A modular approach for simulation-based optimization of MEMS

Peter Schneider*, André Schneider1, Peter Schwarz2
Fraunhofer Institute for Integrated Circuits, Design Automation Division EAS, Zeunerstraße 38, 01069 Dresden, Germany

Received 31 January 2001; accepted 21 August 2001

Abstract

The importance of MEMS optimization concerning performance, power consumption, and reliability is increasing. A variety of specialized tools is available in the MEMS design flow. FEM tools (ANSYS, CFD-ACE+) are widely used on component level. System level simulations are carried out using simplified models and simulators like Saber or ELDO. There is a lack of simulator-independent optimization support. Only a few simulators offer optimization capabilities.

Our approach aims at a flexible combination of simulators and optimization algorithms by partitioning the optimization cycle into simulation, error calculation, optimization and model generation. This new method is translated into a modular optimization system named Mosquito implemented in Java. Several optimization algorithms are available: methods without derivatives, methods using derivatives and stochastic approaches. Interfaces to simulators like ANSYS, Saber, MATLAB are implemented. Thus the optimization problem can be handled on different levels of model abstraction (FEM, ordinary differential equations, generalized networks, block diagrams, etc.). A graphical user interface supports control and visualization of the optimization. The modules of the optimization system are able to communicate via the Internet to allow a distributed, web-based optimization.

Keywords: Optimization; Microsystems; Heterogeneous systems; MEMS; Modular optimization system; Distributed optimization via Internet; Multi-level simulation; Finite element method; System simulation

1. Introduction

In the past, design optimization of technical systems was usually based on the experience of the designer, who analyzes the system performance and modifies individual system parameters. This kind of optimization becomes more and more difficult with an increasing number of design parameters. That is why the use of numerical methods for parameter optimization combined with simulation is a promising approach to overcome this problem. The most important task to apply optimization algorithms is to transform the technical problem, which is to be optimized, into a mathematical optimization problem. The formulation of the objective function, i.e. the measure for the deviation of the actual system behavior from the desired one, is difficult, especially if there are diametrical optimization criteria. Often one is faced with a lot of constraints regarding parameter values or system properties.

A lot of numerical methods are known for an effective search of an optimal parameter combination, which represents the desired system behavior. Besides ‘classical’ methods (direct search methods without calculation of derivatives, algorithms using derivatives) for finding a set of optimal parameters, there are methods based on stochastic approaches or empirical knowledge. Approaches like genetic algorithms or evolutionary strategies are well known and used for optimization problems with a huge number of parameters. Despite this variety of available methods for the application of numerical optimization algorithms the following problems persist:

- the transformation of the general optimization task into a mathematical problem;
- the description of the objective function;
- optimization criteria, which are often opposite to each other (multi-criteria optimization, poly-optimization);
- the choice of a suitable algorithm; and
- probably the most important problem, finding the global optimum.

In addition, there are problems that are essentially due to the system class to be optimized. Specific problems of microsystem optimization are discussed and an approach
for simulation-based optimization in this field is introduced in the following sections.

2. A modular approach for the optimization of microsystems

Complex systems in microelectronics, or microsystem technologies have to fulfill a variety of requirements regarding system performance, power consumption, reliability and production costs. These properties have to be considered in the design process usually at very early stage. Computer-based methods are widely used in this field. Especially simulation is a very important technique for a fast and low-cost analysis of different versions of a design and the estimation of the system behavior.

 Usually microsystems are very heterogeneous. Different physical domains and different levels of abstractions — from device to system level — have to be considered in the design process (Fig. 1).

A variety of simulations has to be performed depending on the level of abstraction:

- FEM calculations in different physical domains;
- modeling and simulation of subsystems; and
- simulations on system level (including mechanics, optics, electronics, etc.).

These different kinds of calculations cannot be performed always with a single simulator. Often it is necessary to couple specialized tools for an appropriate analysis of a subsystem [1].

