AMERICAN
UNIVERSITY OF BEIRUT

MAROUN SEMAAN FACULTY OF
ENGINEERING & ARCHITECTURE

Design Automation of Wireless Connectivity Blocks

Mohamad Zbib[1], Hadi Dayeh[2], Shafik Houeidi[3], Nour Shammaa[4], Tamer Slim[5], Riwa El Kari[6], and Mahdi Zwain[7]

May 18, 2024

Contents

1 Abstract **4**

2 Preliminary Information **4**
2.1 Introduction to Cadence Virtuoso 4
2.2 Installation of Cadence Virtuoso 4
2.3 Cadence Hotkeys 4
2.4 Creating New Library and Cell Views in Cadence 4
2.5 Opening ADE Assembler 4

3 Introduction **4**

4 Current Mirror **5**
4.1 Current Mirror Overview 5
4.2 Objective 5
4.3 Constraints 6
4.4 Design and Optimization 6
4.4.1 Setting Up the Schematic 6
4.4.2 Setting up the Simulation 6
4.4.3 Simulating and Optimizing 7
4.4.4 Results 7

5 Low-Pass Filter **7**
5.1 Simulation and Optimization 9
5.1.1 Setting Global Variables 9
5.1.2 Setting AC and Noise Tests 10
5.1.3 Setting AC and Noise Outputs 11
5.1.4 Running the Simulation 12
5.1.5 Reducing Variables 12
5.1.6 Regulating Variables 13
5.1.7 Final Results 14
5.1.8 Problems that Might Arise and their Solutions 14
5.1.9 Exporting OceanScript File 15
5.2 Running the OceanScript Simulation 15
5.2.1 Create the Ocean Script 15
5.2.2 Modify the Ocean Script 15
5.2.3 Implement the Value Substitution 15
5.2.4 Execute the Script in Cadence Virtuoso 15
5.2.5 Create a CSV File 16
5.2.6 Read the CSV File 16
5.3 Updated Optimized OceanScript 17

6 Differential Amplifiers **17**
6.1 Introduction and Objectives 17
6.2 Design Constraints 17
6.3 Steps of Execution 17
6.3.1 Building the Circuit Schematic 17
6.3.2 Setting Global Variables 18
6.3.3 Setting AC and NZ Analyses 18
6.3.4 Adding Constraints to the Output Setup 18
6.3.5 Running the Simulation 19
6.4 For M = 4 20

7 Iterative Optimization Algorithms **22**
7.1 Newton's method in optimization . . . 22
7.2 Gradient Descent . . . 22
7.3 Gradient Descent vs Newton's method in optimization . . . 23
7.4 A General Methodology in Circuit Optimization . . . 24
7.5 Pseudo code and Script Analysis . . . 24
7.5.1 Algorithms . . . 24
7.6 Mathematical Analysis of the Optimization Algorithm . . . 25
7.6.1 Objective Function and Optimization Problem Formulation . . . 25
7.6.2 Convergence Analysis . . . 25
7.6.3 Gradient Calculation . . . 25
7.6.4 Step Size Selection . . . 26
7.6.5 Variable Update Rule . . . 26
7.6.6 Termination Criteria . . . 26

8 Using Skill **26**
8.1 Launch . . . 26
8.2 SKILL Functions: . . . 26
8.2.1 Gradient Function: . . . 27
8.2.2 readCSV Function: . . . 27
8.2.3 MinAbsVal Function: . . . 28
8.2.4 Optimization Algorithm Function: . . . 28
8.3 Summary of SKILL Syntax: . . . 29

9 Testing and Results **29**

10 Future Work **29**

11 Conclusion **30**

12 APPENDIX I: Cadence Hotkeys **31**

13 APPENDIX II: OceanScript Samples **32**
13.1 OceanScript 1: . . . 32
13.2 OceanScript 2 . . . 34

1 Abstract

This report documents the advancements in the project focused on the design automation of SOC wireless transceivers operating within the RF frequency range of 2.5-7GHz. Leveraging tools such as Ocean Script, Skill, and Cadence Tools, the project concentrated on the optimization of critical components including the second-order low pass filter and the first-stage differential amplifier. The optimization techniques utilized enabled precise tuning of component values to meet desired frequency responses, enhancing both performance and efficiency. The methodologies employed are detailed within, demonstrating significant enhancements in the automation of RF circuit design. Results highlight the successful application of these methodologies, and the report concludes with suggestions for future work aimed at continuing to advance the field of SOC design automation.

