ASCO
A SPICE Circuit Optimizer

Written by João Ramos

Revision : 0.0.1

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 propose new circuit arrangements, but simplifies the design process as fine-tuning, verification and optimization of circuit functionality over process-voltage-temperature corners is automated.

ASCO tool can be seen as the engineer that is willing to make the boring 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

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 result of the previous sentence, the ASCO tool ask for and experienced designer which has the task of selecting the circuit topology; find reasonable operating conditions with realistic design goals; define the test benches and measurements for achieving the desired design objectives; evaluate the proposed sized circuits and elect the most suitable circuit. All this based on knowledge of the application. In return, the tool automates the test of multiple candidates for a given fixed circuit using a high-performance differential evolution (DE) optimization algorithm [SP95]. Different circuit architectures can be tried, falling to the designer the task to select 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. Whereas it has been designed to work with existing SPICE simulators, at this moment 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 select which one is expected to help better achieve their goals.

2.1 Features and Applications

ASCO is the result of an academic research which in itself did not intended 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 DE optimizer 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® and LTSpice™ 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 know to produce good results in an acceptable time.
- Within the supported SPICE simulators, 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 be always readable in the future. In addition, it makes possible to use other existing tools to post-process the optimization results.
- Its free software: the code is available under the GNU GPL license.

ASCO has been designed to address problems that are particular of 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.
- Migrated 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) for 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 5 for ready to use practical examples to introduce you to ASCO, a SPICE circuit optimization tool.

Chapter 3

Installation and Operation

3.1 Installing ASCO

ASCO is written in C. Portability on *NIX type operating systems should follows 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 place so that they can later be used.

3.2 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 3.2.2) 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 accomplish without the user intervention. In most of the cases, only the specification of the parameters range and constraints is sufficient before starting a new optimization.

3.2.1 Encapsulation to the SPICE Simulator

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

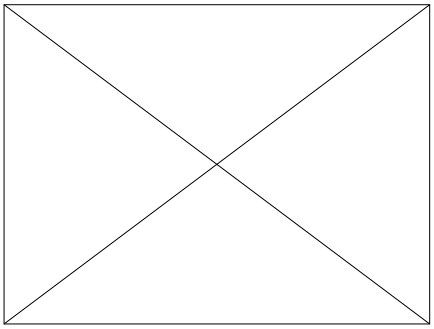


Figure 3.1: Interface between ASCO tool and the SPICE simulation program.

3.2.2 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. 3.1). 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 compared. For each vector $\mathbf{x}_{i,G}$ of generation G, a perturbed vector $\mathbf{v}_{i,G+1}$ is generated as follows:

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) \tag{3.1}$$

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 and 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:

[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 its inverse.

$$\begin{aligned} Cost &= W_{obj} \cdot \sum_{i=1}^{i=n} P_{sim_i} \\ &+ W_{con} \cdot \max_{j \in [1,m]} \left(\frac{P_{spec_j} - P_{sim_j}}{P_{spec_j}} \right) \end{aligned} \tag{3.2}$$

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 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 could 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 (3.2). 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.

3.2.3 Stop Criteria

Allows to stop the optimization under user-defined conditions. Currently defined, they are that the maximum number of generations is reached or the minimum cost-variance is satisfied. See sub-section 3.3.2/Differential Evolution Options for further details.

3.3 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 formated SPICE input netlist. Default extension is `.cir` for Eldo™, `.sp` for HSPICE® and `.net` for LTSpice™.
`<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 output files.
`asco.log`	Optimizer log file.

3.3.1 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 off 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 implies 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.
- 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.

3.3.2 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 in 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 exist 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 #`), 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 separator 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 exist ideas to make them user defined in the order they are executed and in total number of possible steps.

```
#Optimization Flow#
Alter:no            $do we want to do corner analysis?
MonteCarlo:no       $do we want to do MonteCarlo analysis?
Minumum cost:0.00   $point below which ALTER and/or MONTECARLO can start
SomethingElse:      $
#
```

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

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
Crossing Over 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=10xD 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. F is usually between 0.5 and 1 (in rare cases >1). DE is also somewhat sensitive to the choice of the stepsize F.

- `Crossing Over 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 worth to be tried first. 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 (3.2)

- `Cost constraints`

  W_{con} in (3.2)

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

Alter Options

SPICE syntax that is used to re-run the same netlist with different options using the command `.ALTER` 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. Do not use but the necessary lines, because upon expansion (to all possible combination), 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.

Monte Carlo Options

```
#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.

Parameter Options

```
# Parameters #
Supply voltage:#V_SUPPLY#:2.0:0:0:LIN_DOUBLE:---
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 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` or `LOG` (not yet implemented) while y can take `DOUBLE` (continuous)or `INT` (discrete) as possible values for the type of variables.
- 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 with the `Supply voltage`, the symbol to look 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 ignore anyhow. In this case `LIN_DOUBLE` indicates a parameter with double precision with linear parameter variation.

