ASCO
A SPICE Circuit Optimizer

Written by João Ramos

Companion to version 0.4.11

Contents

Chapter 1

Preface

1.1 Tool Fitness

1.2 Scope and Audience

The information described in this manual assumes that is to be used by an expert circuit designer with the knowledge of the operation of a SPICE simulator. The ASCO tool is intended as a helper to the designer in their quest to "better" design a circuit. It shall not be used as an automatic way to size a circuit by someone which cannot understand the consequences of each one of the assumptions made during the optimization.

ASCO tool does not create nor suggest new circuit arrangements, but simplifies the design process, since fine-tuning, verification and optimization of circuit functionality over process-voltage-temperature corners is automated.

ASCO tool can be seen as the automaton that does the tedious work, thus giving time to the designer to concentrate on intellectual challenges of devising new architectures and solutions for existing problems.

1.3 Document Conventions

This document uses the following conventions for fonts and commands, which are shown in Table 1.1.

Table 1.1:

Convention	Description
`courier`	Indicates a code fragment
Brackets (`[ ]`)	Indicates the component is optional
Arrows (`< >`)	Indicates the component is mandatory
Pipe (`|`)	Indicates that one of the items can be selected

1.4 Trademarks

All products mentioned in this document are the property of their respective owners and carry the appropriate trademarks, registered trademarks, and/or copyrights. Any trademark infringements are unintentional.

Chapter 2

Quick Start Guide

This Chapter describes the minimum set of commands and information needed to start using ASCO to optimize a CMOS inverter with a 10 pF load operating at 850 MHz. The goal is to find the PMOS transistor width for minimum power consumption.

2.1 0 seconds: Compiling

Download the latest version and compile ASCO by typing at the command prompt

```
tar -zxvf ASCO-<version>.tar.gz
cd ASCO-<version>
make
```

2.2 20 seconds: Running ASCO

Copy the executable `asco` to `examples/<YOUR_SIMULATOR>/inv`. If you are using Eldo™ execute the following commands

```
cp asco examples/Eldo/inv
cd examples/Eldo/inv
./asco -eldo inv
```

otherwise, replace the word `Eldo` and `eldo` by the appropriate text representing your simulator.

2.3 140 seconds: Interpreting ASCO Results

Depending on your computer the optimization can take up to a few minutes. At the end, two log files are available: `asco.log` which has a summary of the optimization evolution and `<hostname>.log` where the input and output variables of every SPICE simulation call are stored.

2.3.1 File `asco.log`

With the data in the following file, it is possible to know the number of SPICE calls (`nfeval`), the minimum cost (`cmin`, lower the better) which is a measure on how good the circuit is, and the `cost-variance` which is an indication of the optimization convergence. The best test vector (`best[0]`), scaled from -10 to +10 is produced. In addition, a summary of the Differential Evolution optimization algorithm parameters are printed for future reference. In the last lines, the reason for ending the optimization is given.

```
nfeval=20   cmin=3.1914          cost-variance=3.9852
nfeval=30   cmin=3.1914          cost-variance=2.6059
nfeval=40   cmin=3.188           cost-variance=1.1017
nfeval=50   cmin=3.188           cost-variance=0.11595
nfeval=60   cmin=3.188           cost-variance=0.048998
nfeval=70   cmin=3.188           cost-variance=0.036212
nfeval=80   cmin=3.1879          cost-variance=0.0048361
nfeval=90   cmin=3.1879          cost-variance=0.0025139
nfeval=100   cmin=3.1879          cost-variance=0.00048362
nfeval=110   cmin=3.1795          cost-variance=0.00041958
nfeval=120   cmin=3.1696          cost-variance=0.00012673
nfeval=130   cmin=3.1696          cost-variance=5.568e-05
nfeval=140   cmin=3.1696          cost-variance=1.873e-05
nfeval=150   cmin=3.1696          cost-variance=1.1294e-05
nfeval=160   cmin=3.1696          cost-variance=5.8484e-06
nfeval=170   cmin=3.1696          cost-variance=3.3654e-06
nfeval=180   cmin=3.1696          cost-variance=3.5067e-07



 Best-so-far obj. funct. value = 3.1696

 best[0]=-7.990420912

 Generation=18  NFEs=180   Strategy: DE/rand-to-best/1/exp
 NP=10    F=0.5     CR=1        cost-variance=3.5067e-07


INFO:  de36.c - Minimum cost variance reached (cvarmin=1.000000E-06)
INFO:  Ending optimization
```

2.3.2 File `<hostname>.log`

All simulation calls are stored in this file having the name of the running machine. The data is saved in plain text format and allows further post-processing with a spreadsheet or

a graphic plotting program. The last line of this file is now shown

```
+cost:3.169600E+00:     P_SUPPLY:3.169600E-01: +VHIGH:2.154500E+00: //
+VLOW:2.605600E-02:      WP:1.904311E-03:
```

The plus signal as the first character in the line, as opposed to `-cost`, is an indication that all optimization constraints have been met. In the last column, the transistor size, 1.904 mm, that originates the minimum power consumption is presented.

2.4 More Information

Please keep on reading. Should doubts arise, contact the developer.

Chapter 3

Introduction

ASCO aims to bring circuit optimization capabilities to existing SPICE simulators. It takes an unsized netlist and design criteria and outputs a sized netlist. As a result of the previous sentence, the ASCO tool requires an experienced designer who selects the circuit topology, finds reasonable operating conditions with realistic design goals, defines the test benches and measurements for achieving the desired design objectives, evaluates the proposed sized circuits and select the most suitable circuit. All this is based on knowledge of the application. In return, the tool automates the test of multiple candidates for a given fixed circuit using a hybrid optimization strategy with a high-performance differential evolution (DE) global optimization algorithm [SP95] coupled with a local optimizer (Hooke&Jeeves or Nelder-Mead). The best result is passed from one to the next. Different circuit architectures can be tried, falling to the designer the task of selecting the most appropriate one.

The ASCO tool as been written from the ground-up with the purpose of being simulator independent. As long as the simulator reads its inputs from text files, outputs its results in ASCII format, and can be launched from the command line, it can most likely be added to the list of supported SPICE simulators. Even though it has been designed to work with existing SPICE simulators, at this moment it is flexible enough to interact with other tools, for example, with FastHenry.

Today, it is possible to find various offers of commercial products with similar characteristics. In some cases, the optimization algorithms are suitable only for local optimizations. In other cases, they are better than the ASCO tool, but this comes at a monetary cost. Once again, it is up to the designer to decide the best option to meet the design goals.

3.1 Features and Applications

ASCO is the result of academic research which in itself did not intend to create a new tool, but only to design high performance analog low-power low-voltage circuits for mobile communications. Interaction with other experienced designers has resulted in the ideas existing in the ASCO tool. With the exception of the optimizers, all code has been personally

written. The key features of the ASCO tool are:

- Simulator independent: currently out-of-the-box support for Eldo™, HSPICE®, LTspice™, Spectre®, Qucs and ngspice exist. More are to be included in future releases.
- Number of variables: there is, in theory, no limit to the number of circuit variables that can be optimized, except those constraints imposed by the available computer memory and/or the time required to generate a functional circuit. It is currently hardcoded in the C code.
- PVT corners: by using the simulator functionality, the possibility to test various design corners and Monte Carlo analysis is only limited to the simulator capability and by the time it takes to finish the optimization.
- Efficiency: the optimization algorithm features a global optimization using differential evolution. It has been used on a variety of applications and is known to produce good results in an acceptable time. Furthermore, this algorithm can be chained with a local optimizer to gain convergence speed as suggested by open literature.
- Parallel/distributed computation: time savings are achieved by simultaneously evaluating the proposed values over multiple computers on the network. No limit exists on the total number that can be used concurrently.
- Within the supported SPICE simulators, an arbitrary netlist can be optimized on different conditions without having to recompile the code.
- File format: all outputted data and log information is stored in plain text format. This guarantees that they will always be readable in the future. In addition, it makes possible the use of other existing tools to post-process the optimization results.
- It is free software: the code is available under the GNU GPL license.

ASCO has been designed to address problems that are oriented to electric circuits. Although not limited, some possible applications can include:

- Fully redesign a new circuit described in a SPICE netlist.
- Reuse, optimize an existing circuit.
- Migrate an existing and working design to a more advanced semiconductor technology process effortlessly.
- Increase the robustness and yield of an already designed circuit by guaranteeing that it comply with all design goals and constraints in some/all process corners at will.
- Easily explore a new operating point (design space) of an already existing topology, to reduce power consumption, area or both.

- Look for a feasible new design topology before investing a considerable time trying to derive equations that describe its operation.

Refer to Chapter 6 for ready to use practical examples to introduce you to ASCO, a SPICE circuit optimization tool.

Chapter 4

Installation and Operation

4.1 Installing ASCO

ASCO is written in ANSI C. Portability on *NIX type operating systems should follow relatively easily. Download the latest version and at the command line type the following:

```
tar -zxvf ASCO-<version>.tar.gz
cd ASCO-<version>
make
```

Two executables are created: `asco` and `asco-test`. Copy them to a common directory so that they can be used later.

4.1.1 Building with Autotools

For portability building and installing across different environments, the use of GNU Autotools is desirable. The necessary `configure.ac` and `Makefile.am` files are packed in `Autotools.tar.gz` distributed alongside with ASCO. Execute

```
tar -zxvf Autotools.tar.gz
```

to extract all necessary files. Then, type the following lines to configure and compile

```
aclocal
automake -f -c -a
sh autogen.sh
./configure
make
```

and obtain the same executables.

4.1.2 Building in win32

ASCO has been patched to compile natively in win32 using cygwin and MinGW32. However, should difficulties continue, try using Makefile.win32 instead. You are kindly invited to report progress and difficulties.

A binary version is also made available to download at the project homepage, for situations where a build system is not installed.

Text file format

Before using the example files, the end-of-file format must be converted. This is necessary because ASCO is developed in a *NIX system, where the EOF is `0x0D` instead of `0x0D0A`.

4.1.3 Compile time options

For increased flexibility but also because sometimes SPICE simulators differ considerably, ASCO flow is sometimes defined at compile time.

Optimization algorithms

Currently, one global (DE) optimization algorithm and two local (Hooke&Jeeves or Nelder-Mead) optimizers are available to use in ASCO. The parameters for tuning the differential evolution (DE) algorithm are accessible via the configuration file. On the contrary, local optimizers can only be tunned in the source code. Furthermore, the selection of the two local optimizers, is only possible at compilation time by editing the file `asco.c` around `Step4`.

The local optimizers use an intrinsically a sequential method. To gain design speed in situations where a good local starting point is known, the parallel global DE algorithm can be used in an emulated local mode. Thus benefiting from distributed computing while strongly limiting the search space.

Work distribution method

At the present time, three parallelization methods are supported in ASCO. By default, scattering with load balance is used if the source file `de36.c` is not edited and line: `#define MPI_METHOD 3` is not changed to one of the possible methods:

1. Send: each process receives one vector at a time and returns the single cost back to the Master process before receiving new parameters (one vector).

2. Scatter: within the same generation, all vectors are scattered among all existing processes. All evaluations are performed before returning all the cost values back to the Master process. In the next generation, it repeats itself.

3. Scatter with load balancing: for situations where the computer power is evenly available to each one of the processes, either due to different machines or to machines with different loads, balancing the number of simulations according to the computational resources, decreases optimization time. On a perfect balanced situation, all processes start and finish their work at the same time. This, despite having received a different number of vectors.

Spectre® BSIM selection

It is not possible to know on before hand which BSIM transistor model level is going to be needed or has been used in Spectre®. The output from Spectre® version changes from version-to-version but the change is more significant between different transistor models. As such, the selection between BSIM3 and BSIM4 (the only two supported models) has to be made by editing the source files. Search for `BSIM4` string. File `auxfunc_measurefromlis.c` requires 2 one byte changes, while the file `auxfunc_updatelis.c` requires commenting 5 lines and un-commenting another 5 lines of code. The changes are self explanatory for an experienced C programmer.

4.2 Using ASCO

Usage of ASCO requires the existence of a determined number of files that must reside in the current directory. The simulator that evaluates the cost function (see sub-section 4.2.2) must be on the search path.

The definition of the cost function is calculated automatically by the ASCO tool. This is the function that has to be minimized. Individual minimization or maximization of each one of the measurements is also accomplished without user intervention. In most of the cases, only the specification of the parameters range and constraints is sufficient before starting a new optimization.

4.2.1 Encapsulation to the SPICE Simulator

TODO: A description on the program interface between ASCO and the SPICE simulation program is to be included here.