Furthermore, one is faced with different mathematical descriptions of the system on different levels of abstraction (e.g. partial differential equations on device level, ordinary differential equations — differential algebraic equations on system level). For that reason in microsystem design there is a strong need for modeling and modeling methods (Refs. [2–4] and others), unified modeling approaches [5], standardized languages [6], automated model generation [7], and optimization capabilities.

The number of design variables may be high in microsystems and due to nonlinearities the shape of the objective function can be extremely complex. The different optimization methods behave differently depending on the shape of the objective function. It is useful to apply different algorithms to a specific optimization problem.

The following characteristics of microsystems have to be considered:

- coupled physical domains;
- different simulators and design tools for the physical domains (and their coupling, if necessary);
- different levels of abstraction;
- changing between the levels of abstraction in the design process; and
- different tool environments (e.g. integration in design frameworks).

Different tools and methods dealing with this characteristics have to be integrated into the optimization cycle. It should be possible to exchange them in an efficient and user-friendly manner.

2.1. The optimization cycle

The main idea of our approach is to divide the optimization process into distinct subtasks or modules. The calculation of the system behavior is an essential task for simulation-based optimization. Regarding the characteristics of microsystems (or other heterogeneous systems) discussed above, this calculation is mostly done using a simulator.
The optimization algorithm is another important module. A new set of parameters is calculated based on a value, which represents the deviation of the actual from the desired system behavior (objective function). There are several possibilities to connect simulators and optimization methods.

Some simulator vendors offer optimization features for their tools [8–10]. Thereby optimization algorithms are called from the simulation system (Fig. 2, left). The definition of the optimization problem is done in the vendor specific environment. The use of other simulators is usually not supported.

Optimization systems, mostly with their own modeling languages [11,12], support the execution of external routines in the optimization cycle (Fig. 2, right). Usually this routine is a function (written in C or FORTRAN) for calculating an error value. It may be also a complex code, like a simulator. Mostly the use of external optimization algorithms is insufficiently supported by these systems.

The amount of available optimization algorithms, the variety of simulators used in the field of microsystem technologies and the specific needs of their design process led us to a third approach [13,14]. The optimization cycle was divided into modules: model generation, simulation, error calculation and optimization (Fig. 3, see also Section 2.2) considering the following requirements:

- simple integration of optimization algorithms and simulators;
- modularity, in order to exchange algorithms and integrate problem-specific tools;
- standard or quasi-standard interfaces between modules;
- interactive control capabilities to integrate designer experience and decisions into the optimization;
- visualization of the optimization progress to allow interactive control of the optimization.

These requirements and the above-mentioned characteristics of microsystems are important for a wide spectrum of other technical systems. The modular optimization system Moscito (modular system for constraint nonlinear microsystem optimization) was developed considering the requests defined above. Thus, a broad area of application arises.

The modular structure of the optimization cycle decreases the overall effort for optimization tasks. The user has always to formulate the optimization problem independent from the used simulator and the applied optimization algorithm. One has to specify:
• the initial values for the parameters;
• the desired system behavior (specification);
• restrictions for system variables; and
• bounds on the parameters.

Additionally the user has to provide a system model (mostly simulator specific) and the control parameters for the optimization algorithm.

The main structure of the modular optimization system Moscito is represented in Fig. 4. Each subtask is encapsulated in a module and is executed within the optimization cycle. Informations are exchanged between the modules using an uniform interface.

2.2. Modules of the optimization system

The modules of the optimization system Moscito are described in Section 2.2.1. First the function of each module is introduced. Section 2.3 deals both with the implementation of the modules using Java programming language and with the Internet aspect of Moscito.

2.2.1. Model generation

The model which is to be optimized contains degrees of freedom in the form of freely adjustable parameters. Such a model is called a generic model. This model has to be supplied with a set of parameter values for the actual optimization step before each simulation. At the beginning of the optimization cycle this has to be done using the initial values specified by the user. In the optimization process new parameter values are supplied by the optimization module as a result of the actual optimization step.