- `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 later be reused.

```
# 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 3.3.3 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 implement, the hardcoded value of 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 there is 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

```
# Post Processing #
#
```

TODO: Commands to parse any type of information from a text file is stored in here upon revising the existing post-processing language.

3.3.3 Extract Commands

Each one of the templates is stored in a file in the `extract/` directory that must resides below the place where the SPICE input netlist is. It is mandatory to name the file as the name used in `# Measurements #` category of the configuration file (Sub-section 3.3.2/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 3.3.2/Measurement Options. Currently, the symbol is automatically filled by the tool.

- `# Post Processing #`

  Sometimes it is necessary to further manipulate a given value to obtain the final measurement.

  - This can be to take a voltage and multiply with with a resistance value, or, simply because the SPICE extraction command do not allow further arithmetic. For these cases, the flexible post-processing language implemented in the tool is used.
  - The other possibility is to parse information from the netlist that is not possible to obtain using the the SPICE extract command, or in alternative, it is readily available in the SPICE output file.

3.3.4 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.

3.4 Invoking ASCO

Copy the ASCO executable to the directory where are your files. You do not really have to, but having everything stored in one place, means that you can move around from computer to computer, without having to worry if the optimizer is installed or even with different versions changing the format somehow. Furthermore, by doing this you can easily compare differences between versions. You decide which option best suit your needs.

To invoke ASCO, simply type at the command line:

```
./asco -<eldo|hspice|ltspice> <inputfile>
```

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

3.4.1 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`. To 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 `Minimum cost` is defined in the configuration file with a high value making that that simulation re-runs start immediately, because the returned cost from the previous simulation is lower than the minimum cost defined to start simulating the PVT corners. The normal flow, if all steeps are to be executed is: optimization⇒Alter⇒Monte Carlo. If an error exist 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 being 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> <inputfile>
```

As programmed today, it is not possible to first execute Monte Carlo simulations and only then the Alter. Monte Carlo always follows the optimization or Alter.

3.4.2 Runtime Messages

To provide some help where the execution is going, there is 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 hope they provide enough information to the user to correct the SPICE input netlist.
- 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.

Chapter 4

Efficient Usage

4.1 Accuracy

In itself, the ASCO tool does not define the accuracy of the results. The algorithm used ASCO has been applied to various set of problems and has obtained good results in term of speed, robustness and convergence. More information is available in the author'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.

4.2 Convergence Speed

In the DE algorithm, alongside with the `choice of method` three parameter deserve special attention to define the convergence speed: `NP`, `F` and `CR`. Electric circuits transfer function being multi-modal require large populations to find a working topology. Likewise, the constant `F` must be above a certain so that premature convergence to a local minimum occurs. In addition to that, parameters are usually dependent which makes that large values of `CR` work better. However, overestimating `NP` and `F` has the consequence that the number of SPICE simulation calls to grow quickly, and thus, slowing 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 and .PZ; .TRAN) to the abolute minimum required to efficiently caracterize 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 gained 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. So far this has not yet been implemented.

4.3 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, can 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.

4.4 Objectives and Constraints

Too much objectives and/or constraints might require a circuit that is simply not possible in theory. A reduced number is advisable whenever wanting to explore a new operating point that is simply to difficult to derive equations to. 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.

4.5 Evaluating Optimization Results

Whenever possible try to understand the reasons behind the optimization results. Always doubt 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 5

ASCO Tutorials

5.1 Getting Started

This chapter contains a description of examples included with the ASCO tool which are grouped by simulator name (Table 5.1). The basic step to prepare a new netlist to the format compatible with the ASCO optimization tool involve:

1. `<inputfile>.*`
    - Take a functional SPICE netlist for your simulator.
    - Measurement commands that are used to assert circuit performance and/or correct operation must not be 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/` below 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. Optimization ends; 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 optimization examples can be found in the directory `examples/` of the ASCO distribution, grouped for each one of the supported simulators.

Table 5.1: ASCO Examples

Circuit Name	Description
inv	Digital inverter
amp3	Three stage operational amplifier
classE	Class-E power amplifier

5.2 Eldo™ Examples

5.2.1 Tutorial #1 – Digital inverter

This simple circuit gives a simple introduction the requirements that are necessary to do before being able to optimize a circuit. The complete set of files described below can be found in `examples/Eldo/inv`.

The fact that only has one variable makes convergence quite fast on modern computers. As such, it is a good example to show most of the ASCO existing capabilities: optimization 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).

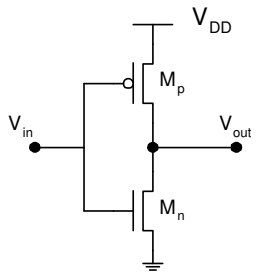


Figure 5.1: Digital inverter.