4.2.2 The Cost Function

The implemented sizing methodology is a simulation-based optimization approach using a differential evolution optimization algorithm [Sto96] (the *global optimizer* block in Fig. 4.1). The key property of this optimization algorithm is that it generates new parameter vectors by adding the weighted difference vector between two population members to a third member. If the resulting vector yields a lower objective function value than a predetermined population member, the newly generated vector replaces the vector with which it is

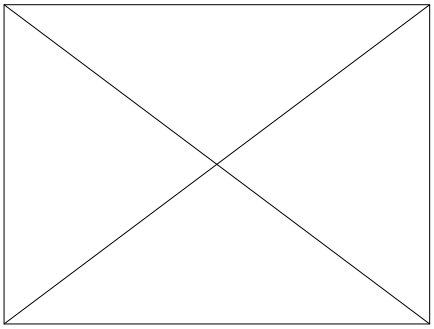


Figure 4.1: Interface between ASCO tool and the SPICE simulation program.

compared. For each vector $\mathbf{x}_{i,G}$ of generation G, a perturbed vector $\mathbf{v}_{i,G+1}$ is generated as follows:

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) \tag{4.1}$$

The indexes r_1, r_2 and r_3 indicate three randomly chosen individuals of the population. They are mutually different integer indexes ($\in [0, (N-1)]$) and also differ from the running index i. The (real) constant factor F ($\in [0, 2]$) controls the amplification of the differential variation. The vector $\mathbf{x}_{r_1,G}$ that is being perturbed has no relation to the vector $\mathbf{x}_{i,G}$ that will potentially be replaced. To increase the potential diversity of the perturbed parameter vectors, crossover is introduced. More information about the algorithm and details of several variants or strategies for constructing new parameter vectors can be found in [SP95, Sto96]. In addition, this algorithm has been altered to include parameter bounding, stop criteria and mixed continuous/discrete parameter support.

Any circuit variable (device sizes, component values, bias inputs, ...) can be selected as optimization parameters. Furthermore, one or more optimization *objectives* (minimize, maximize) can be specified as well as a number of (performance) *constraints* (e.g. $A_{LF} >$ 60dB or $V_{node1} < 0.1$V). All these requirements (n objectives and m constraints) are combined into a single cost function [1] which can be evaluated by the optimizer:

$$\begin{aligned} Cost &= W_{obj} \cdot \sum_{i=1}^{i=n} P_{sim_i} \\ &+ W_{con} \cdot \max_{j \in [1,m]} \left(\frac{P_{spec_j} - P_{sim_j}}{P_{spec_j}} \right) \end{aligned} \tag{4.2}$$

with W_{obj} and W_{con} the weights for the cost due to the objectives and constraints, respectively, and where P indicates performances, either simulated or specified. With

[1] The cost function, or objective function, is the function being optimized. It represents the quantity that is to be minimized by the optimizer in a given search space. This function can be for example the power consumption, circuit area or the sum of both. Maximization of objective such as Phase Margin is obtained by minimizing 1/(Phase Margin).

properly scaled weights (which is very easily accomplished), the optimizer will first try to find feasible solutions (satisfying the constraints) and then further tunes the parameters to optimize the objectives. This scaling can easily be adjusted manually after a "*dry-run*" (which can also be automated) and only requires altering the order of magnitude of one of the weights depending on the cost values which are logged. In order to deal with complex problems with many constraints, a minimax problem formulation is used in (4.2). When the genetic algorithm proposes bad combinations of parameters (e.g. out of bound), a "high" cost is assigned (e.g. 10^8) to such solutions.

In order to facilitate the automated optimization of specific circuit classes (Op-Amps, comparators, ...), constraint and objective templates can be loaded. These could have been stored for reuse by the designer himself or provided by another expert designer.

4.2.3 Stop Criteria

Allows halting the optimization under user-defined conditions. Currently defined as the maximum number of generations that has been reached or the minimum cost variance is satisfied. See sub-section 4.3.2/Differential Evolution Options for further details.

4.3 Input and Output Files

A set of files are required to define the netlist and optimization configuration. First, a brief enumeration is given which is followed by a more in-depth description of each one of the files. They are:

`<inputfile>.*`	Properly formatted SPICE input netlist. Default extension is `.cir` for Eldo™, `.sp` for HSPICE®, `.net` for LTspice™, `.scs` for Spectre®, `.txt` for Qucs and `.sp` for ngspice.
`<inputfile>.cfg`	Configuration file having the same name as the SPICE input netlist with `.cfg` extension.
`extract/`	Extract commands for each one of the performances are stored in this directory.

Upon starting an optimization, the following files are created.

`<hostname>.tmp`	Temporary file containing a post-processed version of the SPICE input netlist `<inputfile>.*`.
`<hostname>.log`	Simulator results log file.
`<hostname>.*`	Simulator specific input/output files.
`asco.log`	Optimizer log file.

4.3.1 SPICE input netlist

In the SPICE input netlist, all devices, sub-circuits, and simulation commands necessary to have a functional simulation must exist. The measurement lines are better excluded from this netlist and should be introduced via the configuration file (`<inputfile>.cfg`) for flexibility. An example for a simple CMOS inverter in Eldo™ is now shown:

```
*Digital inverter

.PARAM V_SUPPLY = '#V_SUPPLY#'
.PARAM INP_FREQ = '#INP_FREQ#'
.PARAM INP_PERIOD = '1/INP_FREQ'
.PARAM NO_PERIODS = '4'
.PARAM TMEAS_START = '(NO_PERIODS-1)*INP_PERIOD'
.PARAM TMEAS_STOP = '(NO_PERIODS)*INP_PERIOD'
.PARAM TMEAS_1 = 'TMEAS_STOP -3*INP_PERIOD/4'
.PARAM TMEAS_2 = 'TMEAS_STOP -1*INP_PERIOD/4'

*** *** SUPPLY VOLTAGES *** ***
VDD VDD 0 V_SUPPLY
VSS VSS 0 0

*** *** INPUT SIGNAL *** ***
VSIG IN VSS PULSE V_SUPPLY 0 'INP_PERIOD/2' 'INP_PERIOD/1000'
+               'INP_PERIOD/1000' 'INP_PERIOD/2' 'INP_PERIOD'

*** *** CIRCUIT *** ***
MP OUT IN VDD VDD PMOS W='#WP#'   L=#LMIN#
MN OUT IN VSS VSS NMOS W='#WP#/3' L=#LMIN#

CL OUT VSS 10p

*** *** ANALYSIS *** ***
.TRAN 'INP_PERIOD/1000' 'NO_PERIODS*INP_PERIOD'
.MC 2 ALL
.PROBE TRAN V(IN)
.PROBE TRAN V(OUT)
.OPTION EPS=1E-6
.INCLUDE p.typ
.INCLUDE n.typ
.END
```

In the above netlist representing a digital inverter, all lines are the same as in a normal simulation with a couple of exceptions:

- All values that are to be replace by the optimizer are enclosed in number sign # #. This does not necessarily imply a variable to optimize. It can also be a fixed number whose value is set for flexibility in the configuration file. This is a simple method to optimize a write protected SPICE input netlist circuit with different operating conditions.
- This is optional: no measurements exist in the netlist, although those that will not be used to verify circuit performance and/or correct operation can still be present.

4.3.2 Configuration File

Instructions on how to carry on the optimization are defined in the configuration input file. Information regarding the optimization algorithm schedule, SPICE re-run using the ALTER command, Monte Carlo matching performance, parameters, measurements and post-processing, must be defined in the configuration file which is divided in categories that are now presented.

The following syntax is enforced throughout the configuration file so ASCO can properly write and read the input and output files:

- All comment lines must start with an asterisk (*).
- Comments following a command (in-line) start with the dollar ($) sign.
- Category name is enclosed within the number (#) symbol. If no space exists in the beginning and at the end of the category name (#text#), the command lines in that block of lines cannot be interchanged nor deleted, only the value. On the contrary, if a space is present (# text #), then the category is not locked and an arbitrary number of lines can exist and their relative position is irrelevant.
- Number (#) character at the end of a category must exist at all times.
- Syntax is a colon separated list.

Optimization Flow Options

In this category, which is very likely to be revamped in future releases, optimization chain steps are described. So far, only two exist but there is the possibility to make them user defined in the order they are executed and in a fixed number of possible steps.

```
#Optimization Flow#
Alter:no            $do we want to do corner analysis?
MonteCarlo:no       $do we want to do MonteCarlo analysis?
AlterMC cost:0.00   $point below which ALTER and/or MONTECARLO can start
ExecuteRF:no        $Execute or not the RF module to add RF parasitics?
SomethingElse:
#
```

Word `yes` and `no` are used to perform or not a given analysis in the optimization loop. The scheduled simulations with `Alter` and/or `MonteCarlo`, are only executed if all constraints have been met in the previous cost evaluation, otherwise, they are executed immediately after the real value specified in the `AlterMC cost`.

During and optimization, misconvergence can occur if before starting simulations reruns using `Alter`, a lower cost has already been obtained than the possible best when including design corners. This situation can be avoided by not setting `AlterMC` too low. Similar situation applies to the `MonteCarlo` case.

Differential Evolution Options

The optimization algorithm settings are grouped in this category which are now explained. Some text has been verbatim copied from the source code.

```
#DE#
choice of method:3
maximum no. of iterations:50
Output refresh cycle:2
No. of parents NP:60
Constant F:0.85
Crossover factor CR:1
Seed for pseudo random number generator:3
Minimum Cost Variance:1e-6
Cost objectives:10
Cost constraints:100
#
```

There is no choice of parameters that fits all. Each optimization problem has an ideal choice of the above factors. However F=0.5 and CR=0.8 can be taken as good starting point alongside with method 3 or 4. Read the following lines for further clarification.

- `choice of method`

  An explanation of the naming-convention follows for the DE/x/y/z. DE: stands for Differential Evolution; x: a string which denotes the vector to be perturbed; y: number of difference vectors taken for perturbation of x; z: crossover method (exp = exponential, bin = binomial). When the DE/best... schemes fail DE/rand... usually works and vice versa. One of the following methods can be chosen using a number between 1 and 10.

  1. DE/best/1/exp: The oldest strategy but still not bad. Several optimization problems where misconvergence occurs have been found.
  2. DE/rand/1/exp: It works especially well when the "bestit[]"-schemes experience misconvergence. Try e.g. F=0.7 and CR=0.5 as a first guess.

3. DE/rand-to-best/1/exp: This strategy seems to be one of the best strategies. Try F=0.85 and CR=1. If you get misconvergence try to increase NP. If this doesn't help you should play around with all three control variables. Similar to DE/rand/1/exp but generally better.
4. DE/best/2/exp: Another powerful strategy worth trying.
5. DE/rand/2/exp: Seems to be a robust optimizer for many functions.
6. DE/best/1/bin: Essentially same strategy but binomial crossover.
7. DE/rand/1/bin: Essentially same strategy but binomial crossover.
8. DE/rand-to-best/1/bin: Essentially same strategy but binomial crossover.
9. DE/best/2/bin: Essentially same strategy but binomial crossover.
10. DE/rand/2/bin: Essentially same strategy but binomial crossover.

- `maximum no. of iterations NI`

  Stop criteria. Be aware that the maximum possible number of SPICE simulation calls can be as large as NIxNP. Generations is another name used for iterations.

- `Output refresh cycle`

- `No. of parents NP`

  Number of population members. To start off NP equal to ten times the number of parameters is a reasonable choice. Increase NP if misconvergence happens. If you increase NP, F usually has to be decreased. The number of population members NP is also not very critical. A good initial guess is 10xD. Depending on the difficulty of the problem NP can be lower than 10xD or must be higher than 10xD to achieve convergence.

- `Constant F`

  DE-stepsize F from interval [0, 2] which affects the differential variation between two individuals. The scale factor F must be above a certain minimum value to avoid premature converge to a local minimum (sub-optimal solution). However, making F too large causes the number of function evaluations to increase before converging to an optimum solution. On the other hand, it allows global exploration of the search space. F is usually between 0.5 and 1 (in rare cases >1). DE is also somewhat sensitive to the choice of the stepsize F.

- `Crossover factor CR`

  Crossover probability constant from interval [0, 1] which affects the diversity of population for the next generation. Helps to maintain the diversity of the population and is rather uncritical, with 0.0, 0.3, 0.7 and 1.0 being optimal first choices. If the parameters are correlated, high values of CR work better. The reverse is true for no correlation. In low-dimensional problems (<10), higher values of crossover probability work better to preserve the diversity in the population.

- `seed for pseudo random number generator`

  Self-explanatory.

- `Minimum Cost Variance`

  Another stop criteria. Simulation stops if current cost variance is smaller than the defined value.

- `Cost objectives`

  W_{obj} in (4.2)

- `Cost constraints`

  W_{con} in (4.2)

More information can be found either in the C source file `de36.c` or in [SP95, Sto96].

Alter Options

Refer to Section 7.1 for information on this category.

Monte Carlo Options

Refer to Section 7.3 for information on this category.

Parameter Options

```
# Parameters #
Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:---
Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:OPT
Input frequency:#INP_FREQ#:850E6:0:0:LIN_DOUBLE:---
PMOS width:#WP#:70u:75u:250u:LIN_DOUBLE:OPT
Multiplier:#M#:4:2:5:LIN_INT:OPT
#
```

A sequence of colon separated specifying different fields: Text description, Symbol, Initial value, Minimum, Maximum, Number format and Type. A clarification of each one the parameters is now presented:

- Text description: of the variable. Any text is acceptable.
- Symbol: must be enclosed with `#  #`. ASCO searches the SPICE input netlist and replaces every single occurrence by its numerical value.
- Initial value: of the parameter.
- Minimum: lower bound of the parameter.