The kind of parameter transfer to the simulator depends on its possibilities. Many simulators need a model formulated in a specific language (e.g. an ANSYS APDL-File, a netlist for electrical circuits or a behavioral description). In that case, the generic model file needs to be modified in every optimization step. Therefore, the model source code (e.g. ANSYS system description, Spice netlist) contains dummy variables, which are replaced with the actual parameter values during the model generation. Other simulators support the modification of model parameters in the simulator database. In this case a one-time reading of the model description at the beginning of the optimization process is sufficient. If a new record of parameter values is present, they are modified in the simulator database using appropriate access mechanisms. The model generation module encapsulates these different possibilities and offers an uniform interface for the user.

2.2.2. Simulation

The calculation of the actual behavior of the system is performed by the module simulation. It consists of starting the simulator with appropriate parameter values, in some cases the execution of special tools for visualization, data logging or other tasks. For the actual values of the optimization parameters, the system model is built up in the simulator database (or modified, respectively) and is calculated with the solution algorithm of the simulator. A uniform (simulator-independent) interface is used for data exchange with the neighboring modules of Moscito.

The extraction of simulation results from the simulator database is usually performed using the simulator functionality. That is why the amount of data and the data structure are different for each simulator. Thus, it is necessary to implement a converter for each simulator and output format. These converters have to transform the simulation results into the data exchange format of the next module error calculation.

2.2.3. Error calculation

The module error calculation determines the deviation of the current simulation results (i.e. the actual system...
behavior) from the specification (i.e. the desired system behavior) given for several system variables. A specification may be either the definition of upper and lower limit of an admissible tolerance area for a variable or a strict demand. Both can be provided analytically or numerically. Furthermore, a rating of the specification for distinct variables can be specified by weighting factors.

The deviation $e(t)$ between the actual and desired behavior of the system can be calculated as an absolute or a relative error value for given sample points. With an evaluation function (e.g. quadratic sum, sum of absolute values or minimum function) a scalar quality measure is extracted from the total of all deviations. This measure is the value of the objective function which is to be maximized or minimized.

2.2.4. Optimization

The module optimization determines the optimization parameters considering linear or nonlinear restrictions. The objective function reaches a local or the global minimum or maximum, respectively. So far the following optimization algorithms [15–17] are available in the standard optimization module of Mosquito called Opal:

- Quasi Newton method;
- Conjugate Gradient algorithm;
- the method developed by Powell;
- Nelder–Mead Simplex-Algorithm; and
- the stochastic search method Simulated Annealing.

The strategies of the algorithms for determination of the optimum differ heavily. Comparison results for different procedures using mathematical test functions are given in [18]. Linear or nonlinear constraints are considered for all algorithms using penalty functions. The penalty function sets up a term, which punishes leaving the admissible area defined by the constraints. The penalty is added to the objective function and forms a modified optimization problem.

2.2.5. Visualization (graphical user interface)

The values of the optimization parameters, the objective function and the actual vs. the desired system behavior can be observed during the optimization cycle using the front-end of Mosquito. The parameter values and the values of the objective function are displayed for each iteration in order to compare the typical runs of different optimization algorithms. Simulation results, i.e. time plots or characteristic curves are displayed for each simulation. If a simulation results is a scalar value it is displayed as a function of the number of iterations. Additionally the given specifications (e.g. tolerance areas) are visualized.

2.3. Implementation as an Internet-based optimization system

Mosquito was implemented regarding the following aspects:

- Encapsulation of design tools and adaptation of tool-specific control and data input/output.
- Communication between tools for data exchange and distributed, Internet-based operation.
- Uniform graphical front-end used for configuration of tools, control of optimization process and visualization of results.

One important goal is to provide the functionality of a tool (e.g. Saber: circuit simulation) as a service in a local area network (LAN) or in the entire Internet. The user should be able to assemble all the services into a problem specific workflow. The tools need not to be installed on the users local computer. The availability of the services via the net is completely sufficient. The effort for installation, configuration and maintenance of software will decrease due to that fact. Furthermore, specialized tools can be executed on a system providing a high performance (e.g. supercomputer with fast CPUs and large memory, Workstation-Cluster). Remote computing in this way is important for applications with a huge amount of computing time: simulation as well as optimization.

