Circuit Synthesis Project

Hershel Millman
1210541979

Introduction

Circuit design is often a tedious process. Designing an analog circuit generally requires one to intimately understand the operation of a circuit, and how each of the design parameters affect the performance. However, when one needs a simple circuit to be designed as part of a larger project, designing such a circuit, although fairly simple, takes time away from other more important tasks. Automating the design process of simple circuits allows engineers to spend their time on more important problems. Additionally, automated processes often produce better results in a shorter amount of time than an engineer could designing by hand.

Approaches to Automation

When it comes to automating the design of a circuit, the simplest method is to brute-force it — to try every possible combination of parameters. With circuits larger than a few parameters, this method becomes infeasible: a circuit containing 10 parameters which have only 20 possible values each, the number of combinations to test would be 10.24 trillion. This is completely impractical.

Another possible method is through machine learning algorithms. However, in order for machine learning algorithms to be effective, often large amounts of data is required. Tens of thousands or hundreds of thousands of already-designed, functional circuits would be required for a neural network or similar algorithm to learn how to create a circuit, and the training time for these networks would be prohibitively long.

Fortunately, there is a method that is not prohibitively computationally expensive, and does not require an exhaustive search of parameters: simulated annealing. Simulated annealing is a “perturb and observe” method, where random changes are made to the circuit parameters, and a scoring metric is used to determine whether the changes should be kept or discarded. Three circuits were synthesized using a simulated annealing method I have come up with, based loosely on a method found in [1].

My Method and Original Circuit

The first circuit used to develop the algorithm was a 5-transistor amplifier. The schematic for this amplifier is shown to the right in Figure 1. The bottom transistor was biased with a voltage source for simplicity. The circuit had four variable parameters: Vbias4, Wp (width of PMOS), Wn (width of input pair), and Wtail (width of tail transistor). The specs required of the circuit were DC gain, 3dB frequency, unity gain frequency, and DC current consumption.

First, the values of the parameters were given random values, with reasonable upper limits and lower limits. Then the program entered the loop consisting of four steps, which are broken down on the following pages. A temperature parameter was also chosen, the purpose of which will be discussed later.

First, the program entered a design loop. Three random numbers were generated: one which decided which parameter to change, one which determined how much to change it, and one which determined

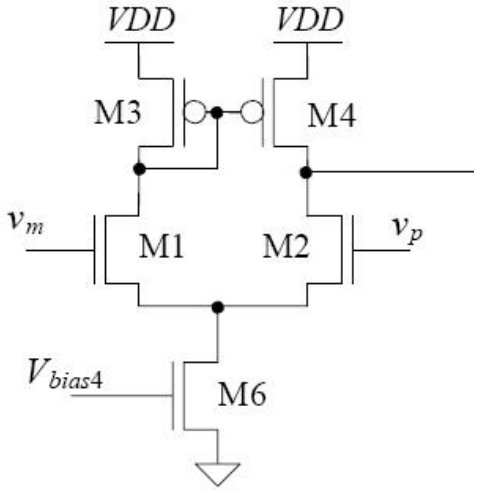


Figure 1: 5T Op-amp

whether the change would increase or decrease the value of that parameter. Then, a simulation was run with these new parameters using open-source circuit simulator ngspice.

Next, the results of the simulation had to be analyzed to determine whether they were valid. The gain and phase curves presented a challenge. Visually identifying a valid gain curve is trivial, but it was necessary to develop a function to programmatically determine the validity of the curve. The method eventually decided on was one which analyzes three attributes of the curve: the flatness of the top near DC, the height of the flat top, and the existence of a unity gain frequency. To determine whether the top of the curve was sufficiently flat, five points were sampled between 10 Hz and 1 kHz, and the relative differences in magnitude were identified. If the difference between the maximum value and minimum value in that range was greater than 0.1 dB, the curve was considered invalid, and the function would return without extracting any more information about that iteration. If the curve was flat on top, but the magnitude of the gain was not greater than 10 dB, the curve was considered invalid. And finally, if the gain curve did not cross 0 dB, the curve was considered invalid. If any of these conditions cause the curve to be considered invalid, the function would return a value to indicate that the shape of the curve was incorrect, which would affect the scoring metric discussed in the following section. If both the gain and phase curves were considered valid, the values of DC power, 3 dB point, and unity gain frequency were obtained from the extraction function. Once these values were returned, the circuit was given a score based on its performance.