- Maximum: upper bound of the parameter.
- Number format: Naming-convention follows x_y, where x stands for the scale in the feasible range; y is the number format. x can either be `LIN` (linear interpolation) or `LOG` (logarithmic interpolation) while y can take `DOUBLE` (continuous) or `INT` (discrete) as possible values for the type of variables.

  For IEC 60063 preferred number series (E3, E6, E12, E24, E48, E96 and E192 only) use for example `:E48:` alone. The code was contributed by Stefan Mahr.
- Type: if the parameter is to be optimized, `OPT` must be added, otherwise use `---` to represent a parameter that is to be kept constant throughout the optimization. The previous is useful to simulate the exactly same SPICE input netlist under different conditions that are changed only in the configuration file, that is, for netlist integrity purposes.

Notes:

- With the initial value, minimum and maximum, units must not be included, only scale factors. This is: `2A` is read as 2 atto instead of 2 Ampere, while `3F` is read as 3 femto instead of 3 Farad. The Volt unit (`V`) must not be added.
- Exponential format (1e2, 1e-5, 1e+4) or engineering format (K, T, N) can be interchangeably used. Use one of the following:

		T=1E12	G=1E9	MEG=1E6	K=1E3
A=1E-18	F=1E-15	P=1E-12	N=1E-9	U=1E-6	M=1E-3

- Character case is ignored.

As such, each one of the above lines implies:

- `Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:---`

  Text clarifies that this parameter is related to the `Supply voltage`, the symbol to look for and replace in the SPICE input netlist is `#V_SUPPLY#`. Initial value used for the optimization is `2.0` Volt with a minimum and maximum of `0`. Because it is only a parameter to replace (note the string `---`) the maximum values would have been ignored. In this case `LIN_DOUBLE` indicates a parameter with double precision with linear parameter variation.
- `Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:OPT`

  Owing to the fact that the maximum is equal to the minimum, but the variable is set to optimized, the parallel global DE algorithm is used in an emulated local mode. The minimum and maximum are internally set to -10/+10% of the initial value.

- `Input frequency:#INP_FREQ#:850E6:0:0:LIN_DOUBLE:---`

  The `Input frequency` where the symbol to look and replace is `#INP_FREQ#` has an initial value of `850E6` Hertz. Again, because it is not a variable to optimize, due to the presence of `---`, the minimum and the maximum values are ignored.

- `PMOS width:#WP#:70u:75u:250u:LIN_DOUBLE:OPT`

  The `PMOS width` has the symbol `#WP#`. The initial value is `70u`. In this case, the parameter is to be optimized (`OPT`) with a minimum value of `75u` and a maximum value of `250u`. The double precision is used alongside with a linear swept of the optimization parameter range due to the `LIN_DOUBLE` keyword.

- `Multiplier:#M#:4:2:5:LIN_INT:OPT`

  Possible values are 2, 3, 4 and 5 for multiplier parameter because `xxx_INT` is used. Initial value in the optimization is set to 4.

Measurement Options

To avoid the introduction of a user-defined cost function, objectives and constraints must be manually introduced in the configuration input file. As such, measurements that assert circuit performance and/or correct operation shall not be included in the input SPICE netlist (`<inputfile>.*`). This also simplifies the user work, as the definition of the cost function is frequently tricky. The preferred method is to add an entry in the `# Measurements #` category of the input configuration file. Furthermore, the added advantage is that this knowledge in the form of objectives and constraints is stored in a template which can be reused later.

```
# Measurements #
P_SUPPLY:---:MIN:0
P_OUT:OUT:GE:0.0316
#
```

A sequence of colon separated specifying different fields: Measurement, Node, Objective or Constraint, Gain or Constraint value, having the following meaning:

- Measurement: The name of the measurement to perform. See sub-section 4.3.3 for the format.
- Node: at which the measurement is to be done.
- Objective or Constraint: objectives can either be `MIN` (minimize) or `MAX` (maximize) while constraint can be `LE` (lower-or-equal), `GE` (greater-or-equal) and `EQ` (equal: hardcoded to 1 %). `MON` keyword is used to monitor a measurement while ignoring its value from the cost function calculation.

- Gain or Constraint: value that is dependent on the previous selected parameter. If it is an objective, the entry represents a gain to the cost function (currently not implemented and hardcoded to 10) or else it is the user-defined constraint value.

Taking as example the above lines:

- `P_SUPPLY:---:MIN:0`

  Measure the power supply, no node is specified, minimize is the optimization objective. Because the objective gain is not yet implemented, the hardcoded value of 10 is used instead of `0`.

- `P_OUT:OUT:GE:0.0316`

  Measure the output power at node `OUT` which must be greater-or-equal than `0.0316` Watt.

Only one objective and one constraint is specified, but in theory there is no limit to the maximum number of objectives and/or constraints that can be considered in an optimization. Neither is there a limit to the type of measurements to perform. However, different cost functions or different penalties can lead to distinctive "optimal" solutions. It is thus advisable to have only one objective combined with the necessary constraints. A figure-of-merit (FOM) which accounts for all partial minimize/maximize goals is then minimized by the optimizer.

Post Processing Options

Refer to Section 7.4 for information on this category.

4.3.3 Extract Commands

Each one of the templates is stored in a file in the `extract/` directory that must reside in the same parent directory where the SPICE input netlist is. It is mandatory to name the file as the name used in `# Measurements #` category of the configuration file (Subsection 4.3.2/Measurement Options).

In Eldo™, the following example is used to measure the output power at a user-defined frequency.

```
# Info #
Name:P_OUT
Symbol:ZP_OUT
Unit:W
Analysis type:TRAN
Definition:Output power at the fundamental harmonic.
Note:
#
```

```
# Commands #
.OPTFOUR TSTART=TMEAS_START TSTOP=TMEAS_STOP NBPT=1024
.FOUR LABEL=fftout v(#NODE#)
.EXTRACT FOUR LABEL=#SYMBOL# {((YVAL(FOUR(fftout), INP_FREQ))^2)/(2*FILT_RES)}
#

# Post Processing #
#
```

Each file defining one parameter extraction must have the three categories shown above:

- `# Info #`

  All fields are self-explanatory. Currently ignored.

- `# Commands #`

  Commands to be included in the SPICE netlist to extract waveform information from a simulation run are defined here. The node where the measurement is to be done, is replaced by the text given in sub-section 4.3.2/Measurement Options. Currently, the symbol is automatically filled by the tool.

- `# Post Processing #`

  Refer to Section 7.4 for the appropriate syntax. Note, that only the `MEASURE_VAR` command can be used in here.

4.3.4 Output Files

The option of naming the output files with the machine name the optimization is running on, is to ease the transition to a multi-CPU environment where multiple machines optimizing the same input SPICE netlist write their output to different files.

All output files are simulator specific and are created by the simulator that is being used. The exception, two log files that report all the steps that have been done during the optimization loop, more precisely:

- `<hostname>.log` In this file, the results of each one of the simulations is stored in a character separated value for easy importing by a spreadsheet. In each one of the lines a detailed report quantities is given:
  - The current cost of the evaluation, which carries the character "+" if all constraints are met, otherwise "-" is added.
  - A list of objectives (minimize, maximize), constraints (lower-or-equal, greater-or-equal, equal) and measurements to monitor. Again, those constraints that have been met, have the character "+" added.

  - All components values that have been optimized have their value documents on the right most part of each line.

  Before exiting, one last simulation is executed with the best set of set of values obtained during the optimization. In this way, it is thus possible to analyze the SPICE output log file and visually see each one of the waveforms for correct operation assurance. The line, or lines in case corner analysis, should start with "+cost".

- `asco.log` General information about the optimization process is stored in this file.

4.4 Invoking ASCO

Copy the ASCO executable to the directory where your files are. Not obligatory but having everything stored in one place, means that you can move from computer to another, without having to worry if the optimizer is installed or even with different ASCO versions having different formats. Furthermore, by doing this you can easily compare differences between versions. You decide which option best suits your needs.

To invoke ASCO, simply type at the command line:

```
./asco -<eldo|hspice|ltspice|spectre|qucs|ngspice|general> <inputfile>
```

The simulator to use must be specified in the first input argument. The `<inputfile>` can include the file extension.

4.4.1 The usefulness of `asco-test`

During a long optimization, which can include corner analysis and/or Monte Carlo simulation, the netlist (`<hostname>.tmp`) is changed at the beginning of each new step. It might be that everything is running as expected, but the execution of the Alter or Monte Carlo options introduces an error in the temporary file `<hostname>.tmp`. For this purpose, the executable `asco-test` is used, since it ignores the fulfillment of any constraint and simple goes from optimization to Alter and then to Monte Carlo.

As programmed, an Alter and/or Monte Carlo simulation is only started after all constraints are met in the previous step. This, unless `AlterMC cost` is defined in the configuration file with a high value making that the simulation re-runs start immediately, because the returned cost from the previous simulation is lower than the `AlterMC cost` defined to start simulating the PVT corners. The normal flow, if all steps are to be executed is: optimization⇒Alter⇒Monte Carlo. If an error exists in any of the subsequent steps, the simulator might not run at all. The necessary corrections must be made so that the sequence of the three steps can end without errors. After this, the long optimization sequence can start, so that at this time one can rest assured that no errors exist in either Alter and/or Monte Carlo netlist. At the command line, type:

```
./asco-test -<eldo|hspice|ltspice|spectre|qucs|ngspice|general> <inputfile>
```

As programmed in this current version, it is not possible to first execute Monte Carlo simulations and then Alter. Monte Carlo always follows the optimization or Alter.

4.4.2 Runtime Messages

To provide some help where the execution is going, there are three type of messages that are written to the standard output:

- Informative messages are outputted with `INFO: ...`.
- Debug messages are written with `DEBUG: ...` and only appear when executing `asco-test` with the objective to provide enough information to the user to correct the SPICE input netlist and the ASCO configuration file.
- All other remaining messages are error messages. The C code filename and the function at the point the program cannot continue, with a small explanation of the error type is given. The program always exits after this message.

4.5 Invoking ASCO with multiprocessor support

The differential evolution optimization algorithm can be easily parallelized in order to distribute computational effort among different CPUs. This is a consequence of the inherently parallel nature of the DE algorithm within each generation, where population members are individually evaluated. The parallel/distributed version of ASCO can use as many as NP (number of parents in the DE algorithm) computers in parallel because there are no gains in using more.

Interprocess communication is handled by the Message Passing Interface (MPI), a standard for a parallel program on a distributed memory system communicate to other nodes. In the model implemented, there is one master process responsible for giving to each slave the data which is then simulated on another computer. The cost function at the end of each simulation is then returned to the master. Due to the time required for a packet to travel among nodes, and in comparison with the sequential code, the parallel implementation is more suitable to expensive function evaluation.

First it is necessary to download, compile and install the implementation of the MPI standard available at:

```
http://www.mpich.org/downloads/
```

If the executable `mpicc` is not in the `PATH` environment variable, the ASCO `Makefile` must be edited and the full path added to `CC_MPI`. Compile `asco-mpi`, the executable with parallel processing support by typing at the command prompt:

```
make asco-mpi
```

Before continuing, make sure the simulator is available through `ssh` or `rsh`. If the `PATH` environment variable is set in your current shell but not in your remote shell environment, the simulator might not be visible. To test this, type the following at the command line

```
ssh|rsh <MACHINE_NAME> <simulator>
```

In case of an error two possible solutions exist. You can either edit your shell configuration file to add the path to the directories that are searched by default, or you can edit the file `errfunc.c` around line 600-650 (in `Step3`), and add the full path to the simulator. To execute the parallel version with just one computer, type

```
mpirun -np 2 asco-mpi -<simulator> <inputfile>
```

This is exactly the same as executing the sequential optimization algorithm, but with two processes, one being the master and the other being the slave. Should a network of computers be available, the following method can be used:

```
mpirun -f machines.txt [-np X] asco-mpi -<simulator> <inputfile>
```

where, to execute on three computers, `machines.txt` is a text file containing a list of machines similar to

```
node1
node1:2   #2 processes on this node
node2:1
node3
```

in which `nodeN` is the name of the machine as given by the *NIX command `hostname`. More than one process can be launched per computer. The simulation netlist(s) and configuration file(s) is/are then copied to the local disk (of the node computer) to reduce traffic in the network and as such decrease total optimization time.

The result given by the parallel version of ASCO is exactly the same as given by the sequential implementation. The exception is in the case where PVT corner analysis is performed. Nevertheless, a similar result is reached.

4.5.1 MPICH configure options

If the ASCO binary uses all your CPU cycles, you are advised to recompile MPICH using the sock channel instead of the default nemesis channel. You can do this by configuring MPICH with the `--with-device=ch3:sock`.

4.5.2 `asco-mpi` with new Spectre® versions

If when using `asco-mpi` with a Spectre® simulator version from circa 2012 and onward, the Spectre® simulator starts but do not take any CPU, you need to downgrade your MPICH to an older version. New Spectre® simulator versions also use MPI (possibly also MPICH) and it (possibly) interferes with recent MPICH commands. You are advised to use

`http://ftp.mcs.anl.gov/pub/mpi/mpich-1.2.7p1.tar.gz`

Chapter 5

Efficient Usage

5.1 Accuracy

In itself, the ASCO tool does not define the accuracy of the results. The algorithm used in ASCO has been applied to various sets of problems and has obtained good results in term of speed, robustness and convergence. More information is available in the DE's homepage in: http://www.icsi.berkeley.edu/~storn/code.html.