2 Preliminary Information

2.1 Introduction to Cadence Virtuoso

Cadence Virtuoso stands out as a premier Electronic Design Automation (EDA) software suite for integrated circuit (IC) development. With its comprehensive toolset for layout design and simulation, Cadence empowers users to design circuits according to their specifications and simulate them as if in the real world. Additionally, Virtuoso enables engineers to create complex semiconductor devices with precision and efficiency, thereby facilitating technological advancements in the field of ICs.

2.2 Installation of Cadence Virtuoso

The purpose of this guide is to provide a comprehensive overview of the project's accomplishments as of May 2024. Throughout this guide, it is assumed that the reader has already installed Cadence on their laptop. Therefore, a detailed step-by-step setup of Cadence will not be covered. If you have not configured Cadence yet, please contact the IT Department through the IT Helpdesk (it.helpdesk@aub.edu.lb), and they will provide you with the necessary setup guide.

2.3 Cadence Hotkeys

Refer to APPENDIX I.

2.4 Creating New Library and Cell Views in Cadence

To create a new library in Cadence within the Linux environment, begin by navigating to the top left corner of the interface. Click on "File," then select "New," followed by "Library." Next, input the desired name for the new library. After specifying the name, proceed to choose the technology file. It's advisable to attach the new library to an existing technology file, as this ensures compatibility and access to predefined process parameters and design rules. To create a new Cell View, click on "File" once more, then select "New," followed by "Cell view." Choose the recommended type, which is typically "Schematic," and proceed by clicking enter. This action initiates the creation of a new cell view, allowing you to begin designing your schematic within the Cadence environment.

2.5 Opening ADE Assembler

The Analog Design Environment (ADE) Assembler in Cadence is a tool utilized for automating the setup and execution of simulations, facilitating efficient and customizable analysis of analog and mixed-signal circuits. To access a new ADE Assembler, click on "Launch" at the top-left corner of the screen, then select "ADE Assembler." Click on "Open a New View," enter the desired name, and press enter to proceed. This will display the ADE Assembler interface, shown below.

3 Introduction

We contributed to this project which is focused on the development of automated optimized design solutions for SOC Wireless Transceivers operating in the RF frequency range of 2.5-7GHz. Our primary

objective is to leverage powerful tools such as Ocean Script, Skill, and Cadence Tools to achieve efficient and effective designs for various blocks and components.

The core of our project revolves around optimizing three key blocks: the current mirror, the second-order low pass filter, and the first stage differential amplifier. Our goal in the current mirror is to make the reflected current as close as possible to the reference current. For the second-order low pass filter, our goal is to fine-tune the resistors and capacitors to achieve the ideal transfer function at a specific frequency. As for the first stage operational amplifier, our focus is on maximizing gain. Our aim is to design a high-gain differential amplifier. This optimization step is crucial for ensuring superior signal amplification within the wireless transceiver circuit block.

During the optimization process for the second-order low pass filter, we experimented with different techniques to find the ideal frequency response. We employed optimization algorithms such Newton's method to fine-tune the component values using skill. These algorithms allowed us to iteratively update the values based on the filter's response characteristics, aiming to achieve the desired frequency response. By carefully adjusting the component values and utilizing optimization algorithms, we sought to find the optimal frequency response that meets the desired specifications for our second-order low pass filter.

We assembled a substantial dataset from optimization simulations, which formed the foundation for employing machine learning techniques in the future, particularly neural networks. These advanced algorithms played a pivotal role in refining our designs and achieving optimal performance. Furthermore, we developed tools for easy access and analysis of the data, including CSV result viewing. These tools are instrumental in extracting meaningful information and guiding our design choices.