Each design specification was assigned a weight based on its relative importance. The weights for magnitude of DC Gain, 3dB frequency, unity gain frequency, and DC current were 2, 1, 1, and 2 respectively, with a weight of 100 given to the shape of the gain curve. If the shape of the gain curve was incorrect, the scores for the other specifications were given default values, which meant that the only way for the score to improve would be for the algorithm to find a set of parameters which produced a gain curve of the correct shape. If the shape of the gain curve was correct, the scores of the other values were calculated using the following formulas. If the spec was an upper limit, such as power consumption, the score would be calculated with the following function:

$$s = 1.2^{(a-g)/g} \quad if \quad a > g, \quad else \; s = 0$$

If the spec was a lower limit, such as DC gain, the score would be calculated with the formula below:

$$s = 1.2^{(g-a)/g} \quad if \quad a < g, \quad else \; s = 0$$

where s is the score, a is the achieved value and g is the spec value. The value 1.2 was chosen through trial and error. The lower the score, the better the circuit had performed.

Once the circuit performance had a score, the program would compare that score to the previous score. If the score was better than the previous iteration, the new values of the parameters would be kept to progress further in the next iteration. If the score was higher, the program would generate a random number between 0 and 1. If that number was lower than the temperature parameter mentioned above, the new set of parameters would be kept, even though it was worse. The temperature chosen was .15, giving a design a 15% chance of being kept even if the performance was worse. The reason for this practice is that it allows the algorithm to avoid getting trapped in local minima, meaning that it is able to optimize on a more global scale.

The algorithm would continue to loop until a design was found which had a performance score of 0. At this point, the program would return, print out the values of the parameters, and plot the progressions through the iterations. A sample output is shown at the top of the following page in Figure 2.