The methodology applied to the ASCO tool has been proved on silicon. For further details see Chapter 2 [Ram05] for a low-voltage low-power design of a three stage operational amplifier and Chapter 6 [Ram05] for the design in the presence of passive and board parasitics, a high-efficiency 30 dBm class-E CMOS two-stage power amplifier for the GSM standard.

5.2 Convergence Speed

In the DE algorithm, alongside with the `choice of method`, three parameters deserve special attention to define the convergence speed: `NP`, `F` and `CR`. Electric circuits transfer function being multi-modal require large populations (`NP`) to find a working topology. Likewise, the constant `F` must be above a certain value so that premature convergence to a local minimum does not occur. In addition to that, parameters are usually dependent which makes large values of `CR` work better. However, overestimating `NP` and `F` has the consequence that the number of SPICE simulation calls grow quickly, and thus, slows the optimization process.

A careful selection of the number of optimization PVT corners and a thoughtful number of Monte Carlo simulations, is a good trade-off between optimization time and robustness. In addition, planning the exact moment to start sweeping the process corners is critical for minimizing CPU time. A two-step approach, where first only the optimization is done, followed by shrinking the variables search space to the range where all constraints have been met, i.e., upper and lower range from all the lines starting with `+cost`. Only then, the more computer intensive PVT optimization is performed.

Minimizing the number of ASCII and binary files created by the simulator, the number of measurements (.EXTRACT or .MEAS) and output variables (.PROBE), and the number of analysis (.AC, .PZ and .TRAN) to the absolute minimum required to efficiently characterize the circuit in conjunction with a keen speed/accuracy simulation compromise, saves precious CPU time in the optimization loop.

Open literature suggests that further gains in terms of convergence speed could be obtained by first doing a global search to find a good starting point that is latter used by a local optimization algorithm such as Hooke&Jeeves or Nelder-Mead. This decision is taken at compilation time by editing the file `asco.c` around `Step4`.

5.3 Number of Optimization Variables and Search Space

Fully optimizing from scratch a complex system with a considerable number of variables might end up being a road block, although being possible to do by the tool. In same cases, it may be more efficient to steadily increase the number of variables to optimize and learn from experience. Furthermore, overly increasing the search space with the hope of finding *the very best* solution, might once more lead to an endless optimization loop.

5.4 Objectives and Constraints

Too many objectives and/or constraints may require a circuit that is simply not feasible in theory. A reduced number is advisable whenever wanting to explore a new operating point that is simply too difficult to derive equations for. Yet, in same situations a given constraint is paramount to guide the optimizer to a functional solution that otherwise is difficult to reach. Experience is once more an added value.

5.5 Evaluating Optimization Results

Whenever possible, try to understand the reasons behind the optimization results. Always proceed with caution unless you can justify the proposed circuit sizes. To increase confidence, re-run the simulation with the Alter option enabled and/or with a different range of the search space for the optimization variables. If a similar result is obtained, the optimizer is probably converging to the global minimum and the circuit is likely working as desired. In this situation in `asco.log`, the cost-variance is $<< 1$.

Thoroughly analyze the `asco.log` file as the circuit can be working on an undesired operating point or on the edge of stability. Then, explain what the optimizer has found.

Chapter 6

ASCO Tutorials

6.1 Getting Started

This chapter contains a description of examples included with the ASCO tool which are grouped by simulator name (Table 6.1). The basic step to prepare a new netlist to the format compatible with the ASCO optimization tool involves:

1. `<inputfile>.*`
   - Prepare a functional SPICE netlist for your simulator.
   - Measurement commands that are used to assert circuit performance and/or correct operation are (preferably) not included in the input SPICE netlist.
   - Select the variables to optimize. Replace their value with a unique string and enclose it in `#  #`.
2. `<inputfile>.cfg`
   - Edit the configuration file. Pay special attention to the category `# Parameters #` and `# Measurements #` in which the symbol and measurement name must match those in file `<inputfile>.*` and directory `extract/`, respectively.
   - Adapt the three DE control parameters: NP, F and CR, according to the difficulty of the optimization problem.
   - Carefully review if the remaining configuration file suits your needs.
3. Create the necessary measurements, each one in a separate file and place them in the directory `extract/` which is in the same parent directory where the SPICE input netlist is. If available from a central repository or another simulation, simply copy the necessary files.
4. Run the `asco-test` executable to remove any existing error. Only then proceed with the following steps.

5. Copy the ASCO executable to the place where your SPICE file is. Start the optimization loop.

6. During the optimization loop, data similar to the one below is printed to the terminal,

```
asco-0.4.1 - Copyright (c) 1999-2006 Joao Ramos
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

INFO:  Current directory on 'linux': /home/users/asco/examples/inv
INFO:  Eldo initialization on 'linux'
INFO:  Initialization has finished without errors on 'linux'

                          PRESS CTRL-C TO ABORT

INFO:  Starting global optimizer on 'linux'...

                          PRESS CTRL-C TO ABORT

 Best-so-far cost funct. value=1.6891

 best[0]=-4.180591714
 best[1]=6.665600201
 best[2]=0.03964113694
 best[3]=-7.372717493
 best[4]=3.691704024

 Generation=1  NFEs=80   Strategy: DE/rand-to-best/1/exp
 NP=40    F=0.7     CR=0.9    cost-variance=8.2357e+12
```

   which is a summary of the optimization startup procedure. No errors have occurred during the initialization of the optimization which has 5 optimization variables. The values of `best[]`, always between -10 and +10, are the values proposed by the DE algorithm. They are later scaled to the range defined in `<inputfile>.cfg`.

7. Upon ending the optimization, text similar to the next is printed in the terminal showing the final optimization results.

```
 Best-so-far cost funct. value=0.40609

 best[0]=-5.173398086
 best[1]=9.592135291
```

```
best[2]=9.780050221
best[3]=-8.086606726
best[4]=9.822119525

Generation=51  NFEs=2080   Strategy: DE/rand-to-best/1/exp
NP=40    F=0.7     CR=0.9    cost-variance=0.026777


INFO:  de36.c - Maximum number of generations reached (genmax=50)
Ending optimization


INFO:  ASCO has ended on 'linux'.
```

The cause for the optimization ending is also printed, in this case the maximum number of generations has been reached. The minimum obtained cost of `0.40609` is achieved after `50` generations and `2080` function evaluations. The optimization has converged as result of the low value of the cost-variance. Although this may be true, some of the `best[]` are close to the search range of [-10, +10]. This can be the result that toward the minimum and/or maximum value of a given vector lies the optimum point. Therefore it is advisable to increase the lower bound for `best[]` vectors close to -10 and the upper bond for `best[]` vectors close to +10.

8. Analyze the results stored in `asco.log`, `<hostname>.log` and the other simulator specific output files. If not already created (to increase optimization speed), re-run one last simulation with the necessary command(s) to save waveforms in an appropriated binary format suitable for a graphical viewer.

The following ready to use optimization examples can be found in the directory `examples/` of the ASCO distribution, grouped for each one of the supported simulators.

Table 6.1: ASCO Examples

Circuit Name	Description
inv	Digital inverter
amp3	Three stage operational amplifier
classE	Class-E power amplifier
bandpass	Chebyshev band pass filter

6.2 Eldo™ Examples

Latest tested version

Up to 2008.

6.2.1 Tutorial #1 – Digital inverter

This simple circuit gives a quick introduction to the procedures that must be executed before optimizing a circuit. The complete set of files described below can be found in `examples/Eldo/inv`.

The fact is that circuits having only one variable makes convergence quite fast on modern computers. As such, an example will show most of the ASCO present capabilities: optimize from scratch a circuit to achieve minimum power consumption while fulfilling the design constraints; guarantee that this is valid for different process corners and also take into account device parameter mismatch (Monte Carlo).

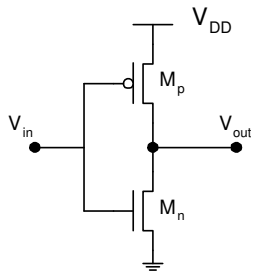


Figure 6.1: Digital inverter.

Summary

- One optimization variable
- One objective
- Two constraints/performance goals
- Three design corners (ALTER)
- Monte Carlo analysis

Full Netlist

```
*Digital inverter

.PARAM V_SUPPLY = '#V_SUPPLY#'
```

```
.PARAM INP_FREQ = '#INP_FREQ#'
.PARAM INP_PERIOD = '1/INP_FREQ'
.PARAM NO_PERIODS = '4'
.PARAM TMEAS_START = '(NO_PERIODS-1)*INP_PERIOD'
.PARAM TMEAS_STOP = '(NO_PERIODS)*INP_PERIOD'
.PARAM TMEAS_1 = 'TMEAS_STOP -3*INP_PERIOD/4'
.PARAM TMEAS_2 = 'TMEAS_STOP -1*INP_PERIOD/4'

*** *** SUPPLY VOLTAGES *** ***
VDD VDD 0 V_SUPPLY
VSS VSS 0 0

*** *** INPUT SIGNAL *** ***
VSIG IN VSS PULSE V_SUPPLY 0 'INP_PERIOD/2' 'INP_PERIOD/1000'
+               'INP_PERIOD/1000' 'INP_PERIOD/2' 'INP_PERIOD'

*** *** CIRCUIT *** ***
MP OUT IN VDD VDD PMOS W='#WP#'   L=#LMIN#
MN OUT IN VSS VSS NMOS W='#WP#/3' L=#LMIN#

CL OUT VSS 10p

*** *** ANALYSIS *** ***
.TRAN 'INP_PERIOD/1000' 'NO_PERIODS*INP_PERIOD'
.MC 2 ALL
.PROBE TRAN V(IN)
.PROBE TRAN V(OUT)
.OPTION EPS=1E-6
.INCLUDE p.typ
.INCLUDE n.typ
.END
```

The CMOS inverter, in which the transistor width is to be optimized has a 10 pF load capacitance. All measurements have been removed from the netlist and included in the `extract/` directory.

Configuration File

From the complete configuration file, available at `/examples/Eldo/inv`, only those categories that might require extra attention are now discussed.

```
#Optimization Flow#
Alter:yes           $do we want to do corner analysis?
```

```
MonteCarlo:yes      $do we want to do MonteCarlo analysis?
AlterMC cost:3.00   $point at which we want to start ALTER and/or MONTECARLO
ExecuteRF:no        $Execute or no the RF module to add RF parasitics?
SomethingElse:      $
#
```

Simulation re-runs are executed immediately upon having a returned cost below 3.00. Immediately after that, and because the `AlterMC cost` is smaller than the returned cost from the simulation, Monte Carlo Simulation is performed. For this to occur, the line `.MC 2 ALL` has to be present which tells the number of simulations runs, also `ALL` must exist.

```
# ALTER #
.param
+    V_SUPPLY=[2.0 2.1 2.2]
#
```

As a demonstration, only three process corners are executed. For an extensive list of possible examples refer to the configuration file.

```
# Parameters #
Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:---
Temperature:#TEMP#:25:0:0:LIN_DOUBLE:---
Minimal gate length:#LMIN#:0.35u:0:0:LIN_DOUBLE:---
Input frequency:#INP_FREQ#:850E6:0:0:LIN_DOUBLE:---
PMOS width:#WP#:10000u:1m:10m:LIN_DOUBLE:OPT
#
```

The PMOS transistor width is the only optimization variable used. The other parameters are used to configure the SPICE input netlist.

```
# Measurements #
P_SUPPLY:---:MIN:0
VHIGH:OUT:GE:1.95
VLOW:OUT:LE:0.05
#
```

In here, minimization of the power supply consumption is the objective. This, while meeting the constraint of having an output voltage above 1.95 V and below 0.05 V at a fourth of the signal period.

Command Line

```
./asco -eldo inv
```

Optimization Results Analysis

An inverter is a rather simple circuit. It can however be used as a good starting point on the steps required to check that not only the optimization has converged, but above all, to confirm that the circuit is indeed working according to the initial constraint and in a stable operating mode.

To verify that the optimization leads to a functional inverter, the binary output file containing the simulation with the best set of values obtained during the optimization is checked with the visual display of the saved output.

In the `<hostname>.log` file, a complete report of all simulations is stored. The last 12 lines refer to the bet test vector (Alter plus Monte Carlo). It is possible that not all lines start with "+cost". It should nevertheless be noted that the measured values are indeed very close to the constraint values. The number of function evaluations, minimum cost and cost-variance are stored in `asco.log` file. The reason for ending the optimization and a brief report of the DE parameters is stored in here for convenience.

The most important place to check correct operation is in the SPICE output log file: `<hostname>.chi`. A meticulous analysis is mandatory.

