5 Analog IC Design Environment Architecture

Abstract. This chapter describes the implementation of an innovative design automation tool, GENOM which explores the potentials of evolutionary computation techniques and state-of-the-art modeling techniques presented in the previous chapters. The main design options of the proposed approach will be here described and justified. First, an overview of the design architecture main building blocks will be provided. Then, the optimization algorithm kernel, as well as, the implemented functionalities are described. Finally, the design options are described in detail using experimental results on a few test cases.

5.1 AIDA Architecture

The GENOM optimization tool can be used as a standalone application, although it holds some functionality which can only be fully accomplished when it is part of the in-house design automation environment called AIDA [1]. AIDA, Analog Integrated Design Automation, is an ongoing project for analog IC design automation at ICSG group IT/IST. A summary of this application architecture will be described next.

5.1.1 AIDA In-House Design Environment Overview

The AIDA platform, which includes a design flow core engine responsible for the design automation is illustrated in Fig. 5.1. The platform is structured in three layers: interface, application and data layer and implemented in several technologies, such as JAVA® for the design core, MySQL® for the databases and Swing® for the graphical user interface (GUI). The AIDA project implements a fully configurable design flow which introduces an increased level of flexibility and reusability when compared to traditional design approaches. The flexibility is achieved by both allowing the designer to define his own hierarchical design organization and, simultaneously, the design flow for each design. The reusability is achieved by introducing a highly organized data structure to store the entire design data allowing an easy reuse and retargeting of pre-design systems and predefined design flows. In addition, AIDA allows the interaction with other CAD tools such as circuit and system level optimizers like GENOM and layout generators [2-3].

M.F.M. Barros et al.: Analog Circuits and Systems Optimization, SCI 294, pp. 109–137.
springerlink.com

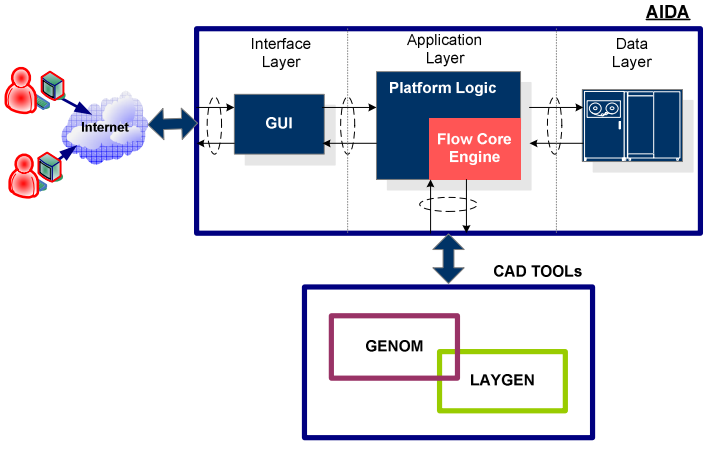


Fig. 5.1 Conceptual view of AIDA environment architecture

The AIDA platform implements a hierarchical methodology matching designers' approach by allowing the complete definition of the design flow tasks at each hierarchical level, as presented in Fig. 5.2 for a filter design case. The design flow definition is based on basic units of work: project specifications, topology selection, several units for device sizing and optimization and a last unit for characterization. In this project, GENOM acts as an external circuit and system level optimizer tool with well defined interface protocol.

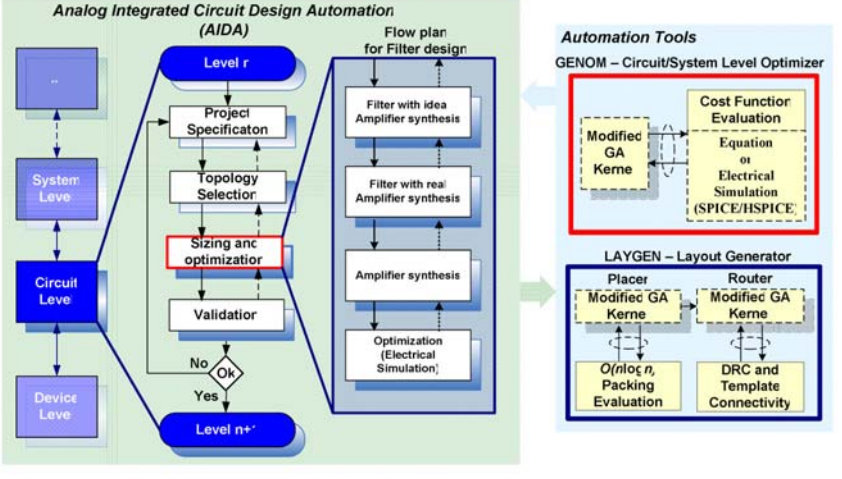


Fig. 5.2 AIDA design flow

The GUI facility of the AIDA platform, illustrated in Fig. 5.3, plays a key role in the definition of project specifications and topology selection required by GENOM.

Through an intuitive user-friendly interface, the user specifies the design specs e.g., circuit class, performance specs, design constraints and technology. These specs, which may be introduced by the user or result from the synthesis in a higher hierarchical level, automatically restrict the set of available topologies. Then, the topology selection may be performed manually by the designer or automatically by an engine (if available) that evaluates the candidate topologies according to design specs. Next, the design flow, organized in several design stages, controlling the optimization process, as exemplified in Fig. 5.2, is defined and executed. Each design stage has the goal of setting a subset of the design parameters (*W*, *L*, *C*, *R*, etc). Therefore, each design stage corresponds to an optimization task submitted to the selected optimization engine, in our case the GENOM optimization engine, using HSPICE, to compute the design objective function. Moreover, the use of other design and simulation tools, if available, is also possible and only depends on user's selection. Although a design stage is considered an atomic operation for the user, during the design flow and at each control point between design stages, he may evaluate the design and redefine parameters, constraints or even change the predefined design flow.

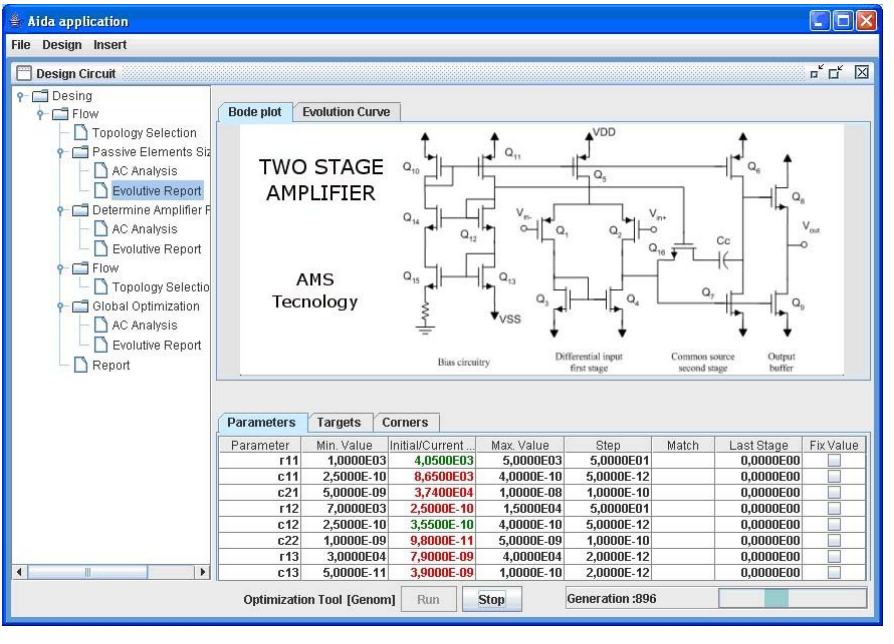


Fig. 5.3 GUI facility implemented in AIDA

5.1.2 Layout Level Tools

The AIDA framework was designed to interact with CAD tools of different hierarchical levels as described in the preceding section for the case of the analog circuit optimizer. In the future, this interaction will be expanded to the layout level for the layout verification and generation. In particular, the objective is to integrate the LAYGEN [2-3] tool illustrated in Fig. 5.4.

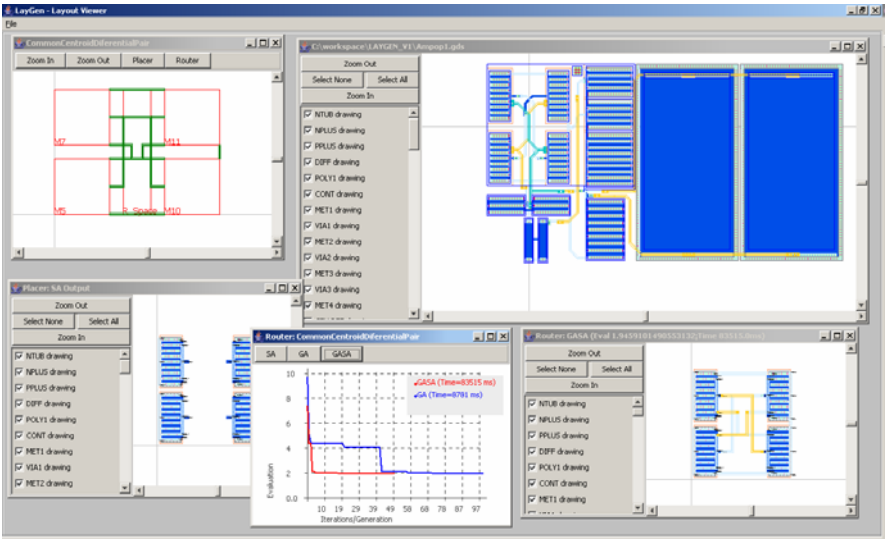


Fig. 5.4. LAYGEN graphical interface

The integration of the layout CAD tool in AIDA framework will allow the inclusion of extracted layout parasitics and circuit reliability design rules, to be taken into account during the design process. The design process now supports the compensation of layout parasitics implementing an iterative loop, involving circuit sizing and the layout generation. Hence, the conformity of analog design specification will be verified taken into account the parasitics of physical implementation.