The Mosquito framework was implemented using Java [19] and can be used on the different computing platforms — e.g. SUN workstation (Solaris) and PCs (Microsoft Windows and LINUX).

2.3.1. Tool encapsulation

An additional software layer was realized as a special agent interface (MosquitoAgent) for the integration of design tools (simulators, optimization algorithms) into Mosquito. This interface has to carry out the following tasks:

- adaptation of input data to the embedded tool, e.g. generation of configuration scripts or input files;
- adaptation of output data, i.e. conversion of the tool-specific data formats (simulation results, log-files);
- mapping of control information to the embedded tool, transfer and conversion of status informations (warning and error messages), which have to be submitted to the user.

To provide the opportunity of the integration of a broad spectrum of tools and use them as a service in Mosquito there are three ways for embedding programs into a Mosquito agent:

- integration of the entire program;
- embedding of a library via the Java Native Interface (JNI); and
- the direct integration of Java-classes and applications.

Up to now the simulators Saber [20], ELD0 [21], Spice [22], KOSIM [23], ANSYS [8], MATLAB [24], the optimization modules Opal, MORDOS [25] and programs for
flexible comparison of simulation results and their specification are available for Moscito.

Encapsulation of the tools as a Moscito agent guarantees an uniform interface to the framework. All tool specific details are collected in a special agent description files. These files are used to create tool specific dialogs for configuration of the tool via the front-end program.

2.3.2. Communication

The implementation of the tool communication is based on TCP/IP-sockets. Thus, the tools can be executed on different computers or on different computing platforms in a LAN or in the Internet. Of course Moscito can be used locally, e.g. on one computer. In this case, all the necessary tools have to be installed on that computer.

The format of data which has to be submitted is a complicated aspect of communication. Usually it is necessary to adapt or convert both input and output data for each tool. The format for all data transmitted in Moscito was set to a special XML-Format \[26\], the Moscito Markup Language (MoscitoML) to decrease the implementation effort for parsers and converters.

2.3.3. Graphical user interface (GUI)

A uniform and consistent concept for the user interaction is a very important feature of any software, which can be crucial for the user acceptance. For that reason Moscito provides a graphical front-end (Fig. 5) for:

- description of the optimization problem (models, specification, initial values, configuration);
- choice and configuration of tools for a special workflow from the set of available services;
- control of optimization cycle;
- display of messages generated by tools (observation of the proper operation, trouble shooting);
- display of all result data during optimization process (parameter values, objective function).

The graphical front-end aims at the use of design tools via the Internet in a simple and efficient manner. At the moment the front-end is available as a Java application. A future extension of Moscito will be a front-end written as Java applet to provide it via the Internet.

3. Examples for system optimization

Moscito was used for the optimization of a couple of systems. The following examples are within the wide area of heterogeneous systems, which covers the field of microsystems or MEMS. They clarify various possibilities for the calculation of system behavior. A FEM-simulator (ANSYS) was used for the force-sensing unit. The optimization of a micropump was performed using a network modeling approach and the circuit simulator Saber. MATLAB was used for the ionization chamber.

3.1. Optimization of a deformation unit for a force sensor system

Moscito was used to optimize deformation units of force sensors developed and produced by AST GmbH, Dresden.
Germany (Fig. 6). Resistive wire strains determine the mechanical deformation and thus the force, which is applied at the center of the sensor. They are located at the base of the inner notch of the sensor (Fig. 7, left and right). Their resulting strains are less than 1 per mill.

The run of the strain curve at the base of the inner notch of the original sensor is shown in the middle of Fig. 7.

The following characteristics are subject to optimize:
- the values of strain and buckling should be roughly the same in their magnitude;
- the magnitudes of this two values should reach the 1 per mill mark;
- the strain curve at the resistive wire strains should have minimal curvature and slope for a low sensitivity due to mounting tolerances.

