	Automated	**C**	x	33 devices	11.8h/run (IBM RS/6000-550)	A few days to add new circuits.	x	x	✓	Simulation + equation Based on waveform evaluation. Setup highly automated.
GPCAD	1998	Simplified Equations + GP	Simple Primal Barrier Method	Manually	**Matlab**	x	10 devices OpAmp	A few seconds	Doesn't specify setup time. Doesn't include automatic generations of equations.	x	x	x	Uses Geometric programming. Short execution times. Designed for Opamps.
MEALSTROM	1999	Circuit Simulator	GA+SA	—	**C++**	x	27 Par 32 devices	3.6 h (15) Sun Sparc - 1)	Uses Cadence GUI. The long times are associated with simulations run.	✓	✓	✓	Uses simulator in distributed environment.
ANACONDA	2000	Circuit Simulator	Stochastic Pattern Search	—	**C++**	x	20 Par, 37 devices	10 h (24 Sun Ultra 10)	Encapsulate commercial simulators (TISpice).	✓	✓	✓	Add constraints on device operating regions to ensure a safety margin from manuf. or envir. variations. Parallelism.
AMGIE	2001	Simplified Equations	SA + several Local Methods	Symbolic analyzer + manual	—	x	14 Par 9 devices	5 min	8 hours	x	x	x	Complete design flow. Device sizing is a compilation of tools
ALPAYDIN	2003	Fuzzy Neural Network	Evolutionary Strategies + SA	Training points + circuit simulator	—	✓	31 devices OpAmp	45 min. (124 min. with mismatch model)	Accuracy depends on training points. Cannot handle mismatch.	x	x	x	The N-F model uses a set of training data from SPICE.
VINCENTELLI	2003	SVM	LIBSVM	Performance model of analog circuits	—	—	4 opt. variables	10's min 50.000 samples	—	—	—	—	Improve accuracy of estimator using Active Learning strategy.
VEMURI	2004	NN	MIT GAlib	performance parameter macro-models	**Matlab**	—	3125 sample	51.9µs of execution time	1h47min SunBlade100 for training samples.	—	—	—	Explores sampling methodologies to improve model quality.
GENOM	2006	Circuit Simulator +SVM	GA / SVM	SPICE/HSPICE engines	**C**	✓	31 Par 21 device 41 const	20min	Encapsulate in-house environment (AIDA).	✓	✓	✓	Distributed processing and robust design.

APPENDIX F Low Pass Spectre Netlist

Circuit Test Benches Low Pass

```
// Generated for: spectre
// Generated on: Apr  8 00:45:07 2011
// Design library name: filterLib
// Design cell name: LowPass
// Design view name: schematic
simulator lang=spectre
global 0 vss! vdd!
include
"/home/cadmgr/cadence/cdn_opus_5141_ISR77/tools/dfII/samples/artist/ahdlLib/quantity.spectr
e"
parameters res1=2 res0=140 CAP=2p _id_ = 0

include "InputParameter.scs"

// Library name: filterLib
// Cell name: supply
// View name: schematic
subckt supply VDD VSS
parameters VSS VDD
    V2 (VSS 0) vsource dc=VSS type=dc
    V0 (VDD 0) vsource dc=VDD type=dc
ends supply
// End of subcircuit definition.

// Library name: filterLib
// Cell name: idealOpAmp
// View name: schematic
subckt idealOpAmp in\- in\+ out
    C7 (in\+ in\-) capacitor c=500f
    I2 (net390 0 in\+ in\-) vcvs gain=1000000.000000
    R8 (net390 out) resistor r=100m
    I1 (in\+ in\-) resistor r=1e7
ends idealOpAmp
// End of subcircuit definition.

// Library name: filterLib
// Cell name: LowPass
// View name: schematic
I8 (vdd! vss!) supply VSS=-5 VDD=5
OP0 (net022 0 Vout) idealOpAmp
V0 (Vin 0) vsource dc=4 mag=1 type=sine sinedc=0  ampl=50m  freq=1M
C0 (net022 Vout) capacitor c=CAP
R1 (Vin net022) resistor r=res1
R0 (net022 Vout) resistor r=res0
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 ckptclock=1800 \
    sensfile="../psf/sens.output" checklimitdest=psf
ac ac start=1 stop=1M annotate=status
dcOp dc write="spectre.dc" homotopy=all maxiters=150 maxsteps=10000 \
    annotate=status
dcOpInfo info what=oppoint where=rawfile
dc1 dc param=temp start=25 stop=27 oppoint=rawfile homotopy=all \
    maxiters=150 maxsteps=10000 annotate=status
modelParameter info what=models where=rawfile
```

```
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts  where=rawfile
saveOptions options save=allpub
```