5.2 GENOM System Overview

The proposed design optimization tool represents an alternative to the traditional design flow, automating some steps of the design methodology. It covers some of the most time consuming tasks of analog design process at the circuit level, like circuit sizing and design trade-offs identification. The main building blocks of GENOM architecture depicted in Fig. 5.5 are decomposed into three units, the optimization kernel, the evaluation module and the application interface.

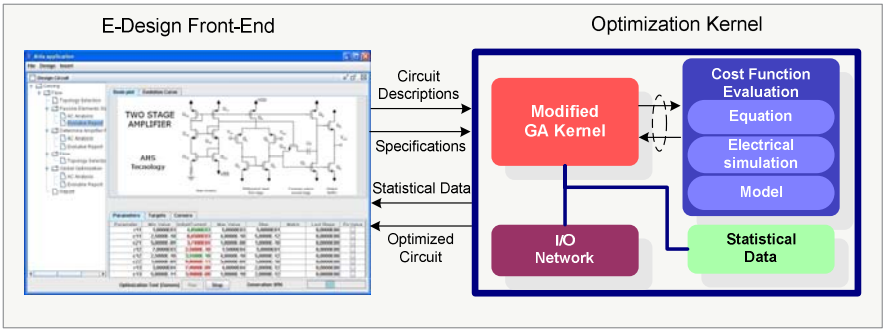


Fig. 5.5 E-Design environment architecture

The GENOM optimizer kernel is based on an evolutionary algorithm (EA) kernel with modified operators and an automatic control mechanism which supports the interaction with equation and simulation evaluation engines, so that the cost function evaluation is made either by behavioral models based on SVM or by electrical simulation, in this case, using Spice-like simulators. Additionally, GENOM includes a distributed processing facility with a high degree of portability across a variety of machines, allowing the increase in computation efficiency when using cost expensive evaluations.

The GENOM core is written in C, programming language, and implemented in a Linux environment, taking advantage of the efficiency and flexibility of C code, free development tools and platform. Although it is commonly used for algorithm development, C language has not traditionally been used to generate a graphical user interface (GUI) for applications. Hence, the front-end was implemented in an independent language platform, the Java™ using the Swing components.

The tool functionality, extended by the addition of an E-Design front-end allowing an incremental growth of the IC design database and an individual management of each project, will be described in the next sub-sections.

5.2.1 Design Flow

In order to support the analog IC design flow methodology and to provide an efficient data management of the inputs and outputs from GENOM, a new design automation environment was developed as illustrated in Fig. 5.5. Like in many analog design automation environments, before the synthesis there is a preparatory stage where the production of user-defined equations (equation-based), training the learning machine for performance models, or incorporating design constraints take place. The design facilities also include the backtracking of the design process, allowing the user to follow the evolution of the design process dynamically or just reporting the final solutions at the end of the optimization process. This feedback is extremely relevant once it provides the information that the designer needs to detect, identify and understand which are the performance bottlenecks for the circuit that is being designed.

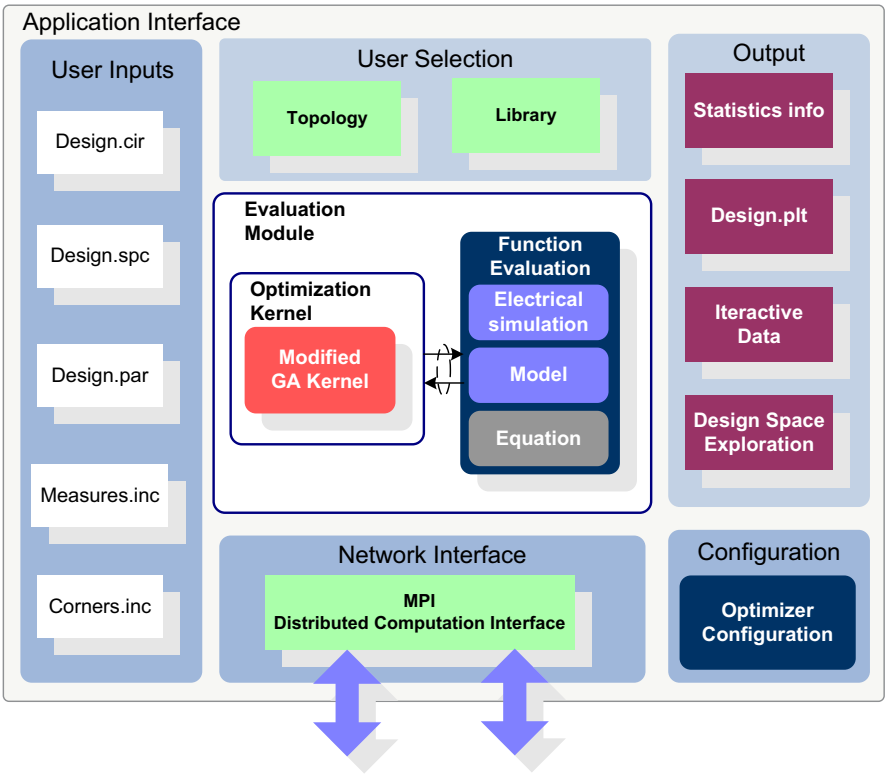


Fig. 5.6 Conceptual view of the Input/Output from optimizer tool

However, when not integrated in the AIDA environment, i.e., in the standalone operation, the user needs to provide and configure manually the necessary input files, depicted in Fig. 5.6, in a suitable form for the optimization process.

5.2.2 Input Data

The aim of this phase is to provide and configure the necessary input files in a suitable form for the optimization process. In order to manage the complex structure of data involved in this project, a graphical interface seems a fairly option to guide all the input data process. The GUI interface, using spreadsheet-like data input forms, aid the designer to input data more easily, minimizing input errors and the setup time to define or redesign a new simulation strategy. In addition, it guides the user through a sequence of logic events and avoids the occurrence of compatibility errors. Through the graphical input interface the user defines the circuit class (amplifier, filter, A/D, D/A, etc), the performance specs (dc gain, gain bandwidth product, phase margin, slew rate, power dissipation, offset voltages, etc) of the analog cell which the designer wants to optimize, as well as the design constraints (corners, matching parameters, overdrive voltages and currents, etc) and the technology process.

Fig. 5.7 illustrates one stage of design specs introduced by the user; in this case it shows the definition of the performance measures required for this project. According to the introduced design specs, a candidate topology is manually selected from the circuit database as depicted in Fig. 5.8. If the design specs do not match any of the existent topologies, a new one have to be created and introduced into the system.