The positions of the lower points of the outer notch are optimization parameters. They should have the same depth and should be within the outer limits of the notch.

Using Moscio the following improvements could be achieved (Fig. 7):
- the values of strain and buckling at the resistive wire strains are almost identically;
- the run of the strain curve at these points has lower curvatures and slopes;
- the measurement range is larger (strain and buckling of more than 0.9 per mill).

This optimized cross-section of the force sensor is shown in Fig. 7 (left). The results discussed in this section are achieved using the Nelder–Mead Simplex-Algorithm in about 50 iterations.

3.2. Optimization of a micropump [27]

The micropump shown in Fig. 8 is produced by GESIM (Dresden, Germany) using an anisotropic etching process. A piezoelectric element on the pump chamber membrane is driven by an electrical voltage. This causes a deformation of the membrane and a flow of fluid through the channels. Due to the extremely nonlinear behavior of the outlet the pump ejects droplets of about 1 nl. The micropump is used in analyzing and proportion systems for chemical and pharmaceutical applications.

Subject of the optimization is to maximize the fluid volume ejected by the pump. The parameters for the
optimization are:

- $t_1$ length of electrical impulses;
- $T$ the distance between two impulses;
- $R$ the width of the outlet; and
- $l$ the length of the outlet.

The fluid volume ejected by the pump was increased by 10% as a result of the optimization using Moscito. This result was achieved using Nelder–Mead Simplex-Algorithm in about 40 iterations.

Fig. 9 shows the behavior (volume at pump outlet) before and after the optimization. It is to be seen, that the number of droplets was increased, the distance between the droplets and the volume per droplet was decreased.

3.3. Optimization of a ionization chamber [28]

Ionization chambers are used for thickness and area mass measurement and for edge control in the production process of material sheets like steel or paper. The measurement system consists of a $\gamma$-radiation source and a radiation detector (Fig. 10). The detector contains a distinct number of cells, which are filled with rare gas. $\gamma$-quanta are absorbed in the cells and produce pairs of electrons and ions which are detected by the measurement of an electrical current.

Due to the very low radiation intensity a relatively low current is measured in the cells situated beneath the material. Typical values of the current in these areas are from $10^{-14}$ up to $10^{-10}$ A.

The optimization task is to achieve a large difference between the currents in the cells beneath the material and the cells which are not covered. The absolute values of all currents should be as high as possible. Furthermore, the resolution and the dynamic characteristics of the measure have to meet customer specifications. Additionally, there are some physical and technological constraints regarding the pressure in the chamber ($< 5$ bar), the voltage ($< 1000$ V), the ionization energy ($< 30$ eV), the height of the chamber ($< 150$ mm) and the electrical field strength ($< 10^{-9}$ V/m).

Parameters for the optimization are:

- distance between the electrodes: $a = 3–15$ mm
- thickness of the shielding: $d = 0.5–5$ mm
- pressure: $p = 1–5$ bar.

The model for the calculation of the absolute values of the cell currents and the current differences was implemented as a MATLAB m-file.

The optimized design (Fig. 11) is characterized by an electrode distance of $a = 5.38$ mm and a thickness of shielding of $d = 0.65$ mm. These values represent an optimal set of parameters for a given specification. For the pressure there is a limitation due to the mentioned constraint. The absolute values of the current have been raised as well as current differences and slopes of the splines in Fig. 11, respectively.
4. Conclusions

An approach for efficient Internet based optimization of heterogeneous systems was introduced. Dividing the optimization cycle into specialized modules using an uniform interface decreases the overall effort for solving optimization problems. Due to the modular structure a multitude of different simulators, optimization algorithms, error calculation routines and other tools can be assembled into a workflow and can be exchanged in a fast and efficient manner. This approach meets a lot of the important requirements of the design of microsystems and other heterogeneous systems.

Acknowledgements

The presented work was done in projects funded by the German Government: DEMIS — Designoptimierung fuer
References