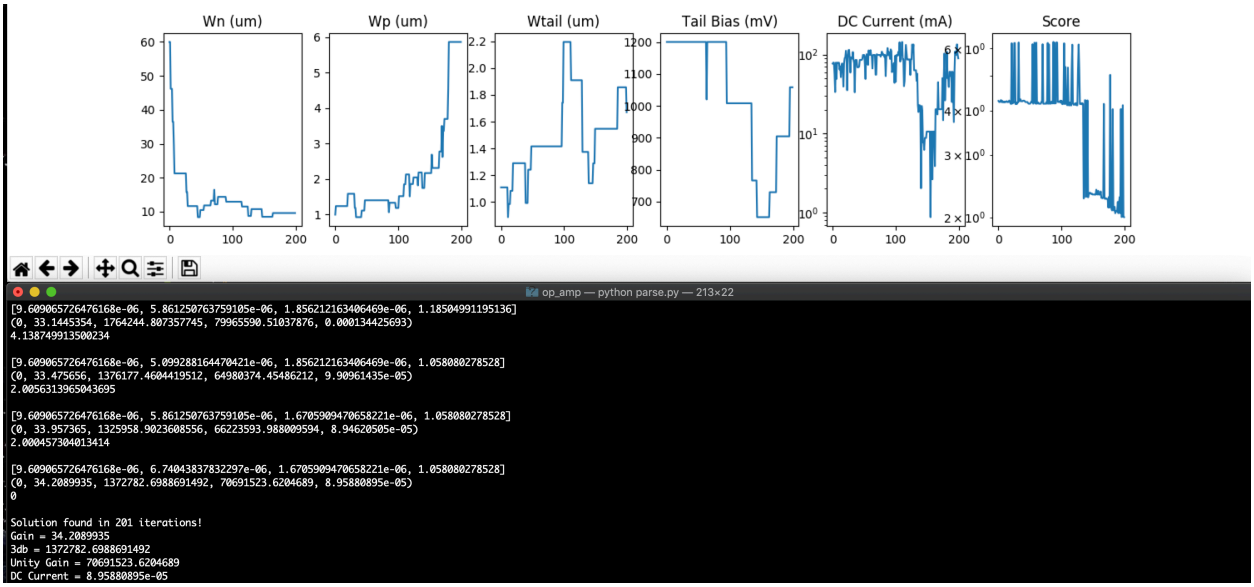


Figure 2: Progression of Parameters and Final Results

The algorithm generally converges within 250 iterations, and takes approximately 8 seconds to arrive at a solution. As can be seen from the plot of Wtail, the parameters can vary upwards and downwards and are not necessarily trapped in one region or range of values. This simple circuit was able to be solved fairly quickly, but what about something a bit more complicated?

Two-Stage Miller-Compensated Op Amp

The next circuit to be synthesized was a two-stage Miller compensated operational amplifier, the schematic of which is shown to the right in Figure 3. The parameters were given reasonable ranges, and a similar algorithm was used to synthesize parameter values. However, the heightened complexity of the circuit required some changes to the algorithm. The 5-transistor op amp previously simulated had only 4 parameters and 4 specs. This amplifier has 10 design parameters, and was designed to meet 5 specifications. One major change was made to the scoring function for this design: the shape of the phase curve was taken into account, due to the more complicated AC response introduced by the Miller compensation. In order to make the test computationally efficient, the

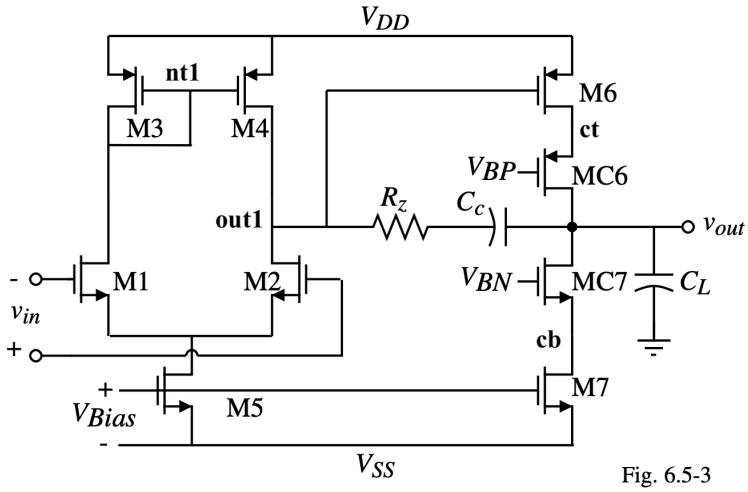


Figure 3: Two-Stage Miller Compensated Op Amp

phase curve was analyzed to ensure that each value along the curve was lower than the one preceding it. This ensured a monotonic response and allowed for a fast check for validity.

Designing this amplifier by hand took my peers roughly a week in my Analog Integrated Circuits course. I had written a script to perform the design for me, but that script utilized the brute force method, and checked every possible transistor value on the grid within a reasonable range. This process took upwards of **14 hours** of simulation time on the EECAD servers. Using my simulated annealing method, the design shown below in Figure 4 was synthesized to meet all required specs in 1263 iterations in only **1 minute and 2.83 seconds** using random initializations of parameters.