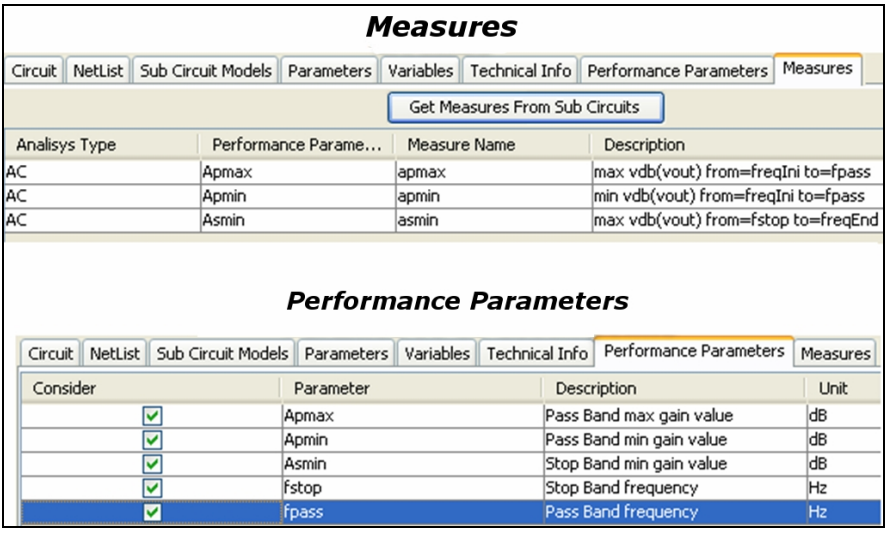


Fig. 5.7 Performance parameters and measures facilities

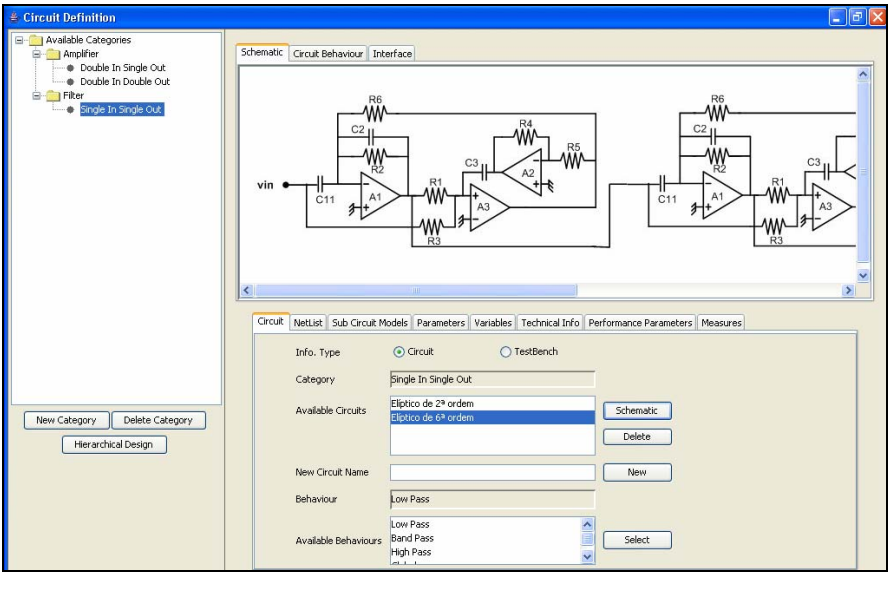


Fig. 5.8 Topology Selection

<design_file>.**cfr** - Configuration file

```
# A line started by a charater "#" is a comment.
<TITLE>
Differential AmPop
Version: November 16, 2007 - Author: F.M. Barros
# ------------------------------------------------------------------------
# 1. Control Parameter Section
# ------------------------------------------------------------------------
<CONTROL>
ProblemType      0     # 0 - Circuit simulation    1- Numerical optimization
OptimizationType 0     # 0 - Genetic algorithms    1- SVM (SA, ...)
# ------------------------------------------------------------------------
# 2-Passing Parameter Section
# ------------------------------------------------------------------------
<PASSING_PARAMETERS>
Seed     99  # SEED - Integer number representing the SEED value ={1-10000}
Timer     2  # TIMER- Simulation time TIMER={SHORT=0, MEDIUM=1, LONG=2}
Quality   2  # Optimization QUALITY={COARSE=0, MEDIUM=1, FINE=2}
Stop      2  # STOP Criterion. STOP={Time=0, Convergence=1, Max_Generations=2}
Debug     1  # DEBUG    - Output text debugging. DEBUG={none=0, YES=1 }
Cluster   0  # CLUSTERS - Parallel Processing ={SERIE=0, PARALLEL=1}
Reports   0  # REPORTS  - Formats {TEXT=0, GRAPHICS=1, Both=2}
Activity 10  # ACTIVITY - Statistics data sampling frequency (for graphics)
StepAC   10  # STEPAC   - Update frequency of bode plots
inDirectory /home/IT/GENOM/workspace/circuits/00_Differential_Ampop
outDirectory /home/IT/GENOM/workspace/circuits/00_Differential_Ampop/RESULTS
# ------------------------------------------------------------------------
# 3-Dependent Parameters Section
# ------------------------------------------------------------------------
<MEASURES>
9
gain_dc;gbw;phfp;phase_margin;ftcmfb;phfpcmfb;phasecmfb;power_a;iavdd
##
<CONSTRAINTS>
34
vov_m0a;vov_m0b;vov_m16;vov_m1a;vov_m1b;vov_m2a;vov_m2b;vov_m3a;vov_m3b
vov_m4a;vov_m4b;vov_m5a;vov_m5b;vov_m6a;vov_m6b;vov_m7a;vov_m7b
delta_m0a;delta_m0b;delta_m16;delta_m1a;delta_m1b;delta_m2a;delta_m2b
delta_m3a;delta_m3b;delta_m4a;delta_m4b;delta_m5a;delta_m5b
delta_m6a;delta_m6b;delta_m7a;delta_m7b
...
```

Fig. 5.9 Partial view of "design.cfr"

At the end of the preparatory phase, five independent text files are created as illustrated in Fig. 5.6. These constitute the configuration files required by GENOM kernel and are briefly described below.

- "*Design.cfr*": This file illustrated in Fig. 5.9 contains the configuration parameters used to control the optimization process, such as, the number of evaluations, the quality of solutions, the stop criterion, type of reports, etc. All the commands used in the configuration file are from the User Guide. This file does not include the commands to modify the behavior of the algorithm kernel. This task is restricted to authorized computer algorithms specialists.

- "*Design.spc*": This file holds the design specifications written in a familiar analog design syntax, using the traditional relational "*min*", "*max*", "less", "*great*",

"*equal*" operators and additional ones for specific constraints expressions such as "*verify_bound(a,b,c)*" illustrated in Fig. 5.10.

- "*Design.par*": The design parameters file depicted in Fig. 5.11 encloses the problem dimension and device names, bounds and step size for each optimization variable.

- "*Design.cir*": This is the circuit netlist file that describes the circuit connectivity either in flattened or hierarchical mode. The optimization variables must be explicit marked with an underscore before the variable's name as depicted in Fig. 5.12. This name must agree with at least one parameter of the design parameters file. The format of this file should be compatible with the evaluation tool.

- "*Corners.inc*": This is an optional input file that specifies the corners conditions. This file showed in Fig. 5.13, will be included in the circuit netlist.

- "*Measures.inc*": This is a user-defined set of statements or commands that retrieve specific electrical measures from evaluation tool. It is a kind of interface between optimizer and the evaluation tool to acquire precise information data. This file, illustrated in Fig. 5.14, will be included in the circuit netlist.

- *Fabrication model*: A fabrication model consists of values for different transistor characteristics needed by the simulator to develop a small signal model for a transistor. In a regular basis, this file is complied with standards and is dependent on the fabrication technology. In GENOM, a library of models aggregates some of the public technological models available. The technological file must be referenced in <Design.cir> file, as illustrated in Fig. 5.15.

<design_file>**.spc** - Specs File

```
   (gain_dc > 55)
+  (gbw     > 100e6)
+  (verify_bound(phase_margin,60,90))
+  (ftcmfb  > 50e6)
+  (verify_bound(phasecmfb,60,90))
+  (min(power_a,0,10e-3))
+  (min(iavdd,0,10e-3))
+  (check_bound(vov_m0a,100e-3,300e-3))
+  (check_bound(vov_m0b,100e-3,300e-3))
+  (check_bound(vov_m1a,50e-3,300e-3))
+  (check_bound(vov_m1b,50e-3,300e-3))
+  (check_bound(vov_m2a,100e-3,300e-3))
+  (check_bound(vov_m2b,100e-3,300e-3))
...
+  (check_bound(delta_m0a,100e-3,1000))
+  (check_bound(delta_m0b,100e-3,1000))
+  (check_bound(delta_m1a,100e-3,1000))
+  (check_bound(delta_m1b,100e-3,1000))
+  (check_bound(delta_m2a,100e-3,1000))
+  (check_bound(delta_m2b,100e-3,1000))
...
```

Fig. 5.10 Partial view of <design.spc>

<design_file>**.par** - Optimization Parameters File

```
21                  # Number of optimization variables of the problem
_w00                # 1st Device name
1.0e-6              # Inferior bound and
300.0e-6            # Upper bound
1.0e-6              # Step size

_L00                # 2nd Device name
0.35e-6             ...
10.0e-6             ...
0.05e-6             ...

_m01                # 3rd Device name
1                   ...
100                 ...
1                   ...
...

_c10                # 21st Device name
1.0e-12             # Inferior bound and
100.0e-12           # Upper bound
1.0e-12             # Step size
...
```

Fig. 5.11 Partial view of <design.par>

<design_file>**.cir** - Circuit Netlist

```
* Differential Ampop Revised: Monday, November 16, 2007
* D:\IT\GENOM\CIRCUITS\AMPOP\AMPOP.DSN      Revision: 1
...
M20 VB3 VB3 N07332 0 nmos  w=_w04 l=_l04 m=1
M21 N07332 VB3 0 0 nmos    w=_w11 l=_l11 m=1
M5A N02095 CMFB 0 0 nmos   w=_w04 l=_l05 m=_m05
M5B N01845 CMFB 0 0 nmos   w=_w04 l=_l05 m=_m05
M7A N11287 N11287 0 0 nmos w=_w04 l=_l05 m='_m05/2'
M7B CMFB N11287 0 0 nmos   w=_w04 l=_l05 m='_m05/2'
*
M6A N11287 VCMI N10772 avdd pmos w=_w02 l=_l06 m=_m06
M6B CMFB   VCM  N10772 avdd pmos w=_w02 l=_l06 m=_m06
*
R10 VOUTP VCMO _R1
*
C10 VCMO VOUTP _c10
*
************************
**** .DATA info  ******
************************
.DATA PIPEdata LAM
FILE = 'PIPE.dat' _w00=1 _w01=2 _w02=3 _w04=4 _w10=5 _w11=6 _l00=7 _l01=8
_l02=9 _l03=10 _l04=11 _l05=12 _l06=13 _l10=14 _l11=15 _m01=16 _m02=17 _m03=18
_m04=19 _m05=20 _m06=21 _R1=22 _c10=23
.ENDDATA

***************************
* setting for AC analysis
***************************
.ac dec 50 1 1e9 SWEEP DATA = PIPEdata
* plot data
.probe ac vdb(voutd) vdb(vcmo) vp(voutd) vp(vcmo)
```

Fig. 5.12 Partial view of <design.cir>

<Corners>**.inc** - Corners File (HSPICE style)

```
* -------------------------
* 1. Corners file
* -------------------------
.ALTER @1 -> lib=slow; temp=-40 +50 +105;
      .protect
      .lib 'cmos035.lib' slow
      .unprotect
      .temp -40 +50 +105

.ALTER @2 -> lib=typ; temp=-40 +50 +105;
      .protect
      .lib 'cmos035.lib' typ
      .unprotect
      .temp -40 +50 +105

.ALTER @ -> lib=typ; temp=-40 +50 +105;  ...
```

Fig. 5.13 Partial view of the Corners file

<Measures>**.inc** - Measures Files (HSPICE commands)

```
* The Measures Section
* -------------------------
* A. Measures for SPECs
* -------------------------
.MEASURE AC 'gain_dc' max vdb(voutd) from=1 to=1000
.MEASURE AC 'gbw' when vdb(voutd)=0 fall=1
.MEASURE AC 'phfp' find vp(voutd) at=gbw
.MEASURE AC 'phase_margin'      PARAM('phfp + 180')
.MEASURE AC 'ftcmfb' when vdb(vcmo)=0 fall=1
.MEASURE AC 'phfpcmfb' find vp(vcmo) at=ftcmfb
.MEASURE AC 'phasecmfb' PARAM('phfpcmfb+180')
.MEASURE AC 'power_a'   PARAM('-P(vdd)/2')
.MEASURE AC 'iavdd'     PARAM('-P(vdd)/avddpar/2')
* ------------------------------------------------
* B. Transistor Bias Measures - overdrive voltage
* ------------------------------------------------
.measure AC vov_m0a    = param('VGS(xampop.m0a)-VTH(xampop.m0a)')
.measure AC vov_m0b    = param('VGS(xampop.m0a)-VTH(xampop.m0b)')
.measure AC vov_m1a    = param('VGS(xampop.m1a)-VTH(xampop.m1a)')
.measure AC vov_m1b    = param('VGS(xampop.m1a)-VTH(xampop.m1b)')     ...

* ------------------------------------------------------
* C. Transistor Transistors Vds voltage margin to VDsat
* ------------------------------------------------------
.measure AC delta_m0a    = param('VDS(xampop.m0a)-VDSAT(xampop.m0a)')
.measure AC delta_m0b    = param('VDS(xampop.m0a)-VDSAT(xampop.m0b)')
.measure AC delta_m1a    = param('VDS(xampop.m1a)-VDSAT(xampop.m1a)')
.measure AC delta_m1b    = param('VDS(xampop.m1a)-VDSAT(xampop.m1b)') ...
...
```

Fig. 5.14 Partial view of the measures file

- Cost Function: This is a module that implements a parser in Lex and Yacc syntax [4] which automatically evaluate the performance of a set of candidate solutions. It is independent from the problem and will be the subject for further discussion in sub-section 5.3.3.1.