4 Current Mirror

4.1 Current Mirror Overview

A current mirror is a fundamental circuit in analog and mixed-signal design, primarily used to copy "mirror" a current from one branch of a circuit to another. It ideally ensures that the mirrored current is independent of variations in the load or supply voltage. Current mirrors are crucial for applications requiring consistent current conditions, such as biasing transistors or generating reference currents. This project will include a current mirror in its simplest form: the Simple Current Mirror. It consists of two identical transistors connected to each other at their gates and sources. This configuration forms a feedback loop that forces the output current through one transistor to replicate the input current flowing through the other.

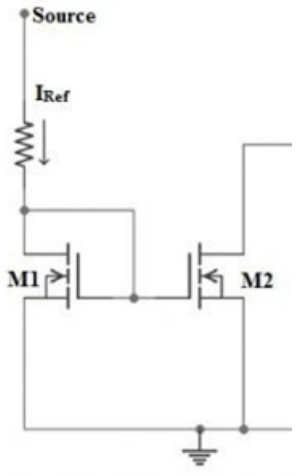


Figure 1: Diagram of a Current Mirror

4.2 Objective

This was the first circuit design task. The objective is to optimize a simple current mirror, which will be part of the differential amplifier in this project. Optimization entails matching the output current IO to the reference current IREF, by finding the optimal values of the length and width of the identical transistors.

4.3 Constraints

We needed to work within the following constraints on the length and width of the MOSFETs:

- $30nm < W < 1um$.
- $30nm < L < 1um$.

Additionally, we have the following voltage and current values:

- $V_O = 200mV$. (to replace the circuit that would normally be connected to the drain of M2)
- $V_{DD} = 800mV$.
- $I_{REF} = 5\mu A$. (arbitrary reference current)

4.4 Design and Optimization

4.4.1 Setting Up the Schematic

We set up the circuit as shown in the below figure.
We used the following components from their respective libraries:

- From library "`analoglib`": current source "idc", voltage source "vdc".
- From library "`basic`": ground "gnd".
- From the `TSMC 22nm` library: MOSFET "`nch-ulvt-dnw-mac`".

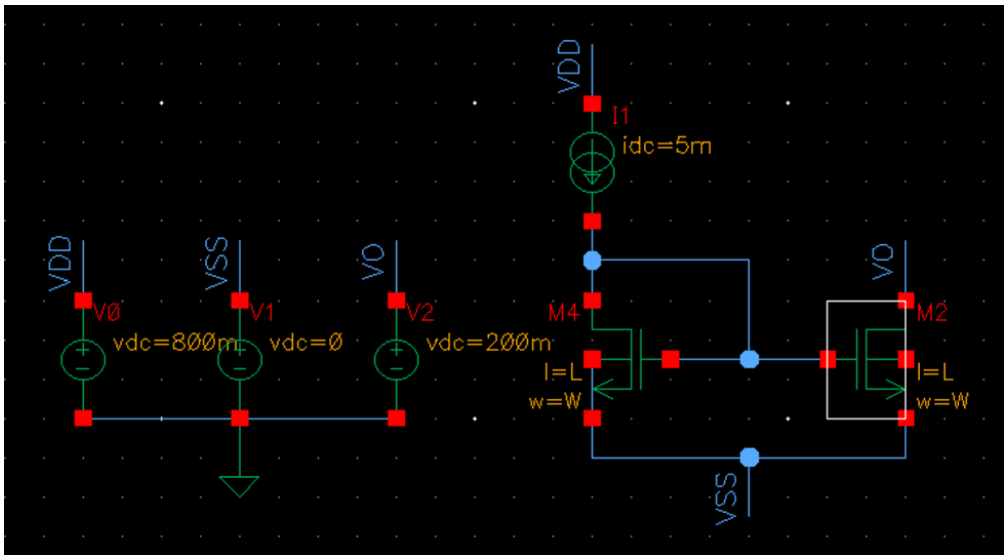


Figure 2: Current Mirror Schematic

4.4.2 Setting up the Simulation

We set up a DC analysis, enabling "Save DC Operating Point".

