Circuit Optimization Using the Simplex Algorithm

Rick Walker, Ken Poulton,
Cheryl Stout, Bill McFarland,
James Kang, Tom Knotts

High Speed Electronics Department, HP Labs
1651 Page Mill Blvd., Bldg. 28C, Palo Alto, CA 94304

ABSTRACT

A large fraction of the circuit design engineer's time is spent in optimization of circuit parameters. An optimization program for SPICE circuit design using the Simplex algorithm is described. This technique can relieve the engineer from much of the tedium involved in circuit optimization. Any number of parameters, such as resistor values, currents and transistor sizes, may be automatically adjusted to optimize an objective measure of circuit performance.

To date, a wide variety of circuits have been optimized, including: several MODFET digital cells, an HP3X high-speed output stage, HP3X and HBT comparators and an HBT sample and hold. Key performance measures have been improved up to 1.5x over hand designs by a skilled engineer.

This paper describes the Simplex algorithm and demonstrates it's application to circuit design. Some pitfalls and heuristics for designing viable objective functions are discussed.

INTRODUCTION

The purpose of this paper is to share the authors' collective experience in the area of computer-aided circuit optimization with HP's design community at large. Our department has long been involved with high speed IC design where a high degree of circuit optimization is required. In the design phase of a typical circuit, a large fraction of the engineer's time is spent doing tedious hand optimization. Often we are working with experimental processes whose parameters are subject to change. In these cases, a given circuit cell may need to be re-optimized for each new variation of the process.

OPTIMIZATION

The first step in setting up a circuit optimization is to derive a quantitative description of circuit performance. Such a performance measure is called an ''objective function''. As an example, an objective function for a digital logic gate would probably include such performance measures as: output risetime, total power dissipation, propagation delay, and so on. We have chosen to define our circuit performance metric so that it is minimized to achieve the highest circuit performance. The objective function may then be intuitively interpreted as a sum of terms which penalize the circuit for any failure to meet the design specification. Once the objective function has been formulated, we can minimize it with any of a number of minima-finding algorithms, thereby optimizing the circuit performance.

There is a wealth of strategies in the literature for finding the minima of a function [1]. Many of the more sophisticated minimization algorithms such as ‘‘steepest descent’’, ‘‘Fletcher-Reeves’’ and ‘‘Marquardt’’, have the drawback of requiring a symbolic calculation of the derivative of the function to be minimized. Since this information is not available from numerical circuit simulation results, these methods are not appropriate for use with SPICE.

The concept of circuit optimization using SPICE or similar simulators is not new. Several systems exist in the realm of university research [2],[3]. Difficulty of obtaining these systems in a form compatible with HPSPICE has forced us to create our own program, however. The optimization strategy we chose uses the Simplex algorithm proposed by Nelder and Mead in 1965 [4]. It requires only function evaluations (no explicit calculation of derivatives) and is simple enough to program and debug easily [5].

It is worth noting that we are optimizing circuit *parameters,* not circuit *topology*. Optimizing circuit topologies for a given task is a *much* harder problem, although there is some AI-based research on this topic at universities [6].

THEORY OF THE SIMPLEX METHOD

In searching an N-dimensional space of the objective function defined by the N parameters of the circuit, it is unrealistic to expect any algorithm to be able to find a global minimum from any starting point. The best that can be done is for the design engineer to provide a starting circuit configuration that is within the ‘‘bowl’’ of the global minimum. The minimization algorithm can then hunt downhill from the starting point to find the minimum of the function. (This should not necessarily be considered a drawback... the need for a skilled operator is why we still have jobs in the computer age!)

Once an initial starting point has been given, how do we converge to a minimum? The algorithm first generates a ‘‘simplex’’: a simplex is a mathematical figure that has one more vertex than the space in which it is embedded. For example, on a plane (2 dimensions), a simplex is a triangle, in 3 dimensional space, a simplex is a tetrahedron, etc.

After generating the initial simplex, the algorithm ranks the vertices according to the value of the objective function at each point. It then takes the worst vertex and attempts to substitute a new, better vertex for it. Improved vertices are generated by either reflection, expansion, contraction or shrinkage. A description of these possibilities for the triangular simplex of a two dimensional optimization is shown in figure 1.