<Fabrication>**.inc** - Technology Process File (HSPICE style)

```
**************************
* Libs
**************************
.protect
    .lib '../../library/cmos035/cmos035.lib' typ/slow/fast
           or
    .lib '../../library/UMC/HSPICE/telescopic/l18u18v.122' L18U18V_TT
           or
    .lib '../../library/AMS/hspiceS/c35/wc49.lib' tm/wp/ws
.unprotect
```

Fig. 5.15 Technological model reference

5.2.3 Output Data

The output data provided by the GENOM tool includes the post-processing reports and evolutionary real time reports. The activation of each type of outputs is left to the designer choice. The post-processing reports include the evaluation of performance parameters coupled with statistical information presented at the end of the optimization, using the data in the data structures generated during the optimization phase. Fig. 5.16 and Fig. 5.17 illustrate the type of documentation provided by the design automation environment. The GENOM outputs are divided in two great groups related with design data and process info.

OUTPUT STREAM OF THE SIMULATION DATA FOR ECCTD-2007 CONF.

- PLOT OUTPUT DATA in each run -

#Run	#nEvals	#Fitness	#wTIME	#FEAS	#found_@	#STATUS	#found_@
1	1408	6.590e-02	68.30s	Y	84 (gen)	Y	84 (gen)
2	880	5.766e-02	43.97s	Y	51 (gen)	Y	51 (gen)
3	576	5.617e-02	29.17s	Y	32 (gen)	Y	32 (gen)
4	480	5.354e-02	25.90s	Y	26 (gen)	Y	26 (gen)
5	656	9.821e-02	34.17s	Y	37 (gen)	Y	37 (gen)
1	704	8.186e-02	36.66s	Y	40 (gen)	Y	40 (gen)
2	512	1.147e-01	29.13s	Y	28 (gen)	Y	28 (gen)
3	1120	1.268e-01	71.57s	Y	66 (gen)	Y	66 (gen)
4	912	1.036e-01	47.73s	Y	53 (gen)	Y	53 (gen)
5	784	1.200e-01	58.21s	Y	45 (gen)	Y	45 (gen)

Comparison Among Different Strategies

3_TwoSAMP.cir	Strategy-1	Strategy-2
Taxa Feas	1.0000e+02	1.0000e+02
Taxa Conv	1.0000e+02	1.0000e+02
Cmean	6.62960e-02	1.09392e-01
C-std	1.64832e-02	1.57181e-02
Tmean	4.03020e+01	4.86600e+01
T-std	1.52727e+01	1.51152e+01
Evalmean	8.00000e+02	8.06400e+02
Eval-std	3.31474e+02	2.03546e+02
Rank	1	2

Table II - FINAL STATISTICs - Comparison Among Diferent Algorithms

///////	//////	nEVALs	nEVALs	TIME	TIME	FITNESS	FITNESS
Taxa Feas	Taxa Conv	Mean	Std	Mean	Std	Mean	Std
100.0 %	100.0 %	8.000000e+02	3.314743e+02	4.030200e+01	1.527274e+01	6.629600e-02	1.648328e-02
100.0 %	100.0 %	8.064000e+02	2.035462e+02	4.866000e+01	1.511527e+01	1.093920e-01	1.571815e-02

Table III - PLOT Best Simulation Data for 3_TwoSAMP.cir

STRATEGY	#Fitness	_w1	_l1	_w2	_l2	_w3	_l3	_w4	_l4	_w5	_l5
Strategy-1	5.354e-02	9.00e-06	9.00e-06	8.00e-06	6.00e-06	8.00e-06	1.00e-06	8.00e-07	1.05e-06	5.00e-07	4.00e-07
Strategy-2	8.186e-02	9.00e-06	2.00e-06	1.00e-05	7.00e-06	8.00e-06	8.00e-06	8.00e-07	3.50e-07	5.00e-07	3.50e-07

Fig. 5.16 Progress reports

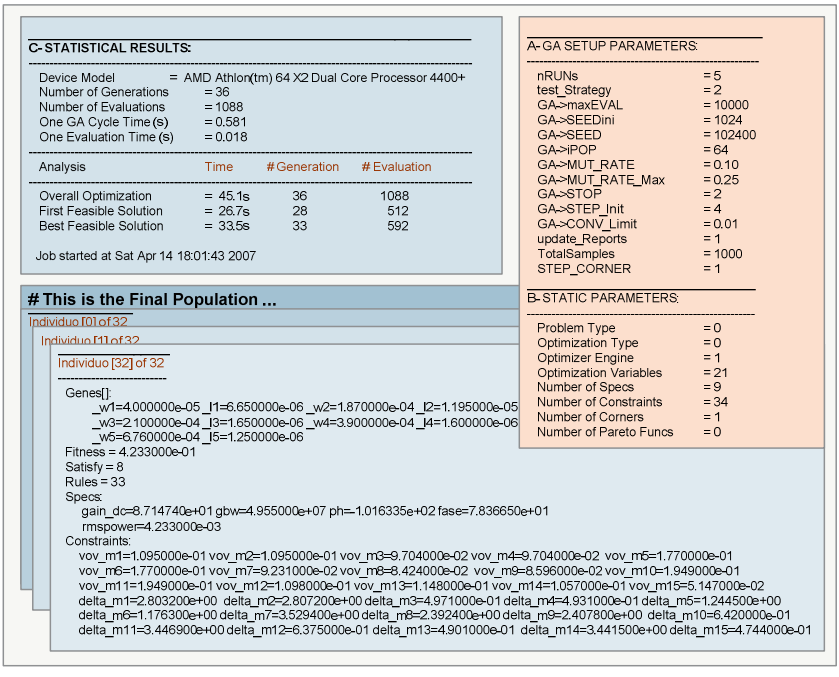


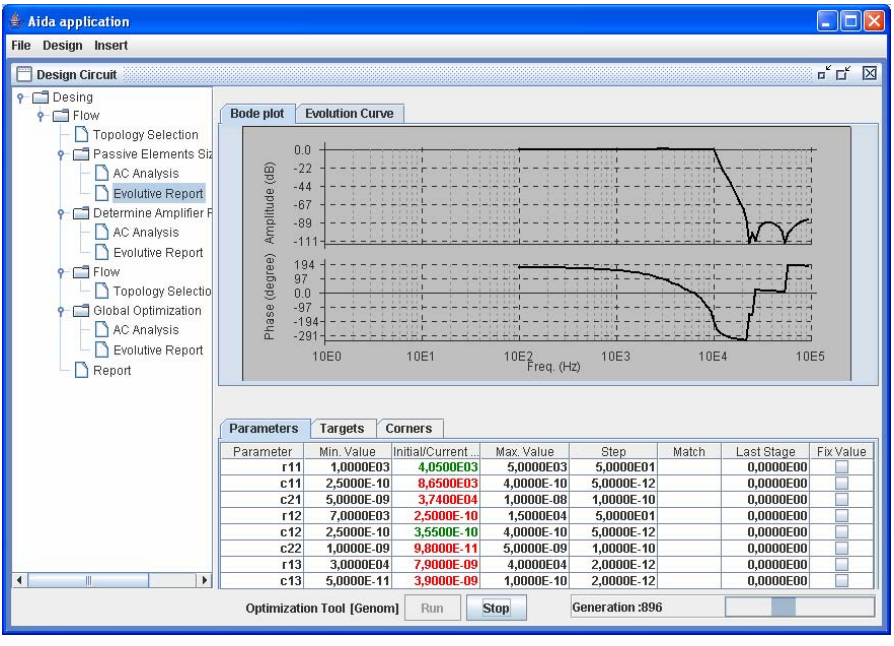
Fig. 5.17 Performance reports from optimization

Process info: This is the union of several statistical metrics gathered from optimization (Fig. 5.16). It includes a huge amount of statistics data about runs, generations, evaluations and time. This data is spread in several thematic files, including the evolution report file, corners file, bode plot file, etc. Optionally, the user can dispatch this info to screen reports for "online" validation purpose as it will be discussed in the next section.

Design Data: This corresponds to the final results from the estimation process (Fig. 5.17). This includes the optimum values of the optimization variables, the performance parameter values and the satisfaction of constraints parameters for the best 32 individuals of the population. In addition, it provides information about the optimization problem progress. These values are confronted with the initial ones to infer about the fulfillment of the synthesis flow objectives.

5.2.3.1 Progress Real-Time Reports

GENOM produces and supplies the required data which allows the visualization of real-time reports in AIDA framework. The progress real-time reports are a set of visual tools available optionally to the user, which indicate the progress status of evolutionary process in each generation. They consist of animated graphics of bode plot figures, the design space exploration figures and of the evolution curve of the cost function. The real-time environment is also represented by a



Parameter	Min. Value	Initial/Current ...	Max. Value	Step	Match	Last Stage	Fix Value
r11	1,0000E03	4,0500E03	5,0000E03	5,0000E01		0,0000E00	
c11	2,5000E-10	8,6500E03	4,0000E-10	5,0000E-12		0,0000E00	
c21	5,0000E-09	3,7400E04	1,0000E-08	1,0000E-10		0,0000E00	
r12	7,0000E03	2,5000E-10	1,5000E04	5,0000E01		0,0000E00	
c12	2,5000E-10	3,5500E-10	4,0000E-10	5,0000E-12		0,0000E00	
c22	1,0000E-09	9,8000E-11	5,0000E-09	1,0000E-10		0,0000E00	
r13	3,0000E04	7,9000E-09	4,0000E04	2,0000E-12		0,0000E00	
c13	5,0000E-11	3,9000E-09	1,0000E-10	2,0000E-12		0,0000E00	

Fig. 5.18 Progress reports provided by the automation prototype

built-in spec sheet that can display a simple pass/fail status, symbolized by green/red colors, of the performance parameters, constraints violations and corners satisfaction as illustrated in Fig. 5.18.