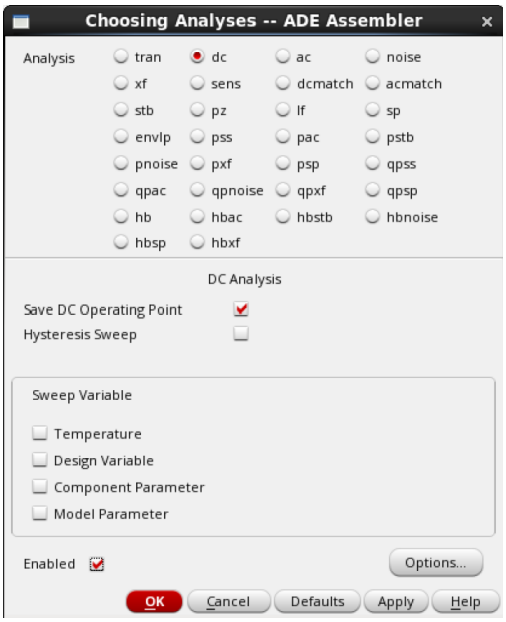


Figure 3: DC Analysis Setup

Then, we add a test that records IO and displays "pass" if its value is between 4.9 and 5.1 μA.

Test	Name	Type	Details	EvalType	Plot	Save	Spec
Filter	Filter	Filter	Filter	Filter			Filter
k:mirror:1	DC_IO	expr	IDC("/M2/D")	point	☑	☐	range 4.9u 5.1u

Figure 4: Test Setup

4.4.3 Simulating and Optimizing

We want to optimize the current mirror by varying the values of the length and width of the MOSFETs. To that end, in the Properties section of the MOSFETs, we set their length and width to "L" and "W", linking them to their respective global variables. Then, in the ADE Assembler, we run sweeps on the two global variables by setting their values in the Data View tab to the desired "start:step:stop", narrowing down the ranges until the test returns a "pass".

Figure 5: Global Variables setup in the Data View tab

4.4.4 Results

After running several sweeps, we narrowed down the values of the length and width to:

- W = 350nm.
- L = 450nm.

With these values, the output current I_O matches the reference current I_{REF}.

5 Low-Pass Filter

In designing a Low-Pass Filter (LPF), we will utilize a second-order multiple feedback filter. This type of filter employs multiple feedback loops within an amplifier circuit to achieve a second-order frequency

response. By doing so, it attenuates higher frequencies beyond a certain cutoff point while allowing lower frequencies to pass through.