In the event that there is a need for statistical analysis on the simulation output log file, the tool `log` available in `tools/log/` can be used. It takes as input the optimization log file and creates a summary of all performed measurements. Should the `<inputfile>.cfg` exist, a more complete report is created. To use such tool, type in the command prompt

```
<PATH_TO_LOG>/log <hostname>.log <hostname>.log.log
```

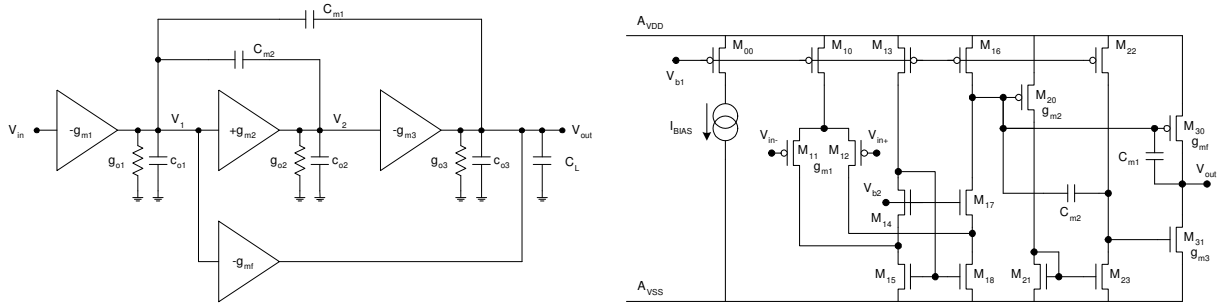
to obtain a file with a summary of the measurements and also with a list of parameters to use in new optimization. However, it is better to create the new parameter list (`# Parameters #`) in the situation where all constraints have been met, i.e., those where the line start with `+cost`. Filtering lines matching a pattern is obtained using the *NIX `grep` command

```
cat <hostname>.log | grep +cost > <hostname>_good.log
<PATH_TO_LOG>/log <hostname>_good.log <hostname>_good.log.log
```

after which a new summary can be obtained, this time with a smaller range for the parameters. With this in hand, a new re-run can be started, either to fine-tune the design or for a new and faster optimization having corners.

6.2.2 Tutorial #2 – Three stage operational amplifier

This tutorial describes the optimization of a three stage operation amplifier featuring the frequency compensation technique described in [Ram05]. The necessary files are available in `examples/Eldo/amp3`.



(a) Block diagram of the PFC amplifier.

(b) Schematic diagram of the PFC amplifier.

Figure 6.2: The PFC amplifier [Ram05].

Summary

- 21 optimization variables
- One objective
- Five constraints/performance goals

Full Netlist

```
*Three stage operational amplifier

*** *** OPAMP SUBCIRCUIT *** ***
.SUBCKT PFC.SUB VP VN VOUT IBIAS VB1 AVDD AVSS

M00 IBIAS IBIAS AVDD AVDD PMOS W=#WM00_10# L=#LM1#

* differential pair
M10 1     IBIAS AVDD AVDD PMOS W=#WM00_10# L=#LM1# M=6

M11 2     VN    1    1    PMOS W=#WM11_12# L=#LM2#
M12 3     VP    1    1    PMOS W=#WM11_12# L=#LM2#

* folded cascode
M13 4     IBIAS AVDD AVDD PMOS W=#WM13_16# L=#LM1# M=3
```

```
M16 5     IBIAS AVDD AVDD PMOS W=#WM13_16# L=#LM1# M=3

M14 4     VB1   2    AVSS NMOS W=#WM14_17# L=#LM3#
M17 5     VB1   3    AVSS NMOS W=#WM14_17# L=#LM3#

M15 2     4     AVSS AVSS NMOS W=#WM15_18# L=#LM4#
M18 3     4     AVSS AVSS NMOS W=#WM15_18# L=#LM4#

* second stage
M20 6     5     AVDD AVDD PMOS W=#WM20# L=#LM6#
M22 7     IBIAS AVDD AVDD PMOS W=#WM22# L=#LM1#

M21 6     6     AVSS AVSS NMOS W=#WM21_23# L=#LM5#
M23 7     6     AVSS AVSS NMOS W=#WM21_23# L=#LM5#

* third stage
M30 VOUT  5     AVDD AVDD PMOS W=#WM30# L=#LM6# M=22
M31 VOUT  7     AVSS AVSS NMOS W=#WM31# L=#LM7# M=5

* compensation
CM1 5 VOUT #CC1#
CM2 5 7    #CC2#
.ENDS PFC.SUB

*** *** SUPPLY VOLTAGES *** ***
VDD VDD 0 #VSUPPLY#
VSS VSS 0   0

*** *** BIAS VOLTAGE *** ***
VVB1 VB1 VSS DC #VBIAS#

*** *** BIAS CURRENT *** ***
IIBIAS IBIAS VSS #IBIAS#

*** *** SUB-CIRCUIT *** ***
XOPAMP VP VN VOUT IBIAS VB1 VDD VSS PFC.SUB

*** *** LOAD *** ***
RL VOUT VX #RLOAD#
CL VOUT VX #CLOAD#
VX VX VSS '#VSUPPLY#/2'

*** *** AC LOOP *** ***
```

```
VIN VP VSS '#VSUPPLY#/2' AC 1
RX  VN VOUT 1m AC=1E12
CX  VN VSS 10

*** *** ANALYSIS *** ***
.AC DEC 100 0.001 1E9
.PZ V(VOUT)
.PROBE AC VDB(VOUT)
.PROBE AC VP(VOUT)
.OP
.OPTION NOBOUND_PHASE
.INCLUDE p.typ
.INCLUDE n.typ
.END
```

Configuration File

```
#DE#
choice of method:3
maximum no. of iterations:100
Output refresh cycle:2
No. of parents NP:100
Constant F:0.7
Crossover factor CR:0.9
Seed for pseudo random number generator:3
Minimum Cost Variance:1e-6
Cost objectives:10
Cost constraints:10e6
#
```

A random starting point is used in combination with a population size of 100 for the evolutionary optimization algorithm. The number of iterations is arbitrarily set to 100, which translates into a long optimization time. However, this allows to verify that the optimal values are no longer changing significantly.

```
# Parameters #
Supply voltage:#VSUPPLY#:3.0:2.4:3.3:LIN_DOUBLE:---
Minimal gate length:#LMIN#:0.35U:0.35U:0.35U:LIN_DOUBLE:---
Bias voltage:#VBIAS#:1.25:1:3.0:LIN_DOUBLE:OPT
Bias current:#IBIAS#:5E-6:1E-6:10E-6:LIN_DOUBLE:OPT
Load capacitance:#CLOAD#:100E-12:100E-12:130E-12:LIN_DOUBLE:---
Load resistance:#RLOAD#:25E3:10E3:50E3:LIN_DOUBLE:---
C compensation 1:#CC1#:15p:2p:20p:LIN_DOUBLE:OPT
```

```
C compensation 2:#CC2#:3p:2p:20p:LIN_DOUBLE:OPT
Length group 1:#LM1#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 2:#LM2#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 3:#LM3#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 4:#LM4#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 5:#LM5#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 6:#LM6#:0.7E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Length group 7:#LM7#:0.5E-6:0.35E-6:7E-6:LIN_DOUBLE:OPT
Width M00_10:#WM00_10#:10E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M11_12:#WM11_12#:40E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M13+16:#WM13_16#:10E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M14_17:#WM14_17#:6E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M15_18:#WM15_18#:11.01E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M20:#WM20#:15E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M22:#WM22#:10E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M21_23:#WM21_23#:2E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M30:#WM30#:1.5E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
Width M31:#WM31#:1.5E-6:0.35E-6:50E-6:LIN_DOUBLE:OPT
#
```

In order to automatically size the amplifier circuit, the netlist of the PFC amplifier is parametrized using 21 design variables (one bias voltage and current, two compensation capacitors, seven transistor lengths and ten transistor widths). The number of transistor geometry variables is somewhat reduced by taking standard analog design constraints (e.g. the matching of differential input pairs and current mirrors) into account. However, constraints on the operating point of the circuit are not included, only the performance specifications are given as input to the tool. On the one hand, this makes the design space much more complex, but on the other hand this doesn't require specific circuit knowledge.

```
# Measurements #
ac_power:VDD:MIN:0
dc_gain:VOUT:GE:122
unity_gain_frequency:VOUT:GE:3.15E6
phase_margin:VOUT:GE:51.8
phase_margin:VOUT:LE:70
amp3_slew_rate:VOUT:GE:0.777E6
#
```

The original performances as in [Ram05] are taken as constraints, except for the power consumption, which is requested to be minimized.

Command Line

```
./asco -eldo amp3
```

Optimization Results Analysis

Depending on the computer speed, it may take between 10 and 30 minutes to find the first circuit that fulfills all design constraints. By comparison, the full optimization procedure takes much more time. To have a cost-variance $<< 1$ it is necessary to increase the maximum number of iterations to about 400. Although this optimization can be done in a single day, a somehow simpler yet accurate representation of the amplifier as depicted in Fig. 6.2(a) can be used. After obtaining the optimum values for the transconductances and compensation capacitors in a fraction of the time, proceeding to the optimization of the transistor level circuit in Fig. 6.2(b) is straightforward.

6.2.3 Tutorial #3 – Class-E power amplifier

In this example, a simple class-E amplifier intended for operation in the GSM-850 band is given. A more realistic model representing a differential two stage power amplifier, including all relevant circuit and board parasitics to better describe the circuit measurement performance alongside with measurements from a manufactured chip in a 0.35 um CMOS commercial technology, is given in [Ram05]. All the files are available at `examples/Eldo/classE`.

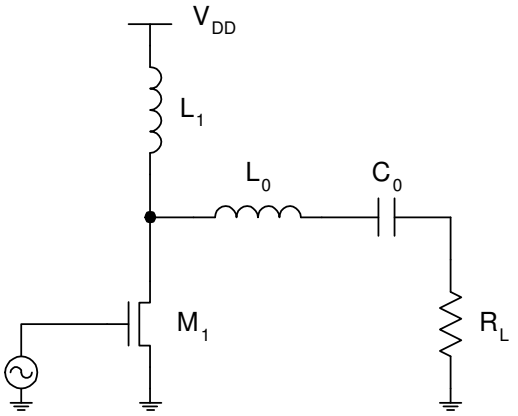


Figure 6.3: Class E power amplifier. It consists of CMOS switch M_1, the finite dc-feed inductance L_1, the series-tuned (L_0-C_0) and the load resistance R_L.

Summary

- Five optimization variables
- One objective
- Five constraints/performance goals

Full Netlist

```
*Class-E power amplifier

.PARAM V_SUPPLY = '#V_SUPPLY#'
.PARAM INP_FREQ = '#INP_FREQ#'
.PARAM INP_PERIOD = '1/INP_FREQ'
.PARAM NO_PERIODS = '#NO_PERIODS#'
.PARAM TMEAS_START = '(NO_PERIODS-1)*INP_PERIOD'
.PARAM TMEAS_STOP = '(NO_PERIODS)*INP_PERIOD'
.PARAM T_PERC = 99
.PARAM TMEAS_AUX = (NO_PERIODS-1)*INP_PERIOD
```

```
+                  + T_PERC/100*INP_PERIOD

*** *** SUPPLY VOLTAGES *** ***
* Voltages and currents
VDD VDD 0 V_SUPPLY
VSS VSS 0 0

*** *** INPUT SIGNAL *** ***
VSIG G1 VSS PULSE V_SUPPLY 0 'INP_PERIOD/2' 'INP_PERIOD/1000'
+               'INP_PERIOD/1000' 'INP_PERIOD/2' 'INP_PERIOD'

*** *** INDUCTOR *** ***
.SUBCKT LBOND.SUB IN OUT L=1
RBOND IN  1   '0.135*(L/1n)' ! 0.135 Ohm/mm; gold
LBOND 1   OUT 'L'            ! 1 nH/mm
.ENDS LBOND.SUB

*** *** OUTPUT STAGE *** ***
* Diffusion length, MOSwidth, MOSlength and multiplier
.PARAM LDIFF='1.2u' WS='#TR1_W#' LS='#LMIN#' MS='1'
M1 D1 G1 VSS VSS NMOS W=WS L=LS M=MS AD='WS*LDIFF' PD='2*(LDIFF+WS)'
+                                   AS='WS*LDIFF' PS='2*(LDIFF+WS)'

XL1 VDD D1  LBOND.SUB L=#L1#
XL0 D1  N2  LBOND.SUB L=#L0#
C0  N2  OUT #C0#

.PARAM FILT_RES = #RL#
R   OUT VSS FILT_RES

*** *** ANALYSIS *** ***
.TRAN 'INP_PERIOD/1000' 'NO_PERIODS*INP_PERIOD'
.PROBE TRAN V(G1)
.PROBE TRAN V(D1)
.PROBE TRAN V(OUT)
.OP
.OPTION EPS=1E-6
.INCLUDE n.typ
.END
```

The above file represents a class-E amplifier with a NMOS transistor acting as a switching device. Minimum inductor parasitics, using SPICE language, are included by the fact of the `LBOND.SUB` sub-circuit.