5.2.3.2 Interactive Design

Interactive design is an extended capability introduced to GENOM framework that allows an experienced designer to incorporate some basic knowledge about a circuit during the search process. With the feedback acquired from real-time progress reports, for example, comparing the initial specs against current measured results and taking into account the present context status of the optimization process (state of design variables, evolution curve, constraints violation and corners satisfaction, etc.), the designer can use his knowledge or intuition to change the dynamic ranges of design parameters, set fixed values to genes of the current population (affix some genes of chromosome), etc, which shifts the course of optimization. Keeping constant values in some design variables has the effect of reducing the number of search variables. One equivalent variation of this approach is done by the matching of some strategic transistors such as, the differential pairs, current mirrors, etc., and in some non-sensitive transistors because they do not have much impact on the functionality of the circuit. Both measures result in the

shrinkage of the design space and shortened run times. The advantage of this approach is that it is independent from the process, it captures the designer knowledge and since it adapts to each individual's knowledge, it is more flexible and can lead to efficient performances. Interactive design becomes a valuable optional tool in the presence of an experienced designer.

5.2.4 I/O Interfaces

The MPI interface block illustrated in Fig. 3.24 is composed by two independent types of communications. The hierarchical level interface is dedicated to future integration with LAYGEN tool. The network communication interface implements a local area multi-computer LAM-MPI interface (Fig. 5.19) used in the development of parallel applications over a network of heterogeneous computers as described in sub-Sect. 3.3.7.

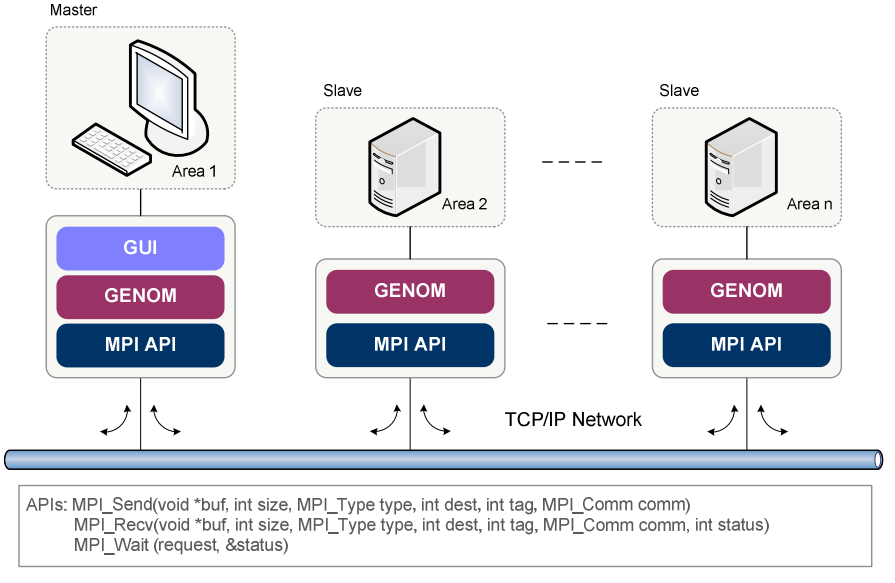


Fig. 5.19 Local area multi-computer system implemented with LAM-MPI

As discussed in 3.3.7, the communication between parallel processes is handled by the Message Passing Interface (MPI). Therefore, it is necessary to download, compile and install the MPI library in the current environment according to instructions in "GENOM Users guide". To make sure that distributed optimization environment is correctly configured and installed in a specific processing node, execute the "test-GENOM" script of Fig. 5.20:

```
#!/bin/bash
# test-GENOM - A script to test remote opt. environment
MACHINE_IP=$1
echo "GENOM: Testing remote machines"
echo "Testing remote GENOM is available …"
ssh  $MACHINE_IP genom
echo "Now, testing if remote simulator is available …"
ssh  $MACHINE_IP hspice
echo "Now, testing SSH to avoid passwords …"
ssh  -x -a $MACHINE -n echo $SHELL
```

Fig. 5.20 Testing GENOM distributed environment

The latter script verifies if the optimization tool, as well as, the evaluation engine are available in a specified processing node by trying to execute an application, e.g."*genom*" and "*hspice*", on all nodes. The last test verifies if the secure "SSH" communications is configured to avoid passwords. If the test is successful, proceed with next sequence of commands to initiate the execution of parallel application, the "*genom*" in the example illustrated in Fig. 5.21.

```
#!/bin/bash
# test-GENOM - A script to activate distributed processing
APPLICATION='/home/genom/Genom/genom filtro.cfr -p'
lamboot -v lamhosts
mpirun C sh.csh $APPLICATION
```

Fig. 5.21 Testing GENOM 'ssh' communications

In the first step, the user creates a file listing ("*lamhosts*") the participating machines in the cluster and then activates the LAM network with "*lamboot*" command. "*Lamhosts*" is a text file that contains the names of the nodes, one per line, with the first one being the machine that the user is currently logged on to.

The activation of GENOM is given by the "*mpirun*" command for the case of a filter optimization problem. With this invocation the application that is being executed has the same pathname on all processor nodes. A more flexible approach is able to run different executable pathname on different nodes. This is achieved through a variation of the "*mpirun*" command and a new definition of "*lamhosts*" as described in Fig. 5.22.

```
#!/bin/bash
# test-GENOM - A script to activate distributed processing
APPLICATION='/home/genom/Genom/genom filtro.cfr -p'
mpirun -p4pg <lamhosts> $APPLICATION
```

Fig. 5.22 Invocation of distributed GENOM application

For example, to run "*genom*" program on machine *baltar*, *malacata* and *everest* all Linux machines, and on *estrela*, a Solaris machine, the <lamhosts> file would contain now the following entries depicted in Fig. 5.23:

```
# a 4-node LAM running 5 processes
baltar   0 /home/PhD/Work/AIDA/GENOM/bin/genom filtro.cfr -p
malacata 2 /home/gneves/AIDA/GENOM/exe/genom filtro.cfr -p
everest  1 /usr/local/linux/GENOM/genom filtro.cfr -p
estrela  1 /home/ngonc/Solaris/GENOM/bin/genom filtro.cfr -p
```

Fig. 5.23 Lamhosts with the names of nodes and the pathname to the executable

The second entry per line, here 0, 2, 1 and 1, is the number of additional processes that can be launched per each machine. Since the MPI run is started from *baltar* the master process runs on it, so it is advisable not to allow the execution of another process on it. The other nodes have associated one or two processes per machine. This approach presents several advantages because it is possible to apply efficient load management of computer power in unbalanced network. An unbalanced network occurs when the computer power distribution is not equally distributed between machines, either due to different machines or to machines with different loads. Balancing the number of processes according to the available computational resources reduces the overall optimization time.

5.2.5 Evaluation Engine

GENOM extends the optimization capabilities to some of the SPICE-like circuit simulators including the standard HSPICE and SPICE which share common characteristics. These simulators are capable of reading their inputs and producing results in text file formats, as well as, being launched from the command line. Other simulators can also be supported as long as these characteristics occur. A detailed description of the entire mode of operation ranging from the moment a chromosome is ready for evaluation until it attains the cost function value is presented in section 5.3.3.

5.2.6 Expansion of GENOM Tool

The GENOM synthesis tool consists of a set of interconnected software modules which comprise the user interface, the evaluation engine, the distributing computing API and the learning machine beyond the optimization engine itself. These modules are called automatically when required by the synthesis flow. Preferentially, AIDA uses a XML text description files to pass information between internal modules taking advantage of the intrinsic XML properties. The XML file format provides the developer with a clean, robust and human readability documented

target, allowing a much easier debugging as well as reading and exporting to other file formats. If the necessary software modules are developed, then the presented system can also be applied to different design environments or can even be integrated in wider industrial applications. Fig. 5.24 depicts an excerpt of the configuration interface file used by AIDA framework to setup some functionalities of the GENOM tool.

```
/*****************************************************************************/
                        interface.c - configuration file
                Copyright (C) 2005 by Manuel Barros, fmbarros@ipt.pt
/*****************************************************************************/
#   This file contains the INPUT parameters to GENOM Optimizer- V.2
#   Using the command line:
#                Ex: ../genom RcIdeal.cfr -s -hspice
 *****************************************************************************/

<?xml version="1.0"?>
<AIDA>
<GAPAR>           # Optimization GA Algorithms under test
<num of runs>     20       #number of runs
<evaluations_max> 10000    #number maximum of evaluations
<initial_seed>    1000     #initial seed
<population_size> 64       #population size
<mut_rate_max>    0.25     #maximum mutation rate
<stop_criteria>   1        #"1=Maximum num_generation 2=1st solution 3=25 STAGNATED generations"
<convergence_lim> 10e-3    #Cost standard deviation limit for the convergence test
<update_report>   1        #1 = each generation 2= logaritmic 3= best changed
<num_of_runs>     20       #number of runs
<update_report>   1        #update reports
<ntotalsamples>   3000     #number of total samples
</GAPAR>
#
<KERNEL> # Optimization Algorithms under test
 ...
 ...
</KERNEL>
</AIDA>
```

Fig. 5.24 Interface between GENOM and AIDA design automation environment.

The Fig. 5.25 demonstrates a communication interface example resultant from the <update_reports> parameter specification defined in Fig. 5.24. At specific time intervals pre-defined by the user, it is carried out an update of the reports and the refresh of screen information. In the example above, <update_reports> is set to '1' meaning an update in each generation (see Fig. 5.24 for other options). The information delivered from the optimization tool intended for visualization purposes is treated by a parser that identifies pairs of keywords or tags (*fSpecs.out*, *fEvolution_Curve.out*, *fCorners.out*, *fParameters.out*). The information between those keywords is sent to the interface defined by the client (the entity that initiated the optimization order). Fig. 5.25 exemplifies one line of results sent from GENOM. The word "*fSpec.out*" is reserved and identifies the performance parameters and the following values have a precise syntax. The first argument specifies the iteration of evolutionary algorithm and the next ones are the optimal values for the performance in the same order of appearance as in the specs file ("*design.spc*").