APPENDIX G BIPAMP Spectre Netlist

```
/ Design library name: amplib
// Design cell name: bipamp_sim
// Design view name: schematic
simulator lang=spectre
global 0 vdd! vss!

parameters vmag=1 v1=0 v2=3 tr=300n Tp=3u tw=1u td=1n vdd_val=6

// optimization variables
// parameter outside amplifier
parameters Rvcvs= 10 R0=1M R6=1 CoutLoad=0

// parameters to be optimized inside amplifier cell

parameters valR0=100 valR2=100 valR4=100 valR6=100 valR9=17.6K
parameters areQ15=1 areaQ5=1 areaQ24=8 areaQ7=2 _id_=0

// parameter variation

include "InputParameter.scs"

// Included Model Files

include "./models/globals.scs"

include "./models/corner_lib.scs" section=tt
// Library name: amplib
// Cell name: bipamp
// View name: schematic
subckt bipamp inm inp out vbias inh_bulk_n
    R0 (vdd! net32) resistor r=valR0
    R2 (vdd! net35) resistor r=valR2
    R4 (vdd! net90) resistor r=valR4
    R5 (vdd! net93) resistor r=200
    R6 (net14 vss!) resistor r=valR6
    R7 (net16 vss!) resistor r=17.6K
    R8 (net18 vss!) resmod l=10u w=2u
    R9 (net20 vss!) resistor r=valR9
    R11 (net22 vss!) resistor r=600
    R10 (net24 vss!) resistor r=240
    R12 (net26 vss!) resistor r=9.2K
    Q3 (net41 net28 net29 inh_bulk_n) kpnp area=1
    Q0 (net29 net29 net32 inh_bulk_n) kpnp area=2
    Q1 (net47 net47 net35 inh_bulk_n) kpnp area=1
    Q2 (net99 net47 net35 inh_bulk_n) kpnp area=1
    Q5 (net44 inm net41 inh_bulk_n) kpnp area=areaQ5
    Q7 (net14 vbias net44 inh_bulk_n) kpnp2 area=areaQ7
    Q12 (net96 net96 net47 inh_bulk_n) kpnp area=1
    Q16 (net56 net74 net29 inh_bulk_n) kpnp area=1
```

```
    Q19 (net52 net52 net35 inh_bulk_n) kpnp area=2
    Q15 (net44 inp net56 inh_bulk_n) kpnp area=areQ15
    Q20 (net78 net99 net52 inh_bulk_n) kpnp area=2
    Q4 (net29 net28 net66 inh_bulk_n) knpn
    Q10 (net63 vbias net16 net16) knpn
    Q6 (net66 inm net63 inh_bulk_n) knpn
    Q11 (net97 vbias net18 net18) knpn area=1
    Q14 (net29 net74 net87 inh_bulk_n) knpn
    Q25 (out vbias net24 net24) knpn area=8
    Q21 (net78 vbias net26 inh_bulk_n) knpn area=2
    Q22 (net102 vbias net22 inh_bulk_n) knpn area=8
    Q18 (net101 vbias net20 net20) knpn
    Q13 (net87 inp net101 inh_bulk_n) knpn
    Q23 (net90 net78 net102 inh_bulk_n) knpn area=8
    Q24 (net93 net102 out inh_bulk_n) knpn area=areaQ24
    Q8 (net96 net63 net97 inh_bulk_n) knpn area=2
    Q9 (net99 net101 net97 inh_bulk_n) knpn area=2
    C0 (net99 net102) capacitor c=25p
ends bipamp
// End of subcircuit definition.
// Library name: amplib
// Cell name: bipamp_sim
// View name: schematic
i1 (inm inp out net13 0) bipamp
r0 (inm 0) resistor r=1M
r6 (inm out) resistor r=1
r4 (net21 0) resistor r=Rvcvs
v0 (inp 0) vsource dc=v1 mag=vmag phase=0 type=pulse val0=v1 val1=v2 \
        period=Tp delay=td rise=tr fall=tr width=tw
V0 (net13 0) vsource dc=-3.5 type=dc
V1 (vdd! 0) vsource dc=vdd_val type=dc
V2 (vss! 0) vsource dc=-5 type=dc
e0 (net21 0 inp inm) vcvs gain=1.0
cload (out 0) capacitor c=0
// Spectre Source Statements

// Spectre Analyses and Output Options Statements

// Output Options
simOptions options
+       save = allpub
+       currents = all

// Analyses
 dc1 dc oppoint=logfile homotopy=all
 tran1 tran stop=6u method=gear2only autostop=no
 ac1 ac start=1 stop=1G dec=20

//Corners

 alter_ss altergroup {
 include "./models/corner_lib.scs" section=ss
 }

 alter_ff altergroup {
 include "./models/corner_lib.scs" section=ff
 }
// End of Netlist
```