Figure 1. Potential Simplex Moves.

The first attempt at a better vertex takes vertex W (the worst one) and reflects it through the centroid of the other vertices to try point R. If the objective function is better at point R than the 2nd best point N, then vertex R replaces W and an attempt is made to step even further in that direction by trying vertex E. If E is still better, it replaces R in the simplex.

If point R gave a worse value of the objective function than the 2nd best point N, an attempt is made instead to contract the worst point W towards the best points by moving to point C. If C is better than N, it replaces W, otherwise, all the points are shrunk to points S in the direction of the best point B.

Once a new simplex has been constructed by one of the four steps, the vertices are reranked and the cycle is repeated until convergence to the minimum of the function has occured.

Figures 2 shows an initial configuration of a simplex on an objective surface which consists of a u-shaped valley with a minimum at (1,1) with the function evaluations shown as dots. Figure 3 shows the simplex after the first iteration. It has accepted a reflection of the worst point and rejected an expansion attempt that gave a worse value. Figure 4 shows the simplex after the second iteration where a contraction was accepted after an initial reflection try. Figure 5 shows a complete trace of the simplex after it has converged to the minimum of the valley.

FORMULATION OF THE OBJECTIVE FUNCTION

The key to enabling the computer to optimize a circuit is to fully encapsulate all relevant measures of the desired circuit behavior into a single numerical value. This value will then be a function $\chi(P_1, P_2 \cdots P_N)$ of each of the variable parameters. An initial try of such a function for an output stage might be

$$\chi(P_1 \cdots P_N) = T_{risetime} + T_{falltime}.$$

Using this function to drive the optimizer would likely reveal some surprises. It may be possible to choose values of $P_1, P_2, \ldots P_N$ so that $T_{risetime}$ is traded off with $T_{falltime}$. Instead of ending up with 100ps rise and fall, you might get 50ps rise and 140ps fall. An even more disconcerting possibility would be that the optimizer might drive the output voltage swing to nearly zero to achieve the best risetime. A better definition of χ to avoid these two problems would be

$$\chi = T_{risetime} + T_{falltime} + \alpha(T_{risetime} - T_{falltime})^2 + \beta \exp(V_{spec} - V_{swing}).$$

Here we penalize the objective function for any imbalance in the rise/fall times. If it is critical that these times be well matched, we can make α large. The exponential term gives us a large penalty whenever the simulated swing V_{swing} falls below the target spec V_{spec}. (In practice, a piece-wise linear function is generally used instead of an exponential, so as to have zero penalty for performance which meets or exceeds the specification.)

It is often appropriate to include a similar limit on total power dissipation. A second DC analysis might be run to calculate the noise margin of the gate. Overshoot and ringing may need to be considered also. A good strategy is to run the optimizer, analyze the minimized configuration, and then modify the objective function to avoid any unpleasant behavior.

In the *optspice* program described below, the objective function may be composed of multiple terms. Given a measure of circuit performance x, a desired performance x_{good}, and a poor performance x_{poor}, the normalized contribution to the objective function due to the metric x is

$$\mathrm{norm}\left(x, x_{good}, x_{poor}\right) = \max\left(0, \frac{x - x_{good}}{x_{poor} - x_{good}}\right) + \max\left(0, \frac{x - x_{poor}}{x_{poor} - x_{good}}\right)$$

As shown in figure 6, the normalized function is 0 for values better than the ‘‘good’’ value, 1 at the ‘‘poor’’ value, and greater than 1 (at twice the slope) for values worse than the ‘‘poor’’ value. The idea is that 1) if the ‘‘good’’ value is achieved, then no contribution is made to the objective function; 2) terms worse than their ‘‘poor’’ values will be emphasized in the total objective value.

Each of the normalized contributions is then multiplied by a relative weight and averaged together to compose the complete objective function.

Figure 6. Objective Normalization Function.

IMPLEMENTATION

Our initial investigation resulted in a program using the Simplex algorithm that had the circuit topology and the calculation of the objective function written into the program. This required rewriting portions of the program for each change in the circuit topology or objective function and required a detailed understanding of the optimization algorithm. This simple version of the program was used by several members of our department.