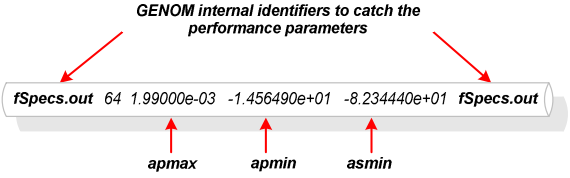


Fig. 5.25 Example of information delivered by GENOM

5.2.7 *Optimization Kernel Configuration*

This section presents the implemented approaches that support the optimization kernel. GENOM includes a kernel configuration file with commands to modify the behavior of the algorithm kernel. This task is limited to authorized computer algorithms specialists. Fig. 5.26 depicts a sample of the configuration interface file "AGPAR.h" used to setup some GENOM functionalities.

Each line between <KERNEL> tags is represented by a set of attributes that defines a particular characteristic of the kernel. The example, depicted in Fig. 5.26 defines the optimization of three different kernels, "GA-STD", "GA-MOD" and

```
/*****************************************************************************/
                        AGPAR.h - configuration file
                Copyright (C) 2005 by Manuel Barros, fmbarros@ipt.pt
/*****************************************************************************/
#   This file contains the the commands used to modify the behavior of the
#   algorithm kernel. This file has restricted access.
 *****************************************************************************/
<?xml version="1.0"?>
…
<COMMENTS>
1=Name              # "Name and specific GA PARAMETERS details"
2=strategy          # "1= Evolution Strategy  2=ES+SVM_REGRESSION 3=ES+SVM_CLASS 4= 1+2+3"
3=sampling          # "Initial Sampling [1= Random  2=DOE  3=LHS  4=outros]"
4=sort              # "Sort method 0= by cost function   1=by Feasibilidade"  2= by Constraints
5=selection_type    # "1= Random  2=Roulette wheel 3=Tournment Selection" 4= By Feasib. 5= By Masks"
6=crossover_type    # "1= 1-Point 2=2-points  3=3-points Uniform Crossover"
7=mutation_type     # "0= fixed   1=variable ==> STAGNATION TYPE"
8=mutation_factor # "1= One gene mutation 2= Two gene mutation 3= Three gene mutation"
9=adaptive_type     # "Adaptive step size [0= None 1= Adaptive]
10=training_size    # "Training set size [0= None 1= Search Space Percentage 2=NEVAL Percentage]"
11=test_size        # "Test set size [0= None 1= Search Space Percentage 2=NEVAL Percentage]"
12=svm_type         # "SVM type [0=none 1= C-SVC 2=nu-SVC 3=one-class SVM 4=epsilon-SVR 5=nu-SVR]"
13=kernel_type      # "SVM Kernel [0=linear 1=polynomial 2=radial basis-RBF 3=sigmoid]"
14=c_parameter      # "Set the SVM Cost parameter C of C-SVC, epsilon-SVR, or nu-SVR (default 1)"
15=gama_parameter # "Set the SVM gamma parameter G in kernel function (0=default 1/k)"
16=epsilon_par      # "Set the epsilon P in loss function of epsilon-SVR (0=default 0.1)"
17=cross_val        # "Cross validation flag [0= none 1= yes (standard)]"
18=agressive_cross# "Aggressive cross validation flag [0=none 1=yes]"
</COMMENTS>
#
<AIDA>
#  "1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18"
<KERNEL>
GA-STD - 1 - 1 - 1 - 1 - 1 - 0 - 1 - 0 - 0 - 0 - 0 - 0 - 1 - 0 - 0.1 - 0 - 1
GA-MOD - 1 - 2 - 2 - 4 - 2 - 1 - 2 - 1 - 0 - 0 - 0 - 2 - 10- 0 - 2.0 - 0 - 1
GA-SVM - 3 - 3 - 1 - 4 - 2 - 0 - 2 - 0 - 1 - 1 - 1 - 2 - 50- 0 - 4.0 - 1 - 0
</KERNEL>
</AIDA>
```

Fig. 5.26 Optimization kernel configuration file

"GA-SVM". At least one line should be presented for the correct functioning of GENOM. The command to execute a single optimization in 5 runs and respective simulation result is showed in Fig. 5.27. Each line depicts the run number, *#Run*, the number of evaluations in each run, *#nEvals*, final fitness value, *#Fitness*, simulation time, *#wTIME*, existence of feasible solution, *#FEAS*, and existence of a solution, "*#SOLUTION*", found at generation, "*found_@*". A feasible solution satisfies all designer rules but may miss one performance requirement, on contrary, if a solution is found, all designer rules, as well as, all the performance specs are satisfied.

```
/*****************************************************************************/
AGPAR.h - configuration file
Copyright (C) 2005 by Manuel Barros, fmbarros@ipt.pt
/*****************************************************************************/

Num_of_runs   5  #number of runs

<AIDA>
#  "1    2   3   4   5   6   7   8   9  10  11  12  13  14  15   16   17   18"
<KERNEL>
GA-MOD - 1 - 3 - 3 - 4 - 2 - 1 - 2 - 1 - 0 - 0 - 0 - 0 - 1 - 0 - 0.1 - 0 - 1
</KERNEL>
</AIDA>
<EOF>
```

RESULTS:

- PLOT OUTPUT DATA in each run -

#Run	#nEvals	#Fitness	#wTIME	#FEAS	#found_@	#SOLUTION	#found_@
1	288	7.542e-02	32.87s	Y	10 (gen)	Y	14 (gen)
2	512	1.162e-01	53.61s	Y	22 (gen)	Y	28 (gen)
3	1088	7.881e-02	114.81s	Y	56 (gen)	Y	64 (gen)
4	608	3.428e-02	59.80s	Y	24 (gen)	Y	34 (gen)
5	640	9.562e-02	61.64s	Y	28 (gen)	Y	36 (gen)

Fig. 5.27 A single kernel configuration and results.

5.3 Data Flow Management

In a design automation tool there is a need to handle two types of data structures, one, to manage the circuit's database and the other to manage the simulation data. A good definition of the data structure can lead to efficient data management and improvements in reusability. For instance, the simulation measures, the performance parameters database, the sub-circuits blocks, the testbenches and the technological files are likely to be shared or reused, avoiding the redefinition of circuit's information. In the same way, the data management of simulation data from the synthesis process can also be improved due to the need to control and to establish relations between the huge amount of simulation data, normally, produced from the optimization process, the need to cope with the variety of file formats from

different simulators or even a simple access to the simulated data of a specific circuit simulation.

In GENOM, the circuit's database is managed externally by AIDA framework but the management of the simulation data is GENOM's responsibility. When used as a standalone application, GENOM requires the input files illustrated earlier in Fig. 5.6.

The next two sections explain how GENOM manages the data and structures.

5.3.1 Input Data Specification

The preferential method to input all the data specification is through a GUI, otherwise the required files have to be manually generated. The GENOM graphical user interface presented in Sect. 5.2.2 inherits some methods of AIDA framework, and, as a result, takes advantage of its technology, namely the data management and data structure used to create and maintain a circuit's library. A multilayered architecture structure organized in tables with relational data, as illustrated in Fig. 5.28 and Fig. 5.29, is used to store the information concerning the circuits introduced through the graphical interface and the data provided by the optimization tool for data visualization. The next screenshots show the input data specification of the filter depicted in Fig. 5.29.

id_circuit	name	categoryId	circuitType	behaviour	bahaviourName
219	TestBench AC Analysis	13	TestBench	22	null
221	Filtro Elíptico de 2ª Ordem	15	Circuit	18	Low Pass
225	stageInfo	-1	stageInfo	(Null)	
226	stageInfo	-1	stageInfo	(Null)	
227	innerFlowInfo	-1	innerFlowInfo	(Null)	

id_designParameters	circuitId	instanceName	name	...	minValue	value	maxValue	...	relation	valueString
8239	221		R1		1e3	1e3	1e3		null	null
8240	221		R2	...	1e3	1e3	1e3	...	null	null
8241	221		R3		1e3	1e3	1e3		null	null
8242	221		R4		1e3	1e3	1e3		null	null
8243	221		R5		1e3	1e3	1e3		null	null
8244	221		R6		1e3	1e3	1e3		null	null
8245	221		C1		1e-12	1e-12	1e-12		null	null
8246	221		C2		1e-12	1e-12	1e-12		null	null
8247	221		C3		1e-12	1e-12	1e-12		null	null
8248	221	XA1	RL	...	0.0	0.0	0.0	...	null	R1.R2/(R1+R2)
8249	221	XA1	RC		0.0	0.0	0.0		null	C2
8250	221	XA2	RL		0.0	0.0	0.0		null	R4

Fig. 5.28 The circuit and the parameter tables filled with data from an elliptic filter

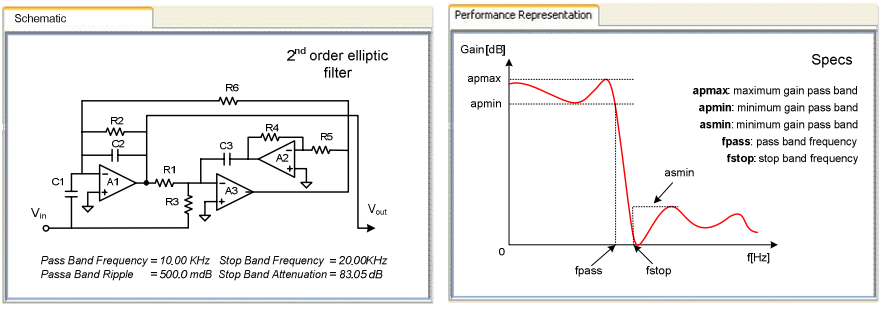


Fig. 5.29 2nd order Elliptic filter section and performance specs.