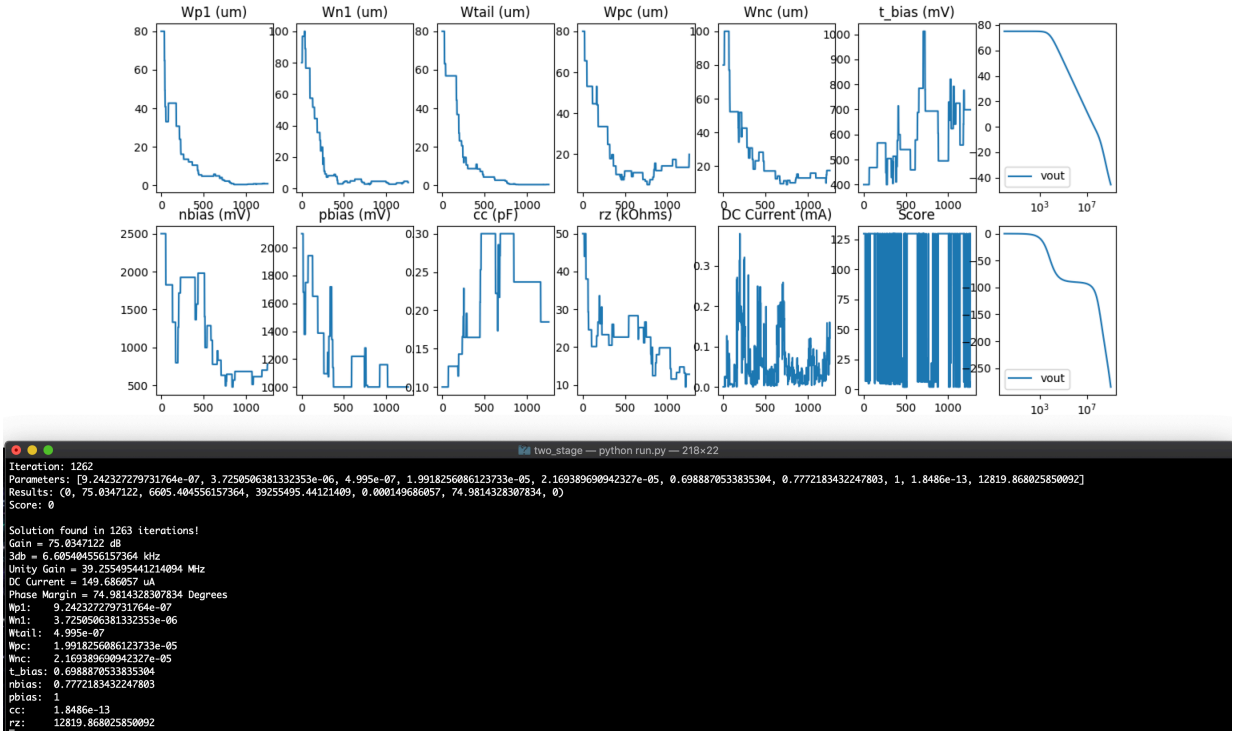


Figure 4: Results of Two-Stage Op-Amp Synthesis

Once these results were obtained, it was time to move on to an LDO.

LDO Design

The LDO architecture chosen is shown to the right in Figure 5. Instead of using a bandgap reference I used a voltage source, for simplicity of simulation. The amplifier used was an OTA with a current mirror, the schematic of which is shown on the following page in Figure 6. To achieve a more robust design, the goal was to design over a range of operating conditions. The specs that were initially designed for were open loop phase margin, and PSRR. The dropout voltage was designed to be .95 V.