Soon it became clear that a more user-friendly interface was needed. The present version of the program, called ‘‘*optspice*’’, provides a simple interface that can be learned with a minimum of effort. *Optspice* operates on a spice deck that includes several command lines recognized only by *optspice*:

The **.opt_param** statement defines the names and starting values of the parameters to be optimized. It also defines starting step size and the error size for convergence.

The **.opt_limit** statement allows the designer to specify hard limits on parameter values. These limits are evaluated by the HPSPICE post(1) program, so fairly general expressions may be used.

The **.opt_obj** statement specifies the names and target values for the objectives for the circuit performance. Multiple **.opt_obj** statements are allowed (with weights); the total objective function is the weighted average of all of the normalized objective terms.

The **.opt_def** statement is used to define how to measure the objectives from the results of the simulation. These expressions are evaluated by post(1) so the full power of that waveform analysis tool is available without additional programming.

Optspice cycles through the following steps:

• Generate New Point
A new point in the parameter space is generated according to the Simplex algorithm.

• Test Limits
The new point is tested against the limits specified by the **.opt_limit** statements by running post(1) to evaluate the expressions.

• Run Spice
A legal HPSPICE input deck is generated and run.

• Evaluate Objectives
Post(1) is used again to determine performance at this point. It generates the single number for the total objective function.

• Save or Discard Point
If the new point is better than any of the old points in the simplex, it is inserted and the previous worst point is discarded.

The cycle continues until the process converges to a set precision or a limit on iterations is reached.

EXAMPLE OPTSPICE INPUT DECK:

```
test3: trivial example of RLC settling optimization

vins %vin 0  pulse (-1 1  100ps 100ps) # input source

### circuit to be optimized
# We wish to minimize the settling time of this RLC circuit
# The R is the variable.
l1 %vin 1  25nH
r1 1 %vout  value=opt.r1
c1 %vout 0 1pF

### parameters to optimize:
.OPT_PARAM r1 start=50  relstep=.3  reltol=.01

### raw file (save only what we need)
.post tr v(vin) v(vout)
.post tr i(vins)

### definitions
# 1% settling tolerance on vin and vout:
.OPT_DEF tol=.01
.OPT_DEF xsettle(w,vtol,t)=xcross(abs(w-yvalue(w,t)),vtol,-1,t)
.OPT_DEF tsett=max(xsettle(vin,tol,10n),xsettle(vout,tol,10n))

# oshootp finds the overshoot in waveform w the time window [x1,x2]
.OPT_DEF oshootp(w,x1,x2)=max((w-yvalue(w,x2)+100)*xlim(x1,x2)-100)
.OPT_DEF xlim(x1,x2)=trunc(time-x1+1,1)*trunc(x2-time+1,1)
.OPT_DEF oversh=max(oshootp(vin,0n,10n),oshootp(vout,0n,10n))

### limits
.OPT_LIMIT  r1>5

### objectives
# Primary objective: short settling time
.OPT_OBJ tsett good=1e-9 poor=3e-9 weight=3
# Secondary objective: small overshoot
.OPT_OBJ oversh  good=5e-3 poor=25e-3 weight=1

.tran 100p 10n
.end
```

EXPERIMENTAL RESULTS

Some circuits which were optimized with one or another version of the optimizer are listed in Table 1.

Circuit	# parms	Objective	initial	optimized	improvement
SAC/MODFET buffer	5	$t_{rise} + t_{delay}$	104 ps	78 ps	1.33 x
HBT comparator	6	min. power at 6GHz clock	2.4 ma	2.3 ma	1.04 x
SAC/MODFET 2:1 Sel.	6	$\bar{t}_{rise} + \bar{t}_{delay}$, 2 nodes	163 ps	132 ps	1.23 x
SAC/MODFET 4:1 Sel.	7	$\bar{t}_{rise} + \bar{t}_{delay}$, 3 nodes	356 ps	278 ps	1.28 x
SAC/MODFET latch	7	$t_{rise} + t_{delay}$	104.5 ps	68.7 ps	1.52 x
HP3X comparator	9	$t_{regeneration} + t_{recovery}$	-	260ps	-
3 stage HP3X driver v.1	12	$t_{transition}$	66 ps	50 ps	1.32 x
2 stage HP3X driver	13	$t_{transition}$	70 ps	52 ps	1.35 x
3 stage HP3X driver v.2	13	$t_{transition}$	64 ps	46 ps	1.39 x
SAC/MODFET driver	13	$\bar{t}_{rise}$, 3 nodes	130 ps	111 ps	1.17 x
HBT sample/hold	7-25	$t_{acq} + t_{settling} + power$	215 ps	140 ps	1.54 x