Essentially, the insertion of a new circuit requires the electrical schematic, a netlist, a technological file, the device parameters, the sub-circuits, the performance parameters and the corresponding measure functions. There are a lot of parameters with different nature associated to a circuit, so all information was arranged (split) in a meaningful storage of well-structured information. The first layer consists in the insertion of elementary data that defines a circuit. The table, at the top of Fig. 5.28, for example, stores the key of circuit identifier (*221*), the name (*Elliptic Filter of 2th Order*), the category (*Filter*, *OpAmp*, etc) of the circuit, the type of circuit (*Circuit*, *testbench*, etc) and the behavior class (*Low pass*). The design parameter table, at the bottom of the Fig. 5.28, represents the parameters table that characterizes each component from the netlist. There is a unique key that identifies each parameter (*8239*) plus the remainder characteristics and it is associated to the circuit where it belongs (*circuited=221*). This table is composed by a long list of parameters which includes a field that marks this component for optimization, another one indicates if the component is matched with any other or not (*matchComponen*t) and the correspondent matching value (*matchRelation*), above others.

The Fig. 5.30 shows the relational tables used to store the performance parameter information. The definition of performance parameters which can be measured in a circuit constitutes one critical step in the GENOM development as will be explored in the next section. Meanwhile, it will be explained how performance parameters and function measures are treated in GENOM.

The first step consists in the selection of the desired performance parameters (*apmin*, *apmax*, *asmin* and *stop band frequency*) for the chosen circuit, from the library of available performance parameters (see top table of Fig. 5.30). To avoid duplication of information, these parameters are stored in the table of design parameters composed by a unique identifier (*id_designPerformance*) and the performance parameter identifier (*performanceId*), for example, the key 28 corresponds to pass band maximum gain of the circuit in question (*circuitId*). The next fields are accounted for the definition of the global objectives of the circuit. For example, to specify that the pass band maximum gain of the filter in question should be inferior to 0.5 dB, the introduced values should be defined as the value

id_globalPerformance	name	description	unit
18	DcGain	Dc Gain	dB
19	Phase_Margin	Phase Margin	degree
20	GainBandWidth	Gain badnWidht product	dB
21	Power	Power	
28	Apmax	Pass Band max gain value	dB
29	Apmin	Pass Band min gain value	dB
30	Asmin	Stop Band min gain value	dB
31	fstop	Stop band frequency	Hz
32	fpass	Stop band frequency	Hz

id_designPerformance	performanceId	circuitId	relation	value	currentValue
99	28	221			
101	29	221			
102	30	221			
103	31	221			
104	32	221			

id_measures	circuitId	performanceId	name	definition	analysisType
1	221	28	apmax	max vdb(vout) from= freqIni to= fpass	AC
2	221	29	apmin	min vdb(vout) from=freqIni to= fpass	AC
3	221	28	apmax	max vdb(vout) from=fstop to=freqEnd	AC

Fig. 5.30 Performance parameters and measures functions table

"*maximum*" in field "relation" and the value 0.5 in the field "*value*". The column "*currentValue*" is used to store the value generated by the simulation tool or optimization.

The last step consists in the definition of the measure functions or simply measures which allow the determination of the performance parameter values. In the example considered above three measures for AC analysis are defined. The measures (*id_measures*) are associated to one circuit (*circuitId*) 221 and one performance parameter. They are characterized by a specific name and defined (field "*definition*") in HSPICE format.

5.3.2 Evaluation/Simulation Data Hardware

The quest behind GENOM tool is to provide the designer with an easy access to most relevant simulated data assent in a model of efficiency and precision of results. A block level representation of the simulation data flow in GENOM is exhibited in Fig. 5.31.

The data flow management is explained in three moments of simulation. The first three blocks of Fig. 5.31 cover the setup phase using the circuit management explained in the preceding section.

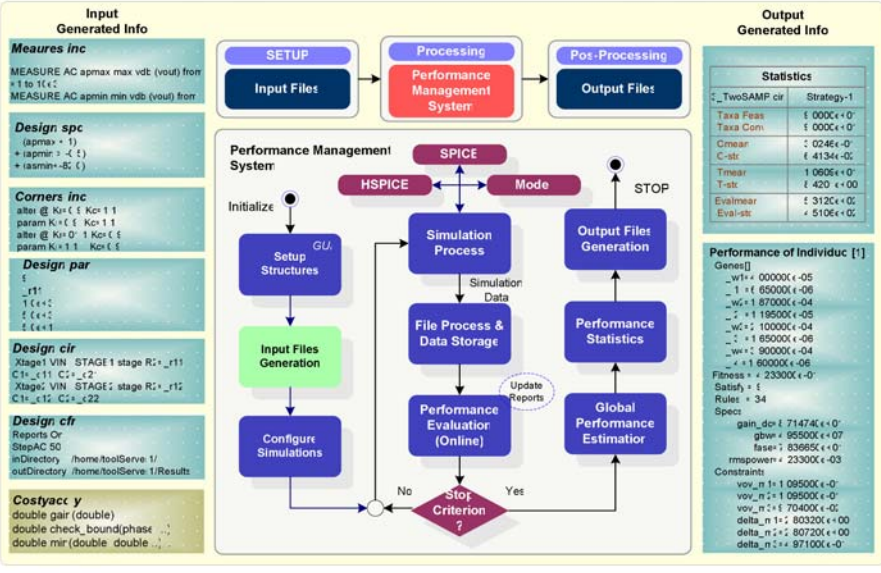


Fig. 5.31 The simulation data management system overview

The second moment is achieved during circuit synthesis process. Here, a parser was created to interpret the language of a circuit specification file and automatically compute the cost function value giving as input the performance parameters of the circuit and the formulations of the cost membership functions. The parser implementation was based in the *Lex* and *Yacc* [4] generation tools so that it is represented by a set of combined grammatical and lexical rules.

The last moment involve the use of built in functions to filter, process and display statistical data from the optimization process either in text or in graphical mode. The primary advantage of text files is that they are very flexible and easy to use. They can be any length, and can accommodate the information to any type of layout and allow the use of database techniques to query a text file.

The principal method of data access involving optimization algorithm and circuit simulator take advantage of the plotting facilities generally found in most electrical simulators. All output variables of interest can be printed in output files using the command ".PRINT" or equivalent. The data format of the response is generally organized in tabular form as depicted in Fig. 5.32. It shows the AC characteristics of the magnitude of voltage and phase in the output node of a filter for a given range of frequencies.

In order to access the data in a file, a file parser is implemented (file process block in Fig. 5.31). The use of file parsing techniques allows the extraction of any necessary information and its employment for later processing. GENOM provides built in functions to view the data in graphic mode version (bode plot characteristics and the cost function evolution). In command mode version, only the

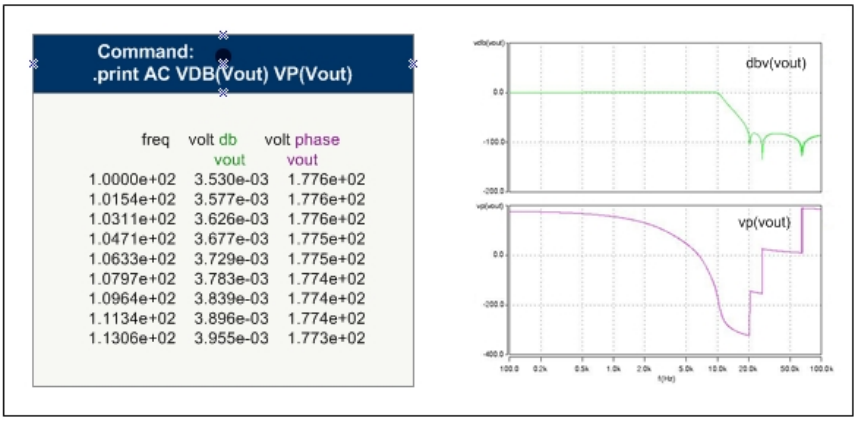


Fig. 5.32 AC analysis in the output node of a filter

extracted plotting files are created, allowing its final treatment with external graphical tools like Avanwaves® [5] or CosmosScope® [5]. The optimization with HSPICE simulator has an extra option that can be automatically invoked to visualize the waveforms in CosmosScope®. The processing of data employing circuit simulators with the purpose of performance estimation employs the same general principle but will be explained next.

5.3.3 Output Data

The entire mode of operation ranging from the moment a chromosome is ready to evaluation until it attains the cost function value will be explained in the following steps and supported by Fig. 5.33.

Step1 - As soon as a new candidate chromosome is submitted to evaluation process, a parser algorithm replaces the optimization parameters values in the target netlist with new ones corresponding to the genes of the chromosome. The "*target.cir*" netlist file is changed.

Step 2 - The new circuit netlist is submitted to electrical simulator (SPICE/HSPICE) producing in the output file (*target.out* or *target.lis*) a long list of simulation data including the matrix of variables and values of interest, and normally the performance parameters resulting from the simulation. This point diverges from simulator to simulator. In SPICE the type of variables are within the scope of command ".PRINT". The HSPICE simulator is more flexible because it incorporates a new command called measures, which gives the user more freedom to print and customize user-defined electrical specifications of a circuit. Actually, this is the preferred method to pass information between HSPICE and GENOM, since in the output file there is only the answer to the requested measures left, thus resulting in a compact file and allowing a more efficient access.