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 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?
Minumum cost:3.00   $point at which we want to start ALTER and/or MONTECARLO
SomethingElse:      $
#
```

Simulation re-runs are executed immediately upon having a returned cost below 3.00. Immediately, after that because the `Minimum 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 example 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 supply power is the objective. This, while meting 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.

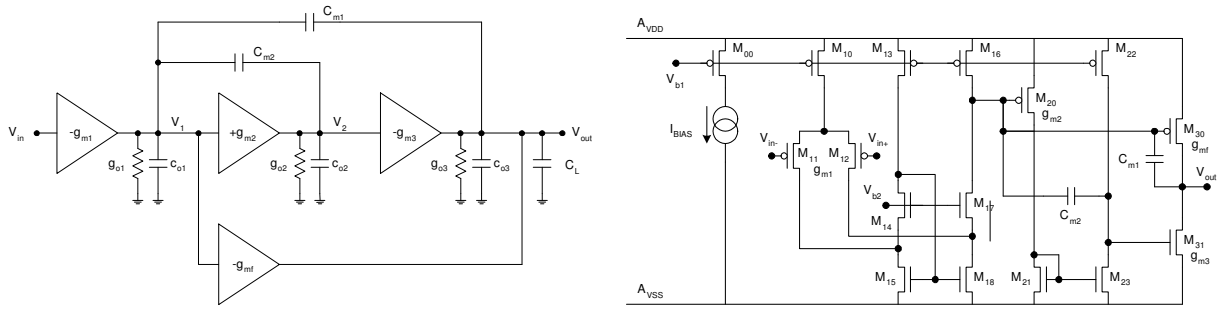
To verify that the optimization as lead to a functional inverter, the binary output file containing the simulation with the best set of values obtained during the optimization is checked with the visual display of the saved output.

In the `<hostname>.log` file, a complete report of all simulations is stored. The last 12 lines refer the the bet test vector (Alter plus Monte Carlo). It can happen 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 behind ending the optimization and a small 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.

5.2.2 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 5.2: 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
Crossing Over 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 arbitrary set to 100, which translated into long optimization times for optimization. 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 parameterized 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 on a single day, a somehow simpler yet accurate representation of the amplifier as depicted in Fig. 5.2(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. 5.2(b) is straightforward.

5.2.3 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`.

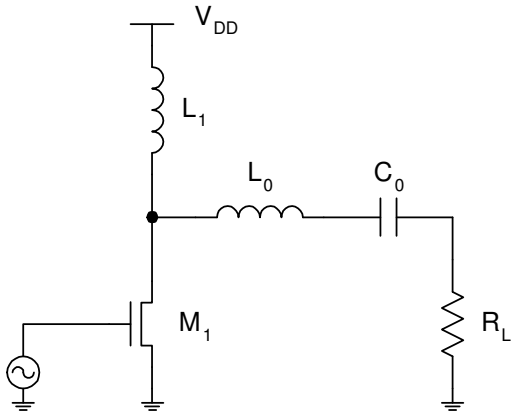


Figure 5.3: Class E power amplifier. It consists of CMOS switch M_1, the finite dc-feed inductance L_1, the series-tuned (L_0-C_0) and the load resistance R_L.

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 show:

```
# DE #
choice of method:3
maximum no. of iterations:50
Output refresh cycle:2
No. of parents NP:60
Constant F:0.85
Crossing Over 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 set 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 meting 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 call 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.

5.3 HSPICE© Examples

5.3.1 Tutorial #1 – Digital inverter

Command Line

```
asco -hspice inv
```

5.3.2 Tutorial #2 – Three stage operational amplifier

Command Line

```
asco -hspice amp3
```

5.3.3 Tutorial #3 – Class-E power amplifier

Command Line

```
asco -hspice classE
```

5.4 LTSpice™ Examples

LTspice/SwitcherCAD III is a fully functional SPICE simulator whit schematich 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 Unix). 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" -b $1
```

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

5.4.1 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
```

5.4.2 Tutorial #2 – Three stage operational amplifier

This example is currently not functional.

5.4.3 Tutorial #3 – Class-E power amplifier

Command Line

```
asco -ltspice classE
```

5.5 General Purpose Simulator

As a demonstration of interacting with other programs that are not SPICE simulators, the ASCO tool is used to optimize two parameters in an executable, that also uses the DE algorithm to solve to Rosenbrock Function (5.1). It is in this case optimizing the best input `F` and `CR` control parameters that originate the minimum error in finding the zeros.

$$f(x) = (1 - x) + 100(y - x^2)^2 \tag{5.1}$$

Summary

- 2 optimization variables
- One objective

Full Netlist

Configuration File

```
# Parameters #
Constant F:#F#:0:0:2:LIN_DOUBLE:OPT
Crossing Over 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 -rosen rosen.dat
```

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 (5.1).

Chapter 6

Adding new Simulators

ASCO is designed to be an encapsulation to a SPICE simulator with the purpose to present 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 to 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 20 lines of code are required, taking less than 5 m. Consequently, it is not specific to SPICE simulators.

Chapter 7

Development Roadmap

This is still a project in its infancy with the objective of having a modular implementation. This is the reason why a 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 simulators.

5. Include the possibility to define which tasks and the order they are to be executed upon an event has happened, 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 parasitics.

7. Implement support for multiprocessor calculations on various architectures. This is simplified by using the work already done by the GAUL 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. Your contribution in this item is welcome.

7.1 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 8 for more information.
- New ideas: You can steer the development by showing where do 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 8

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 9

FAQ

None yet...

Chapter 10

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 not yet included in ASCO. This makes that manual intervention is at this moment very-much necessary. 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, and Monte Carlo support.

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.