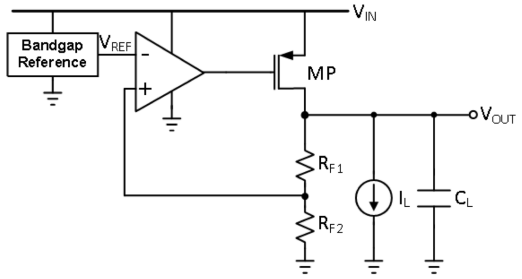
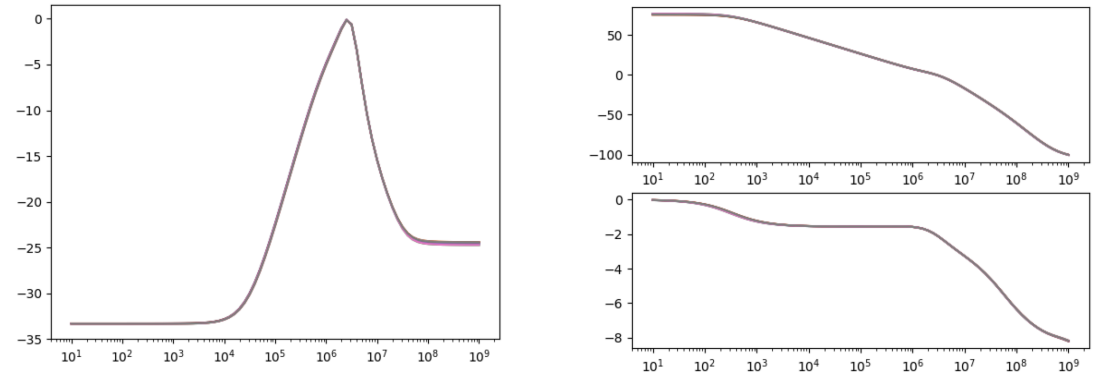


Figure 5: LDO Architecture

The specification for PSRR was -28 dB, and the specification for phase margin was 60 degrees. The corners to meet the specifications are tabulated below in Table 1:

Input Voltage	Output Voltage	Load Current
2.4	2.1	.15mA
2.6	2.1	.15mA
2.4	2.3	.15mA
2.6	2.3	.15mA
2.4	2.1	20mA
2.6	2.1	20mA
2.4	2.3	20mA
2.6	2.3	20mA

However, even though there were only two specifications to meet, the algorithm was not robust enough to meet them easily. It took many attempts and different starting points to even converge one time. The curves for PSRR and Gain and Phase are shown below in Figure 6, along with the final circuit parameters.



```
Solution found in 67 iterations!
phase_margin = 60.82894278309594
wp1:    3e-05
wn1:    1.443e-05
wtail:  1.13e-05
k:      4
wbig:   0.014
r1:     840000.0
r2:     1300000.0
r0_val: 2548.98
c0_val: 5.3808e-11
3.0131047251148875
[3e-05, 1.3e-05, 1.13e-05, 4, 0.014, 840000.0, 1300000.0, 2142.0, 4.56e-11]
```

Figure 6: PSRR, Gain and Phase, and Circuit Parameters

Unfortunately, when the ranges of the current or voltages were increased even slightly, the algorithm became too unstable and would not converge.

Conclusion

The algorithm I have come up with has proven itself to be highly capable with some circuits, and very lacking with others. The performance of the algorithm when it comes to amplifier design is incredible, and allows for extremely fast design and optimization. However, when it comes to LDO synthesis, the algorithm is not as robust. One reason for this is that it is difficult to create scoring methods that are implemented by a computer program that accurately reflect the performance of the LDO. While it is easy to visually determine the quality of a voltage regulator, automating a process that tends towards improvement is difficult and requires more knowledge of the circuit at hand and of optimization techniques.

Future work in this area would be to create semi-random simulated annealing. This could be done by keeping track of which parameters are most closely correlated with which performance metrics, and determining when and how to alter the parameters to improve the score. Additional future work could also include switching between multiple architectures when one is not improving sufficiently to prevent stagnation of results.

The algorithm I have come up with is one that is capable of being applied to many analog circuit design problems, and could be a great time saver for many engineers.

Code

All code and simulation files are in a Google Drive folder accessible at the following link.
https://drive.google.com/drive/folders/11SOqBVCkxysfRO1g3KiloT16IjbT0m3e?usp=sharing

Works Cited

[1] Phelps, R. 2000. ‘Anaconda: Simulation-Based Synthesis of Analog Circuits Via Stochastic Pattern Search’. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 6