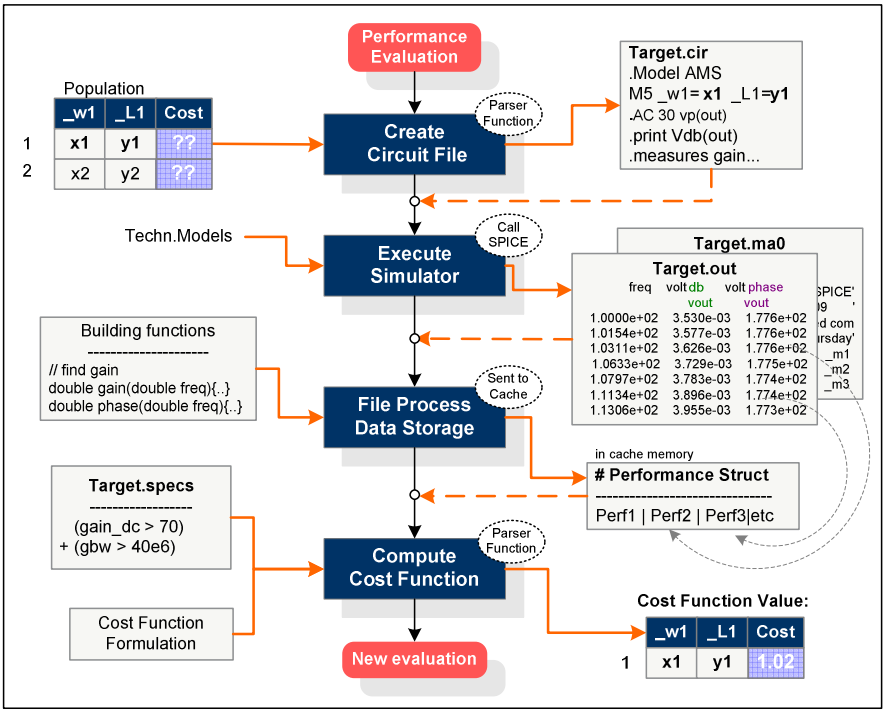


Fig. 5.33 The performance evaluation data flow

Step 3 - Next, a set of built in functions extracts the data information matrix stored in one or more output files and sends it to cache memory structures for fast manipulation. When the required information is not explicit stored, a new built in function is created to compute its value. At the end of this step, all necessary parameters needed to compute the cost function, are organized in memory by the order they appeared in targets specs file.

Step 4 - Finally, the cost function value is automatically computed with the help of a new cost parser function based on the compiler Lex and Yacc (details in sub-Sect. 5.3.3.1). Simultaneously, it collects a set of statistical data that is important to control the optimization algorithm, such as, the number of satisfied solution, the number of violated constraints, the corner's information, etc.

5.3.3.1 The Simulation and Equation Based Cost Function Parser

This section explains the parser implementation behind the cost function computation. The main purpose of the parser is to create a mechanism able to interpret the language of a circuit specification file and automatically compute the cost function value giving as input the performance parameters of the circuit and the formulations of the cost membership functions. The parser implementation was based in the Lex and Yacc generation tools so that it is represented by a set of combined

grammatical and lexical rules as illustrated in Fig. 5.34. The Lex description file identifies a series of symbols (logic and arithmetic operators), regular (mathematical functions and built in functions) and transforms them into tokens (reserved word for the language). Once this transformation is done, the YACC syntaxical analyzer interprets this stream of tokens and converts it into a meaningful grammar. With this specification, the GENOM's parser not only is able to interpret more traditional circuit specification files (based on logic and arithmetic operators, see "*target.specs*" in Fig. 5.35) but also specification files based on user defined equations (equation-based). The user defined equations can be expressed through basic mathematical functions ('*Fabs*', '*SIN*', '*SQRT*', '*POW*', etc) or by more sophisticated built in functions such as 'gain()', 'phase()', '*get_Value_Cache()*', *min()*, etc. For example, the function "*double gain(double freq)*", finds the gain corresponding to frequency from the output file of a SPICE simulation. If the performance measures are already in cache memory then the "*get_Value_Cache()*" function can be used instead.

Fig. 5.35 gives a simplified macro view of the actions taken automatically by the parser machine to carry out a single performance parameter. When the "*cost_Calc()*" function triggers the process, the first line of the design specification file is ready for parser analysis. The expression "*(gain_dc>70)*" is evaluated and the identifier "*gain_dc*" must be resolved first. Since "*gain_dc*" expression did not match any of the parser reserved word, it is interpreted as a performance parameter whose value should be read from memory with "*get_Value_Cache()*". Then, the obtained expression '90 > 70' is resolved by executing a set of operations specified by the operator '>'. One of these operations performs a call to the membership functions that translate the impact of this measure in the overall performance. Then the process is repeated line by line until the end of the target specification file.

Lexical Rules "Costlex.l"

```
%option noyywrap
ID [a-zA-Z][_a-zA-Z0-9]*
VAR _[a-zA-Z][_a-zA-Z0-9]*
FPNUM (([0-9]+)|([0-9]*(\.[0-9]+)?([eE][-+]?[0-9]+)?))
%%
"fabs"     { return FABS; }
"sin"      {  return SIN; }
"sqrt"     { return SQRT; }
"pow"      { return POW; }
"gain"     { return GAIN; }
"phase"    { return PHASE; }
"f_gain"   { return F_GAIN; }
"f_phase" {return F_PHASE; }
"check_bound"     { return CHECK_BOUND; }
"verify_bound"    {  return VERIFY_BOUND; }
"min" {return MIN; }
{ID}   { yylval.name = yytext; return ID; }
{VAR} { yylval.name = yytext; return VAR; }
{FPNUM} { yylval.value = atof(yytext); return NUM; }
"+"    { /*printf("'+\n");*/ return '+'; }
"-"    { /*printf("'-\n");*/ return '-'; }
"*"    { /*printf("'*\n");*/ return '*'; }
"/"    { /*printf("'/\n");*/ return '/'; }
">"    { /*printf("'>\n");*/ return '>'; }
"<"    { /*printf("'<\n");*/ return '<'; } ...
```

Grammatical Rules "Costyacc.c"

```
cost: exp { cost_val = $1; }
exp: exp '+' term {
            $$ = $1 + $3; }
   | exp '-' term {
            $$ = $1 - $3; }
// ...
exp: exp '>' term {
            var=
getValue_Cache($1);
  if (var-$3 >= 0.0)
  {  $$=0.0;
     SAT_SPECS++;
            } else {
  $$=fabs(mf_GTA(var,$3));
            $$=getFit($3,var);
} ...
double gain(double);
double phase(double);
double f_gain(double);
double f_phase(double);
double check_bound(double, ...);
double getValue_Cache(int);
double find_indexMeasure(...);
double verify_bound(double, ...
double min(double, double,..); ...
```

Grammatical Rules "Costyacc.c" (cont.)

```
...
func: FABS '(' exp ')' {
        $$ = fabs($3);
    }
   | SIN '(' exp ')' {
            $$ = sin($3);
    }
   | SQRT '(' exp ')' {
            $$ = sqrt($3);
    }
   | POW '(' exp ',' exp ')' {
            $$ = pow($3, $5);
    }
   | GAIN '(' exp ')' {
            $$ = gain($3);
    }
   | PHASE '(' exp ')' {
            $$ = phase($3);
    }
   | F_GAIN '(' exp ')' {
            $$ = f_gain($3);
    }
   | F_PHASE '(' exp ')' {
            $$ = f_phase($3);
    } ...
```

Fig. 5.34 Cost function parser overview

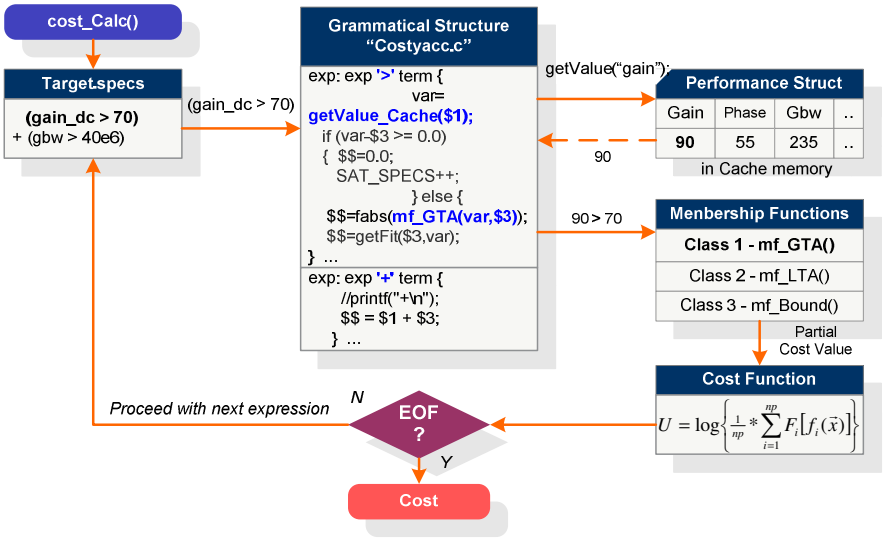


Fig. 5.35 Processing of performance parameters

The parser mechanism allows the implementation of a uniform methodology to access and manipulate data from several sources using simple structures, like the precedence of the operators, their layout, and other grammatical rules which may include built in functions. The use of built in functions allows the integration of new simulators maintaining always a common interface to evaluation of performance parameters.

5.4 Conclusions

This chapter discussed the design architecture, methodology and design implementation of GENOM optimizer tool. The main building blocks included in GENOM are the optimization kernel, the evaluation module and the Graphical User Interface.

The optimization kernel is available with several approaches including the GA standard approach, the modified GA-MOD and the hybrid approach GA-SVM incorporating a learning model based on SVMs.

GENOM was designed to integrate SPICE like simulators, deal with equation based problems and interact with a learning SVM machine. A flexible parser machine was developed to maintain a common interface of the evaluation module allowing the access and manipulation of data from different simulators.

The graphical user interface that controls the inputs and outputs of the system allows the visualization of iterative progress reports. With this feedback an experienced user can assume an active part in the optimization process because he owns some vital information that allows him to twinkle some design parameters during the search mechanism.

References

[1] Barros, M., Neves, G., Horta, N.C.: AIDA: Analog IC design automation based on a fully configurable design hierarchy and flow. In: Proc. 13th IEEE International Conf. on Electronics, Circuits and Systems, pp. 490–493 (2006)

[2] Lourenço, N., Horta, N.C.: LAYGEN – An evolutionary approach to automatic analog IC layout generation. In: Proc. IEEE Conf. on Electronics, Circuits and System, Tunisia (2005)

[3] Lourenço, N., Vianello, M., Guilherme, J., Horta, N.C.: LAYGEN – Automatic layout generation of analog ICs from hierarchical template descriptions. In: Proc. IEEE Ph. D. Research in Microelectronics and Electronics, pp. 213–216 (2006)

[4] Niemann, T.: A compact guide to Lex & Yacc (2004), `http://epaperpress.com/lexandyacc/` (Accessed March 2009)

[5] Synopsys Inc, CosmosScope-Waveform analysis (2009), `http://www.synopsys.com` (Accessed March 2009)