References

[1] Geert Van der Plas, Georges Gielen, Willy Sansen. Survey, A Computer-Aided Design And Synthesis environment for Analog Integrated Circuits, 2000.

[2] Georges G.E. Gielen, Design tool solutions for mixed –signal/RF circuit design in CMOS nanometer technologies, 2007.

[3] Manuel Barros, Jorge Guilherme, Nuno Horta. Analog Circuits Optimization based on Evolutionary Computation Techniques, 2007.

[4] N.C. Horta, "Analog and Mixed-Signal IC Design Automation: Synthesis and Optimization Overview," in Proc. 5th, Conference on Telecommunications, 2005.

[5] F. El-Turky and E. Perry, "BLADES: An artificial intelligence approach to analog circuit design," IEEE Trans. Computer-Aided Design, vol. 8, pp. 680–692, June 1989.

[6] M. Degrauwe et al., "IDAC: An interactive design tool for analog CMOS circuits," IEEE J. Solid-State Circuits, vol. 22, pp. 1106–1115, Dec. 1987.

[7] D. Leenaerts, G. Gielen, R. Rutenbar, "CAD solutions and outstanding challenges for mixed-signal and RF IC design," in Proc. IEEE/ACM International Conference on Computer Aided Design, Nov. 2001, pp. 270-277.

[8] . M. Degrauwe et al., "IDAC: An Interactive Design Tool for Analog CMOS Circuits," IEEE J. Solid-State Circuits, vol. 22, pp. 1106–1115, Dec. 1987.

[9] . R. Harjani, R. Rutenbar, L. R. Carley, "OASYS: A Framework for Analog Circuit Synthesis," IEEE Trans. Computer-Aided Design, vol. 8, pp. 1247–1265, Dec. 1989.

[10] C. Leme, N.C. Horta, J.E. Franca, A. Yufera, A. Rueda, J.L. Huertas, L.Paris, T.Oses, "Flexible Silicon Compilation of Charge Redistribution Data Conversion Systems," in Proc. IEEE /Midwest Symposium on Circuits and Systems, 1991, pp. 403-406.

[11] N.C. Horta, J.E. Franca, C.A. Leme, "Framework for Architecture Synthesis of Data Conversion Systems Employing Binary-Weighted Capacitor Arrays," in Proc. IEEE International Symposium on Circuits and Systems, June, 1991, pp. 1789-1792.

[12] .H.Y. Koh, C.H. Sequin, P.R. Gray, "OPASYN: A Compiler for CMOS Operational Amplifiers," IEEE Trans. Computer-Aided Design, vol. 9, no. 2, pp. 113-125, Feb. 1990.

[13] J.P. Harvey, M.I. Elmasry, B. Leung, "STAIC: An Interactive Framework for Synthesizing CMOS And BICMOS Analog Circuits," IEEE Trans. Computer-Aided Design, vol. 11, no. 11, pp. 1402-1417, Nov. 1992.

[14] P.C. Maulik, L.R. Carley, "Automating Analog Circuit Design Using Constrained Optimization Techniques," in Proc. of IEEE Int. Conf. Computer-Aided Design, Nov 1991, pp. 390-393.

[15] E.S. Ochotta, R.A. Rutenbar, L.R. Carley, "Synthesis of High-Performance Analog Circuits in ASTRX/OBLX," IEEE Trans. Computer- Aided Design, vol. 15, no. 3, pp. 273-294, Mar. 1996.