Configuration File

The relevant code from the configuration is now shown:

```
# DE #
choice of method:3
maximum no. of iterations:50
Output refresh cycle:2
No. of parents NP:60
Constant F:0.85
Crossover factor CR:1
Seed for pseudo random number generator:3
Minimum Cost Variance:1e-6
Cost objectives:10
Cost constraints:100
#
```

The three control parameters controlling the optimization algorithm and that must be chosen by the user: NP, F and CR are set to 60, 0.85 and 1, respectively.

```
# Parameters #
Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:---
Temperature:#TEMP#:25:0:0:LIN_DOUBLE:---
Minimal gate length:#LMIN#:0.35u:0:0:LIN_DOUBLE:---
Input frequency:#INP_FREQ#:850E6:0:0:LIN_DOUBLE:---
No of sim periods:#NO_PERIODS#:50:0:0:LIN_DOUBLE:---
TR1 width:#TR1_W#:1600.0u:5000u:20000u:LIN_DOUBLE:OPT
L1 inductance:#L1#:38.2n:0.1n:10n:LIN_DOUBLE:OPT
L0 inductance:#L0#:14.4n:0.1n:10n:LIN_DOUBLE:OPT
C0 capacitance:#C0#:4.82p:10p:70p:LIN_DOUBLE:OPT
Load resistance:#RL#:27.9:1:10:LIN_DOUBLE:OPT
#
```

In first three lines above, inputs for the SPICE simulation are shown. The next five lines configure the parameters to be optimized with the bounding range for each one of the circuit components.

```
# Measurements #
P_SUPPLY:---:MIN:0
P_OUT:OUT:GE:0.5
VDSOFF:D1:LE:0.2
SLOPEOFF:D1:LE:9E9
VDSON:D1:LE:0.2
VMIN:D1:GE:-0.2
#
```

The above category describes minimization of the power supply (the only objective) while meeting all other five constraints, specifically an output power of at least 0.5 W. The remaining four performance goals, ensure correct operation of the circuit as a class-E amplifier.

The minimization of the power supply while constraining the output power to be higher than 0.5 W is equivalent to maximizing the drain efficiency, thus the cost function has only one objective.

Command Line

```
./asco -eldo classE
```

Optimization Results Analysis

Upon completion, all simulator calls are logged to `<hostname>.log`. Each one of the lines contains the cost of the simulation, the power supply and a description of all performance goals which have the character "+" added if the constrain has been met, otherwise, have "-". The last part of the line have a list of all circuit sizes used in the simulation.

The character separated value makes importing to a spreadsheet easy where performance trade-offs among the various optimized circuit solutions can be studied: results in which the DC-feed inductance (L1) is below a certain threshold although the output power is less than the desired 0.5 W; all the cases where when transistor turns on, the voltage across the transistor drain is below than 0.2 V; etc. However, for situations where all constraints must strictly be met, the simple following shell command can be typed in the command prompt, to filter only those solution that have met all performance goals.

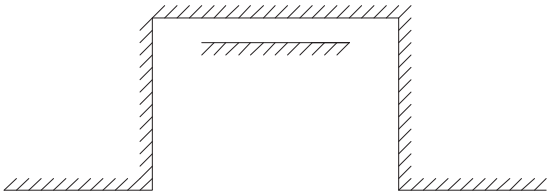
```
cat <hostname>.log | grep +cost > good.log
```

The same spreadsheet can again be used to analyze all results resting assured that only those where the design constraint have been met are shown.

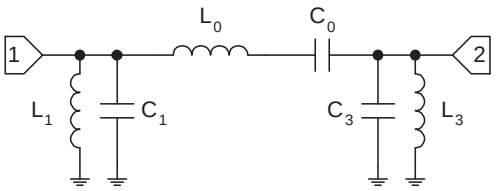
6.2.4 Tutorial #4 – Chebyshev band pass filter

Existing passives, either discrete or integrated have parasitics that must be considered when doing a design at high-frequency. Yet, accounting for their effect is often tedious and time consuming. As such, it becomes increasingly difficult to generate equations that are adequately accurate.

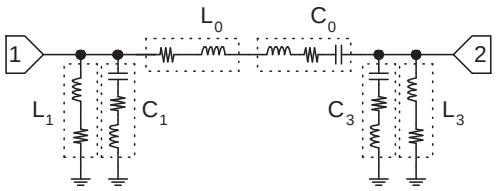
The RF module described in Section 7.5 can be used for situations when is of paramount importance to consider parasitic effects in a circuit. To this end, the circuit can have the necessary devices to describe a real world implementation, including chip, packaging and board parasitics. Considering them during the optimization, leads to a closer match between simulations and lab measurements. All the files are available at `examples/Eldo/bandpass`.



(a) Magnitude constraints tolerance scheme.



(b) Circuit topology.



(c) Equivalent of (b) with parasitics automatically added by the RF module during the optimization loop.

Figure 6.4: The Chebyshev band pass filter.

Summary

- Six optimization variables
- One objective
- Three constraints/performance goals

Full Netlist

```
*Chebyshev Band Pass Filter

*** *** FILTER CIRCUIT *** ***
C1 1 0 #C1#  ! #CSMD_50p80p#
L1 1 0 #L1#  ! #LBOND_350p450p#

L2 1 2 #L2#  ! #LBOND_60n100n#
C2 2 3 #C2#  ! #CSMD_300f340f#

C3 3 0 #C3#  ! #CSMD_50p80p#
L3 3 0 #L3#  ! #LBOND_350p450p#


*** *** PORT *** ***
V1 1 0 iport=1 rport=50
V2 3 0 iport=2 rport=50

*** *** ANALYSIS *** ***
.AC DEC 1000 800e6 1200E6
.PROBE AC SDB(1,1)
.PROBE AC SDB(2,1)
.END
```

The above text, describes a three pole Chebyshev band pass filter. The user-defined device models, are specified after the in-line comment specific to each one of the simulators used. Refer to Section 7.5 for a detailed description of its usage.

Configuration File

The relevant code from the configuration file is now shown:

```
#Optimization Flow#
Alter:no            $do we want to do corner analysis?
MonteCarlo:no       $do we want to do MonteCarlo analysis?
AlterMC cost:1.00   $point at which we want to start ALTER and/or MONTECARLO
ExecuteRF:yes       $Execute or no the RF module to add RF parasitics?
SomethingElse:
#
```

The difference to note is the definition of `ExecuteRF` setting to require the inclusion of RF parasitics in the netlist.

```
#DE#
choice of method:3
maximum no. of iterations:100
Output refresh cycle:2
No. of parents NP:30
Constant F:0.7
Crossover factor CR:0.9
Seed for pseudo random number generator:3
Minimum Cost Variance:1e-6
Cost objectives:10
Cost constraints:1000
#
```

Above, normal settings for an optimization. Increase the number of parents to achieve better results at the expense of a longer simulation time.

```
# Parameters #
C1:#C1#:  64.415p: 50p: 80p:LIN_DOUBLE:OPT
L1:#L1#: 393.233p:350p:450p:LIN_DOUBLE:OPT
L2:#L2#:  79.107n: 60n:100n:LIN_DOUBLE:OPT
C2:#C2#: 320.205f:300f:340f:LIN_DOUBLE:OPT
C3:#C3#:  64.415p: 50p: 80p:LIN_DOUBLE:OPT
L3:#L3#: 393.233p:350p:450p:LIN_DOUBLE:OPT
#
```

Each one of the filter components is defined as an optimization variable. It is based on this device value that the parasitics are found and added to the netlist by the RF module.

```
# Measurements #
Left_Side_Lobe:---:LE:-20
Pass_Band_Ripple:---:GE:-1
Right_Side_Lobe:---:LE:-20
S11_In_Band:---:MAX:---
#
```

The first three lines define the magnitude constraints tolerance scheme as shown in Fig. 6.4(a). The objective, maximization of the in-band S11, is given in the last line. Considering that in `extract/` it is defined as -S11, as result, the optimizer minimizes the in-band return loss (S11) value.

The other important file is in this case the place where the parasitic definition resides. In the file `rfmodule.cfg`, the following information can be found.

```
* This is the RFModule technology file describing circuit parasitics
*for a given process (CMOS, Bipolar, etc)
```

```
# Possible devices: resistor, capacitor and inductor.
#CSMD#
Device:capacitor
Terminal:IN OUT
CSMD IN 1   C
RSMD  1 2   R
LSMD  2 OUT L
#

# At this moment, the first device in each line
# must be R for a resistor, C for a capacitor and
# L for an inductor. Furthermore, the values in the
# first column must be in ascending order, otherwise...
#300f340f#
C=300f R=1m L=0.01n
C=320f R=2m L=0.01n
C=340f R=3m L=0.01n
#

#50p80p#
C=50p R=5m L=0.010n
C=70p R=7m L=0.015n
C=80p R=8m L=0.020n
#

#LBOND#
Device:inductor
Terminal:IN OUT
LBOND IN  1   LS
RBOND 1   OUT RS
#

#350p450p#
LS=350p RS=0.0042
LS=450p RS=0.0054
#

#60n100n#
LS=60n  RS=0.718
LS=100n RS=1.196
#
```

Two devices, one capacitor and one inductor are defined in parallel with the sub-circuit model and device range. Section 7.5 break down the format used in the parasitics configuration file.

Command Line

```
./asco -eldo bandpass
```

Optimization Results Analysis

After about one hour CPU time, which depends on the simulator used, the optimization ends. It should be noted that with the current settings and parasitic values, the pass-band ripple is not fulfilled. In this case, the optimizer returns the best found values, i.e., the sizes the give the minimum cost.

The example is given as a demonstration of capability of using device parasitics during the optimization loop. The components in the `rfmodule.cfg` show extremely low parasitics and do not represent accurate real world values. However, this example has shown how simple it can be to consider parasitics effects during an optimization loop.

6.3 HSPICE® Examples

Latest tested version

Up to 2004.

6.3.1 Tutorial #1 – Digital inverter

Command Line

```
./asco -hspice inv
```

6.3.2 Tutorial #2 – Three stage operational amplifier

Command Line

```
./asco -hspice amp3
```

6.3.3 Tutorial #3 – Class-E power amplifier

Command Line

```
./asco -hspice classE
```

6.3.4 Tutorial #4 – Chebyshev band pass filter

Command Line

```
./asco -hspice bandpass
```

6.4 LTspice™ Examples

LTspice/SwitcherCAD III is a fully functional SPICE simulator with schematic capture and waveform display. The program is available as a free download from Linear Technology.

LTspice™ runs well under GNU/Linux using (Wine which is an Open Source implementation of the Microsoft Windows API on top of X and *NIX). A shell script must exist so that the simulator can be executed by typing `ltspice` in the command line. As an example, the following script file can be used:

```
#! /bin/sh
wine -- "<PATH_TO_SCAD3>/LTC/SwCADIII/scad3.exe" $1 $2
```

while in a win32 platform, create instead the batch file `ltspice.bat` with

```
"C:/Program Files/LTC/SwCADIII/scad3.exe" %1 %2
```

Only for this particular simulator, the log filename is named `<hostname>.log.log` because the output from the simulator already has the `.log` extension.

If the netlist was created using LTspice™, the end-of-file is different than that used in *NIX. As such, the text file must be converted. Use for example the following command:

```
dos2unix <inputfile>.net
```

Latest tested version

Up to 2011.

6.4.1 Tutorial #1 – Digital inverter

In this example, simulation re-runs (using the `.ALTER` command) and Monte Carlo analysis are currently not functional.

Command Line

```
./asco -ltspice inv
```

6.4.2 Tutorial #2 – Three stage operational amplifier

Command Line

```
./asco -ltspice amp3
```

6.4.3 Tutorial #3 – Class-E power amplifier

Command Line

```
./asco -ltspice classE
```

6.4.4 Tutorial #4 – Chebyshev band pass filter

Command Line

```
./asco -ltspice bandpass
```

6.5 Spectre® Examples

Basic support for Spectre®, a non SPICE circuit simulator is implemented. As a result of this, simulation re-runs (using the `altergroup` command) and Monte Carlo analysis are currently not functional.

Although the MDL language has the capability to execute multiple simulations, only one type, currently limited to `dc`, `ac` and `tran`, can exist in the `<inputfile>.scs` without manual editing `<hostname>.mdl` file and changing one (1) line in the C source file. E-mail the developer if you need assistance.

Latest tested version

Up to 2014 (see sub-section 4.5.2)

6.5.1 Tutorial #1 – Digital inverter

Command Line

```
./asco -spectre inv
```

6.5.2 Tutorial #2 – Three stage operational amplifier

Command Line

```
./asco -spectre amp3
```

6.5.3 Tutorial #3 – Class-E power amplifier

Command Line

```
./asco -spectre classE
```

6.5.4 Tutorial #4 – Chebyshev band pass filter

```
./asco -spectre bandpass
```

6.6 Qucs Examples

This is the first free simulator supported by ASCO. Because the executable is a single file, in a multiprocessor environment, it is advantageous to copy the simulator to the same directory where the input files are. This results in the simulator also being copied when the files are copied to other computers. Consequently, the local copy is used instead which results in lower demand on the computer network.

The main advantage of this free simulator is allowing both programs to be built as monolithic executables. The exchange of information can then be performed in memory resulting in tenfold gain in optimization time for simulations that take a few seconds to run.

Latest tested version

Version 0.0.17 (June 2013).

6.6.1 Tutorial #1 – Digital inverter

This example is currently not functional,

6.6.2 Tutorial #2 – Three stage operational amplifier

This example is currently not functional due to the lack of BSIM3 suport.

6.6.3 Tutorial #3 – Class-E power amplifier

This example is currently not functional due to the lack of BSIM3 suport.