Table 1. Summary of Optimized Circuits

In general, *optspice* seems to give the greatest performance improvements when the circuit has many parameters with strong interactions. When a particular objective is controlled mainly by one parameter, a human designer can often make quicker progress than *optspice*. However, when more than 2 or 3 parameters interact, a human designer quickly gets bogged down in the repetition of trials and the difficulty of trying to understand the interactions. This is where *optspice* can make a big contribution: it saves the designer's time by letting the computer do the tedious job of multi-parameter optimization.

Sometimes, the designer gives up some understanding of the circuit by letting the computer do the work. At other times, however, we have gained valuable insight into a circuit when the optimization has lead to designs that we did not anticipate, and that we might not have otherwise found.

Usually, there are conflicting constraints and objectives. It has been found that selecting the right set of objectives often involves running *optspice* several times, adding constraints and objectives as needed. Once these are defined, however, it is possible to make circuit or model changes and reoptimize the circuit with a minimum of effort. This allows designers to play ‘‘what if’’ games with circuits or device models with some assurance that the results are optimized for each trial.

As you might suspect, *optspice* is a conspicuous consumer of compute cycles. In the circuits optimized so far, the runtimes of the individual spice runs have ranged from 15 to 120 seconds on an HP 9000/835. Optimization of N parameters generally takes about $2N^2$ spice runs. Including overhead of 10-45 seconds for evaluation of constraints and the objective function, this yields optimization runs ranging from 15 minutes for 4 parameters up to a day or two for 25 parameters.

One of the problems with any optimization scheme is finding the global minimum in the presence of local minima. Two strategies have been employed to work around this problem. The most obvious method is to start circuit optimizations from several starting points. This has been used by several designers; some found that they reached somewhat different final points, while some found little change in the final result. Presumably, the characteristics of the circuit and the nature of the alternate starting points both play large parts in this behavior.

Another strategy is built into *optspice*: after completing one optimization run, another is started from the best point of the first run, with an increased starting step size for each parameter. This seems to find a better point than the first trial about half the time, though the improvement is usually small. In general, it is very hard to tell what the parameter space is like, so knowing how close your solution is to the very best solution is quite difficult.

CONCLUSIONS

A methodology and program for computer-aided circuit optimization has been presented. Such a system can be easy to use, while saving the design engineer from needless tedious work. Actual performance gains for optimized circuits have been observed to be as high as 1.5 X. In most cases, the program outperformed hand optimization.

Although more sophisticated programs exist in the realm of university research, to our knowledge this work is the first application of these techniques to HPSPICE. It is our hope that these results will encourage HP's designers to make use of computer-aided circuit optimization and encourage HP's tool builders to provide more sophisticated optimization tools.

REFERENCES

[1] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, "Numerical Recipes in C", *Cambridge University Press*, Copyright 1988, pp. 291-352, pp. 542-547.

[2] William Nye, David C. Riley, Alberto Sangiovanni-Vincentelli, Andre L. Tits, "DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits", *IEEE Transactions on CAD*, Vol 7, No. 4, April 1988, pp 501-519.

[3] Gen-Lin Tan, Shao-Wei Pan, Walter H. Ku, An-Jui Shey, "ADIC-2.C: A General-Purpose Optimization Program Suitable for Integrated Circuit Design Applications...", *IEEE Transactions on CAD*, Vol 7, No. 11, Nov 1988, pp 1150-1163.

[4] Nelder, J.A., and Mead, R. 1965, "An Algorithm for Least Squares Estimation of Non-Linear Parameters", *Computer Journal*, vol. 7, p. 308.

[5] Caceci, M.S., and Cacheris, W.P. 1984, "Fitting Curves to Data", *Byte*, vol. 9, no. 5, pp. 340-362.

[6] Marc DeGrauwe, et al, "An Analog Expert Design System", *Digest, 1987 IEEE International Solid State Circuits Conference*, pp 212-213