[16] M. Hershenson, S. Boyd, T. Lee, "GPCAD: A Tool for CMOS Op-Amp Synthesis," in Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD), 1998, pp. 296–303.

[17] F. Medeiro, B. Perez Verdu, A. Rodriguez Vazquez, and J.L. Huertas, "A Vertically Integrated Tool for Automated Design of Modulators," IEEE Journal of Solid-State Circuits, Vol. 30, No. 7, July 1995.

[18] F. Medeiro et al., "A Statistical Optimization-Based Approach for Automated Sizing Of Analog Cells," in Proc. ACM/IEEE Int. Conf. Computer-Aided Design (ICCAD), 1994, pp. 594–597.

[19] A. Torralba, J. Chavez, L. G. Franquelo, "FASY: A Fuzzy-Logic Based Tool for Analog Synthesis," IEEE Trans. Comput.-Aided Design Integrated Circuits, vol.15 (7), pp. 705–715, 1996.

[20] M. Krasnicki, R. Phelps, R. Rutenbar, and L. R. Carley, "MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells," in Proc. ACM/IEEE Design Automation Conf. (DAC), 1999, pp. 945–950.

[21] W. Kruiskamp, D. Leenaerts, "DARWIN: CMOS Opamp Synthesis By Means of A Genetic Algorithm," in Proc. ACM/IEEE Design Automation Conference (DAC), 1995, pp. 550–553.

[22] G. Alpaydin, S. Balkir, G. Dundar, "An Evolutionary Approach to Automatic Synthesis of High-Performance Analog Integrated Circuits," IEEE Trans. on Evol. Computation, vol.7(3), pp. 240–252, 2003.

[23] F. de Bernardinis, M.I. Jordan, A. SangiovanniVincentelli, “Support Vector Machines for Analog Circuit Performance Representation,” in Proc. Design Automation Conference, 2003, pag(s):964 – 969.
[24] G.A. Wolfe, “Performance Macro-Modeling Techniques for Fast Analog Circuit Synthesis,” Ph.D. dissertation, Dept. of Electrical and Computer Engineering and Computer Science of the College of Engineering, University of Cincinnati, USA, 1999.
[25] Analog Design Automation, Inc.
[26] .Cadence Inc., Products: Composer, Virtuoso, DIVA, NeoCircuit, NeoCell, UltraSim, NcSim.
[27] .W. Nye, D.C. Riley, A. Sangiovanni-Vincentelli, and A.L. Tits, “DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits” IEEE Trans. Computer-Aided Design, vol. 7 no. 4, pp. 501-519, April 1988.
[28] R. Phelps, M. Krasnicki, R. Rutenbar, L. R. Carley, and J. Hellums, “ANACONDA: Simulation-Based Synthesis of Analog Circuits via Stochastic Pattern Search,” IEEE Trans. on CAD, vol. 19, no.6, pp. 703-717, 2000.
[29] PhD Thesis, Manuel Fernando Martins de Barros, Analog Circuits and Systems Optimization based on Evolutionary Computation Techniques.2009
[30] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "IPRAIL: Intellectual property reuse based analog IC layout automation," Integration, the VLSI Journal, vol. 36, no. 4, pp. 237-262, Nov. 2003.
[31] I. Baturone, Á. Barriga, S. Sánchez –Solano, C. J. Jiménez – Fernández, D.R. López Microelectronic design of fuzzy logic - based systems.
[32] . G. E. Gielen, Senior Member, IEEE, ROB A. RUTENBAR, “Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits”, PROCEEDINGS OF THE IEEE, VOL. 88, NO. 12, DECEMBER 2000.
[33] . Liu, Bo; Fernandez Francisco, G. E. Gielen , Castro-Lopez, Rafael Roca, Elisenda A Memetic Approach to the Automatic Synthesis of High Performance Analog Integrated Circuits
[34] Cadence Manual 2007, Spectre® Circuit Simulator Reference
[35] Phillip E. Allen, Douglas R. Holberg, Book: CMOS Analog Circuit Design
[36] Manuel Barros, Jorge Guilherme, Nuno Horta. “GA-SVM Optimization Kernel applied to Analog IC Design Automation”, 2006