6.6.4 Tutorial #4 – Chebyshev band pass filter

Command Line

```
./asco -qucs bandpass
```

6.7 ngspice Examples

The successor of the original SPICE simulator from UC Berkeley is the second free software simulator to be supported. All the four examples are functional with ASCO via the usage of both dot commands and control section.

Latest tested version

Release 24 (January 2012)

6.7.1 Tutorial #1 – Digital inverter

Command Line

```
./asco -ngspice inv
```

6.7.2 Tutorial #2 – Three stage operational amplifier

Command Line

```
./asco -ngspice amp3
```

6.7.3 Tutorial #3 – Class-E power amplifier

Command Line

```
./asco -ngspice classE
```

6.7.4 Tutorial #4 – Chebyshev band pass filter

```
./asco -ngspice bandpass
```

6.8 General Purpose Simulator

As a demonstration of interactions with other programs that are not SPICE/electric simulators, the ASCO tool is used to optimize two parameters in an executable, that also uses the DE algorithm to solve to Rosenbrock Function (6.1). It is in this case optimizing the best input `F` and `CR` control parameters that originate the minimum error in finding the zeros.

$$f(x) = (1 - x) + 100(y - x^2)^2 \tag{6.1}$$

To allow the usage of different simulators that have distinct input arguments, the shell script `general.sh` is executed by ASCO. It is this executable that calls the appropriated simulator with the necessary options. An example of which is given below.

```
#! /bin/sh
./rosen $1.txt $2.out > /dev/null
```

Summary

- 2 optimization variables
- One objective

Full Netlist

Configuration File

```
# Parameters #
Constant F:#F#:0:0:2:LIN_DOUBLE:OPT
Crossover factor CR:#CR#:0:0:1:LIN_DOUBLE:OPT
#

# Measurements #
COST:---:MIN:0
#
```

Command Line

```
tar -zxvf ASCO-<version>.tar.gz
cd ASCO-<version>
make -B
cp asco examples/rosen/

# Rosenbrock's function
cd examples/rosen/bin
```

```
make
cp rosen ..
cd ..

#Execute optimizer
./asco -general rosen.txt
```

Optimization Results Analysis

In the last line of the `<hostname>.log` should read

```
+cost:4.930381E-31:    COST:4.930381E-32:    F:4.629414E-01:CR:6.327368E-01:
```

being as such F=4.629414E-01 and CR=6.327368E-01 in conjunction with the other control parameters in the file `rosen.dat`, the best set of values to solve the Rosenbrock Function (6.1).

Chapter 7

Tools and Modules

A set of external functions have been developed as a "plug-and-play" extension of the original functionality of ASCO. Some of them have been integrated into the optimization flow, while others are intended to help analyzing the optimization results. Not all simulators are supported in each one of the tools.

7.1 alter

SPICE syntax that is used to re-run the same netlist with different options using the command `.ALTER` (in Eldo™ and HSPICE®), `altergroup` (in Spectre®) and `altermod/alter` (in ngspice) are entered here.

```
# ALTER #
.protect
.inc [slow.mod typ.mod fast.mod]
.unprotect
.temp [-40 +25 +85]
.param V_SUPPLY=[2.0 2.1 2.2]
.param Ibias=[0.7 1.3]
#
```

The above line format is dependent on the selected SPICE simulator due to the existing variations in the input format. The exactly same type of parameters, devices and commands available with the SPICE simulator can be used in here. To test PVT corners, the options must be enclosed in square brackets (`[ ]`) with only one space character between them. For example, if only one line exists and is the following:

```
.inc [slow.mod typ.mod fast.mod]
```

upon expansion, three re-runs will be executed for the same netlist:

```
.ALTER @1
.inc slow.mod

.ALTER @2
.inc typ.mod

.ALTER @3
.inc fast.mod
```

A total of $3 \times 3 \times 3 \times 2 = 54$ re-runs for the complete above example for Eldo™ are necessary before the cost value can be obtained and returned to the optimizer. Use only the necessary lines, because upon expansion (to all possible combinations), the total number of simulation re-runs can rapidly grow, with the consequent increase in the optimization time. Besides the speed, no other caution seems to exist at this time.

Should a file with the name `alter.inc` exist in the running directory, this file is included in the netlist instead of adding the parameters in the category. This is useful in cases where it is important to use an existing file with re-run commands.

The `alter` tool can be used integrated in the ASCO optimization flow or as a standalone program. In this case, a file named `alter.inc` having the `#ALTER#` commands is created, which can then be included in the simulation netlist.

7.1.1 Spectre®

For Spectre® a restriction exist in that the command `altergroup` must be present and must be also the first line in the configuration file after the `#ALTER#` category name. Use for example

```
# ALTER #
ag altergroup {
  simulatorOptions options temp=[-40 +25 +85]
  parameters xvdd=[2.0 2.1 2.2]
  include "n.typ" section=[slow typ fast]
}
dcOp_ag dc oppoint=logfile
dcOp dc oppoint=logfile
#
```

Command Line

```
./alter -<eldo|hspice|spectre|ngspice> <configfile>
```

7.2 log

The `log` tool can only be used as a standalone program. To obtain a summary of the simulation optimization measurements (`# Measurements #`) alongside with the parameters (`# Parameters #`), execute in the command line:

Command Line

```
./log <hostname>.log <outputfile>
```

7.3 monte

```
#Monte Carlo#
NMOS_AVT:12.4mV          $ This values will be divided by sqrt(2) by the program
NMOS_ABETA:7.3%          $ 'm' parameter is taken into account
PMOS_AVT:10.9mV          $
PMOS_ABETA:3.7%          $
SMALL_LENGTH:0.0um       $ Small transistors if l<= SMALL_LENGTH
SMALL_NMOS_AVT:20mV      $ Small transistors parameters
SMALL_NMOS_ABETA:10%     $
SMALL_PMOS_AVT:10mV      $
SMALL_PMOS_ABETA:5%      $
R_DELTA:0.333%           $ Resistors matching at 1 sigma between two resistors
L_DELTA:0.333%           $ Inductors matching at 1 sigma between two inductors
C_DELTA:0.333%           $ Capacitors matching at 1 sigma between two capacitors
#
```

Parameters describing device parameter mismatch following the Pelgrom's MOS transistor models are defined in here. Only the numerical values between the colon and the unit can be changed. This is also possible for circuit passives: resistors, inductors and capacitors.

The `monte` tool can be used integrated in the ASCO optimization flow or as a standalone program. In both cases, a file named `<inputfile>.mc` having the Monte Carlo parameters is created, which can then be used as the input simulation netlist.

Command Line

```
./monte -<eldo|hspice> <inputfile>.* <configfile>
```

7.4 postp

```
# Post Processing #
#
```

Sometimes it is necessary to further manipulate a given value to obtain the final measurement. Some situations where this functionality is useful is:

- Take a voltage and divide by a resistance value, or, simply because the SPICE extraction command do not allow further arithmetic.
- The other possibility is to parse information from the netlist that is not possible to obtain using the SPICE extract command.
- Or when it is necessary to extract/measure data at a given position in the simulation output file. Data that is readily available in the SPICE output file be default.

To this end, the flexible post-processing language implemented allows to parse data existing in the output file and manipulate it to obtain values that are otherwise not possible. The general syntax is implemented in the `MEASURE_VAR` command where the values are in a colon separated list. The maximum number of measurements, i.e., the number of `MEASURE_VAR` lines is only limited by the memory available. The value is hardcoded in the C code and can be changed as desired. Each one of the five parameters that can be used are now described:

- `MEASURE_VAR`

  The name to give to the measurement (arbitrary).

- `SEARCH_FOR`

  The string that is to be found. Depending on the measurement type, it must be an exact match.

- `S_COL`

  The beginning column position where the search string specified in `SEARCH_FOR` must exist to obtain a valid result.

- `P_LINE`

  The number of lines (below the line where exists a string match) to read before extracting the measurement.

- `P_COL`

  The number of columns range within which the data is to be measured.

- `MATH`

  Enters in math mode and thus enables manipulation of data already measured.

Four simple type variations have been implemented, being each one of them better tailored to a given application with the goal to minimize user work.

1. General purpose variable extraction: is a generic method to obtain a value, being as such the one that requires more input parameters. It can extract any number or text in any point of the file based on a set of keyword values. For example

   ```
   MEASURE_VAR:UGF: SEARCH_FOR:' UGF =': S_COL:01: P_LINE:00: P_COL:09:18
   ```

   will look for the `' UGF ='` (without the quotation marks but including the spaces), starting at position 1, the first character of the line, and print what is in the current line between columns 9 and 18. As such, 7.6280E-04 will be read to variable `UGF` if the following line exist in the text file to be processed:

   ```
   UGF =  7.6280E+04
   ```

   As an example, the same measurement can be performed by using instead, each one of the following examples

   ```
   MEASURE_VAR:UGF: SEARCH_FOR:'UGF =': S_COL:02: P_LINE:00: P_COL:09:18
   MEASURE_VAR:UGF: SEARCH_FOR:' UGF = ': S_COL:01: P_LINE:0: P_COL:8:18
   MEASURE_VAR:UGF: SEARCH_FOR:'UGF = ': S_COL:02: P_LINE:0: P_COL:9:18
   MEASURE_VAR:UGF: SEARCH_FOR:'UGF =  ': S_COL:02: P_LINE:0: P_COL:8:18
   MEASURE_VAR:UGF: SEARCH_FOR:'GF = ': S_COL:03: P_LINE:00: P_COL:09:18
   ```

2. Fast read: upon finding on a given line a user defined string, the data that follows is read. The data of interest is assumed to be within the character(s) space(s). Only parameter `SEARCH_FOR` can exist for syntax correctness. An exact match throughout the file must be enforced which otherwise would case incorrect data extraction. As an example, each one the following command line is equivalent to the command lines in the general purpose variable extraction.

   ```
   MEASURE_VAR:UGF: SEARCH_FOR:' UGF ='
   MEASURE_VAR:UGF: SEARCH_FOR:'UGF ='
   MEASURE_VAR:UGF: SEARCH_FOR:' UGF =  '
   ```

3. MOS transistor variable extraction: given a transistor name and a list of parameters (id, vgs, vth...), their value is measured. Parameters `S_COL` and `P_COL` cannot exist, only `P_LINE`). For flexibility, text and line numbers can be used for variables. If necessary +,-,*,/ can be used as function.

   ```
   MEASURE_VAR:m00: SEARCH_FOR:'M00': P_LINE:vth: vgs: id
   MEASURE_VAR:m00: SEARCH_FOR:'M00': P_LINE:  9:   6:  3
   MEASURE_VAR:m00: SEARCH_FOR:'M00': P_LINE:vgs: vth: vds:
   MEASURE_VAR:m00: SEARCH_FOR:'M00': P_LINE:vgs :vgs-vth: vds-vdsat
   ```

4. Mathematical mode: implements a simplified RPN calculator. It can be used to do arithmetic manipulation on already measured values. Possible arithmetic functions are: +, -, *, /, ^, abs, log10 and sqrt. The syntax is compatible with Octave/MATLAB® format with the exception of '^' which is the square function. `MATH` is the only parameter allowed. To access data already in memory the character `&` must be used when referring to a variable. For example, upon reading some data

   ```
   MEASURE_VAR: a: SEARCH_FOR:'M00': P_LINE:vgs
   MEASURE_VAR: b: SEARCH_FOR:'M10': P_LINE:vth
   ```

   operations on the previous values are possible as described:

   ```
   MEASURE_VAR:20*log(a/b): MATH:&a: &b: /: log10: 20: *
   MEASURE_VAR:a+b        : MATH:&a: &b: +
   MEASURE_VAR:abs(a)     : MATH:&a: abs
   MEASURE_VAR:a^2        : MATH:&a: ^
   ```

The `postp` tool can be used integrated in the ASCO optimization flow or as a standalone program. In this case, up to three files: `nosat.txt`, `summary.txt` and `<inputfile>.?jr` can be created. The first has a report of the transistors that are not simultaneously in strong inversion (V_{GS} <V_{TH}+margin) and saturation (V_{DS} <V_{Dsat}+margin); a synopsis of the measurements is available in the `summary.txt`; in the last file, the output simulation file is updated with the operating state of the CMOS transistors. The standalone program, also accepts a few other parameters that can be added to the correct category.

```
# Post Processing #
CREATE_LJR:yes
PRECISION:3

VOVD:150mV
VOFF:100mV
VDST:100mV

SKIP_NOSAT=0:mi1*; 0:mi2/i7; 0:mi3*; 0:mi4/i23
SKIP_NOSAT=0:mi0/i14
#
```

Each one of the reserved words above can be used to fine-tune the comments and operating point of the electric simulation output file. A description follows:

- `CREATE_LJR`

  Possibilities are `yes` or `no`. Useful when it is only necessary to extract the measurements, using the `MEASURE_VAR` command.

- `PRECISION`

  Numeric precision of the results printed to `summary.txt`. Not yet implemented.

- `VOVD`

  Overdrive voltage margin: if V_{GS} <V_{TH}+`VOVD`, the transistor is in the weak inversion region.

- `VOFF`

  Off voltage margin: if V_{GS} <V_{TH}-`VOFF` the transistor if in the off region.

- `VDST`

  Saturation voltage margin: if V_{DS} <V_{Dsat}+`VDST` the transistor is in the linear region.

- `SKIP_NOSAT`

  Skips from the `nosat.txt` file, all transistors in the semicolon separated list, or all those with the given prefix if the asterisk (*) exists. More than one line can be used.

Command Line

```
./postp -<eldo|hspice|ltspice|spectre> <inputfile>.* <configfile>
```

7.5 RF module

Unless there is an accurate and scalable model of passives, the edge is on table lookup form. This is in spite of their lack of portability and discontinuity. For devices such as capacitors, pads, interconnections and transistors, the parasitics dependent on the layout implementation and can, with limited complexity, be manually extracted by the designer. Coil parasitics, on the other hand, need more attention and must be analyzed by an external tool such as FastHenry or ASITIC. Then tables are generated and typically includes about 10 values around an anticipated value for the target application. Since the tables are generated only once for each technology the computational time is less important. Measured results and manufacturer data sheets can also be used if available. Other values for the devices are then linearly interpolated. Continuity and portability across distinct electric simulators is created by the RF module, an extension to ASCO. The simulation overhead is unnoticeable owing to the fact that all the necessary data is read to memory in the beginning, and taken from there during the optimization loop.

The file format of `rfmodule.cfg` is composed by groups of subcircuit definition and variable sets of tabled data to describe either different devices, different maximum operating current or different layout information.

```
* This is the RFModule technology file describing circuit parasitics
*for a given process (CMOS, Bipolar, etc)
```

```
# Possible devices: resistor, capacitor and inductor.
#CSMD#
Device:capacitor
Terminal:IN OUT
CSMD IN 1    C
RSMD  1 2    R
LSMD  2 OUT  L
#

# At this momment, the first device in each line
# must be  R for a resistor, C for a capacitor and
# L for an inductor. Furthermore, the values in the
# first column must be in ascending order, otherwise...
#1p5p#
C=1p R=250m L=1n
C=2p R=250m L=1n
C=4p R=250m L=1n
C=5p R=260m L=1.1n
#

#5p7p#
C=5p R=260m L=1.1n
C=7p R=300m L=1.5n
#

#7p9p#
C=7p R=300m L=1.5n
C=9p R=320m L=1.6n
#
```

The above example describe the device `#CSMD#` as being a `capacitor` with `IN OUT` as the two nodes of the subcircuit. The next three lines describe the circuit components. After the subcircuit definition, the tabled values for each one of the devices is given. In the example, keys `#1p5p#`, `#5p7p#` and `#7p9p#` are different groups of characterization data. Arbitrary key names can be used, as long as it matches the symbol in the circuit input netlist file exactly.

The first line in the subcircuit definition is the device having parasitics, while the first column is always the device value with the following columns representing the parasitics.

To not interfere with the normal electric simulator flow, devices having parasitics have this instruction after the in-line comment, which is specific for each simulator. As an example, in the following SPICE input netlist for the case of Eldo™, it must be placed after the `!` character.

```
C0 1 2 3p
C1 1 2 3p !
C2 1 2 3p ! #CSMD_1p5p#

C3 1 2 #C0#
C4 1 2 #C0# !
C5 1 2 #C0# ! #CSMD_1p5p#
C6 1 2 #C0# ! #CFLUX_1p5p#

xC7 1 2 CSMD.SUB C=1 R=1
xC8 1 2 CSMD.SUB C=1 R=1 !
xC9 1 2 CSMD.SUB C=3p R=_RF_ L=_RF_! #1p5p#

xCA 1 2 CSMD.SUB  C=#C0# R=_RF_ L=_RF_ ! #1p5p#
xCB 1 2 CFLUX.SUB C=#C0# R=_RF_ L=_RF_ ! #1p5p#

L0  1  2  #L0# ! #LSMD_1nH390nH#
xL1 1  2  LSMD.sub L=390n R2=_RF_ k=_RF_ C1=_RF_ R1=_RF_ ! #1nH390nH#
```

1. The two first devices in the first group do not have parasitics. Element `C2` according to command `#CSMD_1p5p#` is a subcircuit as defined in the file `rfmodule.cfg` with the tag `#CSMD#`. The tabled date is to be taken from tag `#1p5p#`.

2. Only the two last devices of the second group have parasitics. One is of type `#CSMD#` while the other is of type `#CFLUX#`. Both use the same subkey `#1p5p#`, but this refers to different tables, as they are local to each one of the existing subcircuits.

3. Should a device be introduced in the subcircuit format, to represent for example a measurement done on a black box, only the subkey `#1p5p#` can be defined. String `_RF_` is used to represent the values that are to be obtained from the tabled data.

4. Two different subcircuits are defined, having both the same subkey. Again, despite having the same name, they represent different sets of data. This is because each table is local to each subcircuit.

5. In the last group, an example on how to use an inductor having parasitics is given, for the two possible input formats.

Parameter dependent parasitic

In situations where the device value is the result of function evaluation of the tabled data, it has to be clearly indicated. An example is given below for a resistance which has its value equal to $k\sqrt{f}$, but where the input parameter is only k. In this situation, place the necessary parameter after the in-line comment that is specific to the simulator to use.

```
#LSMD#
Device:inductor
Terminal:IN OUT
LL   3 OUT L
RR2 IN 1   R2
RRV  1 3   VALUE={k*sqrt(FREQ)} ! k
CC1  1 2   C1
RR1  2 OUT R1
#

#1nH390nH#
L=1.6n  R1=2  R2=0.001 C1=0.030p k=6.50E-06
L=10.0n R1=35 R2=0.010 C1=0.043p k=2.64E-05
L=21.9n R1=13 R2=0.050 C1=0.054p k=4.80E-05
L=51n   R1=30 R2=0.010 C1=0.050p k=1.17E-04
L=100n  R1=25 R2=0.010 C1=0.060p k=2.34E-04
L=390n  R1=41 R2=1.052 C1=0.059p k=7.69E-04
#
```

Chapter 8

Adding new Simulators

ASCO is designed to be an encapsulation to a SPICE simulator with the purpose of presenting only a numeric cost to the optimizer. Only a few lines of code are required to add support to a new simulator but a few requirements must be satisfied:

1. The simulator program must read and write to text files.
2. Without the user intervention, it must be possible to start the simulator which must exit upon finishing the simulation.

Although designed to be an encapsulation of a SPICE simulator, the netlist parser is programmable which means it can read any ASCII file and look for a particular sequence of characters. About 100∼150 new lines of code are required, taking in total less than 2 hours for a new simulator to be added. Consequently, it is not specific to SPICE simulators. However, a higher degree of work is necessary if the simulator flow differs from those simulators that are already implemented.

8.1 Where to start editing

Start by editing the file `asco.c` around `Step3` after the line where it reads `spice=0;`. Commercial electronic circuit simulators have a lower numbering (in the variable `spice`) since they were first supported. Eldo™ is number 1 because this was the first SPICE simulator to be supported by ASCO. Then came HSPICE® which has number 2, and so on. Open source simulators start at number 50 to make possible to add more commercial simulators in the future, while keeping some logic in the numbering.

Depending on the intended level of support by ASCO, more or less code and files will have to edited. Basically, all `switch(spice)` code sections will have to be adapted to the new simulator. If it bares resemblance to another already supported, just adding a `case` will be enough. Otherwise, the difficulty really depends on how well support for the new simulator is desired.

8.2 Where to continue editing

Once a new simulator has been added to the file `asco.c`, next steps simply require following the code flow and adapting the code in the `switch(spice)` code sections. No two simulators are alike but sometimes some things are done exactly the same way. At this moment, ASCO already covers a reasonable number of different simulators. As such, most of the `switch(spice)` code sections are already present. Should this not be the case, a new one must be started.

Another option is to compile the code, run ASCO and read the file and section were the programs exits. Correct the code, compile and execute ASCO again and wait until all things run well. Certainly, not the best way.

Chapter 9

Adding new Optimizers

First you need to find the source code of a free algorithm should you not want to invent one. Once this happens, edit the file `asco.c` around the line where it reads

```
DE(argc, argv); /*Rainer Storn and Ken Price Differential Evolution (DE)*/
```

Within the optimization algorithm code, replace where appropriate, every call to the function evaluation by the new evaluation function

```
evaluate(int D, double x[], char *filename); /* obj. funct. */
```

Above, `D` is number of parameters of the cost function, `x` contains the parameters proposed by the optimizer routine, and finally, `filename` is the SPICE file containing the netlist to optimize.

Edit the `Makefile` to include the new source file. Then, simply compile the code with `make`. The new optimization algorithm is then ready to be used by ASCO.

Chapter 10

Development Roadmap

This is still a project in its infancy with the objective of having a modular implementation. This is the reason why changes in the goals and architecture are quite possible. There are however a few things that at this moment look certain:

1. ~~Merge the input/output format of the various tools that make up ASCO, to have an unified script like language.~~ Furthermore, make input and output file names programmable.

2. ~~Include support for logarithmic search space for the existing variables.~~

3. ~~Add hybrid optimization algorithm which couples global optimization for the first steps followed by a local optimization. This can be for example Differential Evolution in conjunction with Hooke-Jeeves or Levenberg-Marquardt algorithm.~~

4. Support other common SPICE/electric simulators.

5. Include the possibility to define which tasks and the order they are to be executed upon an event happening, making thus possible to change to local optimization after the cost has decreased below a user-defined value; start Monte Carlo analysis at given time; re-run the simulator with a modified netlist specifying different analysis; i.e., a user defined optimization flow.

6. ~~Code the RF module, needed for a fast, accurate modeling and optimization of circuit with passives having parasities.~~

7. ~~Implement support for multiprocessor calculations on various architectures. This is simplified by using the work already done by the MPICH project.~~

8. Having a considerable amount of simulation results, it would be useful to take the output from the simulator already available in the file `<hostname>.log` and automatically generate compact symbolic models or some kind of regression representation. Your contribution in this item is welcome.

9. Graphical analysis of the simulation output log file to ease in the discovery of a suitable design. An easy approach would be to use `octave` and `gnuplot`. A more elaborated one uses its own GUI. Again, your contribution in this item is welcome.

10.1 How You Can Help

Your involvement in the ASCO development can be done in different non programmer ways. Not restricted to:

- Proofreading this document: which increases readability of the text and clarifies something that is badly explained, out-of-date or totally wrong.
- Expanding this document: with another section, text reorganization or just submitting a new tutorial. At the end, good documentation is important to clarify doubts and increase productivity.
- Bug reporting: See Chapter 11 for more information.
- New ideas: You can steer the development by showing where you think it will be valuable to invest time. Or better, what is missing that without question will make ASCO more user-friendly.

Obviously, you can also contribute new code, either adding new functionalities or correcting implementation errors.

Chapter 11

Submitting a Bug

Bug reporting is the simplest way you can contribute to the development of ASCO. This is the reason why this tool is released with a GPL license. Your help is always appreciated, because you are improving the quality of a tool that has the possibility to benefit all of us.

Simply saying that a bug exist does not help much. Your results have to be adequately transmitted as well. Also, please understand that we live in a busy world. Your contribution is not forgotten if it takes more time than what you find reasonable. Furthermore, please understand that you might be asked further clarification. Besides this, follow all instructions below closely:

1. Update to the most recent version of ASCO.
2. Try to reproduce the bug with a minimum set of files.
3. Remove the files that you cannot distribute, namely the transistor model files.
4. Send to the developer(s) the complete directory tree in tar.gz format with a description of your problem in as much detail as possible.

A useful text on how to ask questions is available in guide How To Ask Questions The Smart Way by Eric S. Raymond.

Chapter 12

FAQ

None yet...

Chapter 13

Acknowledgments

Some functions used in the ASCO tool were originally written in 1999 and have been maintained since then by the author. Nevertheless, the design flow of ASCO is inspired in the work of [Fra03]. The RF module, of which I'm co-author, is paramount to have accurate high-frequency simulations in the presence of board and passive parasitics is already included in ASCO. In conclusion, the goal of ASCO is to extend the well thought ideas presented in [Fra03], with more functionalities in conjunction with the already existing and more flexible netlist parser, Alter re-runs, Monte Carlo, support for multiprocessor calculations and load balancing.

The idea of creating the ASCO project came out of a discussion with S. Xavier-de-Souza and the concepts around optimization algorithms and its applications. The result of which being, that although ASCO is intended to interact with zero effort with a SPICE simulator, only a handful lines of code are necessary to add support to a completely new simulator, as long as the simulator reads from text files, writes its output in ASCII and it can be launched from the command line.

The Internet is home to a huge amount of information it is not as nicely presented as in Wikipedia. No verbatim copy is done but some ideas where developed after reading its pages.

Bibliography

[Fra03] K. Francken, *A Framework for Analyzing and Synthesizing High-Level and Circuit-Level Analog Blocks*. PhD thesis, K. U. Leuven, Belgium, September 2003.

[Ram05] J. Ramos, *CMOS Operational and RF Power Amplifiers for Mobile Communications*. PhD thesis, K. U. Leuven, Belgium, March 2005.

[SP95] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces", Technical report, Technical Report TR-95-012, ICSI, March 1995.

[Sto96] R. Storn, "On the Usage of Differential Evolution for Function Optimization", In *NAFIPS*, pages 519–523, 1996.