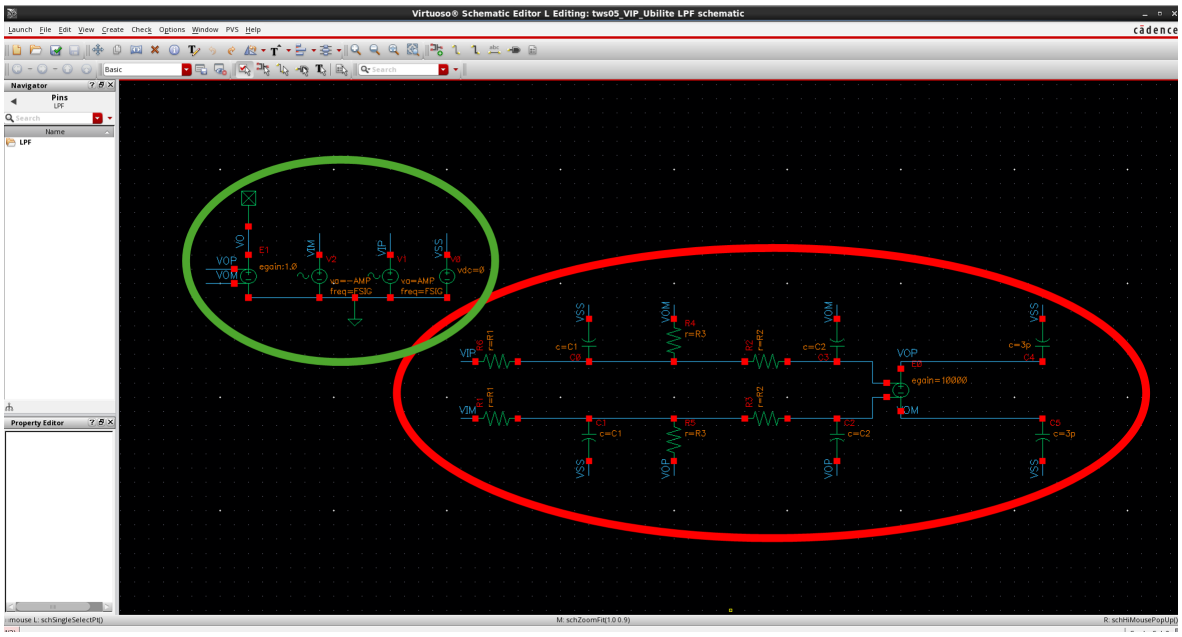


Figure 6: Caption

The characteristics of the filter are determined by the arrangement of resistors and capacitors in the network. In designing the first part of the LPF (part circled in red), we will use a total of six capacitors and six resistors along with a Voltage-Controlled Voltage Source (VCVS) as an amplifier.

Components Category	Library Name	Cell Name	Value
Resistor (R0)	Analoglib	res	R1
Resistor (R1)	Analoglib	res	R1
Resistor (R2)	Analoglib	res	R2
Resistor (R3)	Analoglib	res	R2
Resistor (R4)	Analoglib	res	R3
Resistor (R5)	Analoglib	res	R3
Capacitor (C0)	Analoglib	cap	C1
Capacitor (C1)	Analoglib	cap	C1
Capacitor (C2)	Analoglib	cap	C2
Capacitor (C3)	Analoglib	cap	C2
Capacitor (C4)	Analoglib	cap	3p
Capacitor (C5)	Analoglib	cap	3p
Voltage-Controlled Voltage Source	Analoglib	vcvs	10,000

Table 1: Component Details

After opening the previously set-up cell-view, we can begin inserting the required components to build the schematic of our LPF. To insert components, press “I” and then choose the selected library. After selecting a library, you can search for your desired component in the cell tab. Simply type the cell name of the component of your choice. After that, you can edit the properties of your chosen component and then click apply and enter. You can now place it wherever you want in the schematic. The components required to build the first half of the design (part circled in red) are summarized in the above table.

After that, we can name the wires. To do so, we press “L”, type the desired name, and then click enter and place it on the wire we want to name. Naming wires enables us to set up simulations accurately, ensuring that the behavior of the LPF is correctly represented in our design.

This was the information needed to build the first half of the LPF (part circled in red). Your circuit should look the first half of the above figure. To build the second half (part circled in green), we need various voltage sources summarized below.

Components Category	Library Name	Cell Name	CDF Parameter: Value
DC Voltage	Analoglib	vdc	DC Voltage: 0
Sinusoidal Voltage Source	Analoglib	vsin	Number of noise/freq pairs: 0
			AC Magnitude: 0.5
			AC Phase: 0
			Amplitude: AMP
			Frequency: FSIG
Sinusoidal Voltage Source	Analoglib	vsin	Number of noise/freq pairs: 0
			AC Magnitude: -0.5
			AC Phase: 0
			Amplitude: -AMP
			Frequency: FSIG
Voltage-Controlled Voltage Source	Analoglib	vcvs	Voltage Gain: 1

Table 2: Component Details with CDF Parameters

We also need to add a ground and a noconnection (nocoon). Ground is utilized in electronic circuits to provide a reference voltage level and establish a common point for electrical connections, while a no-connection (noconn) serves to indicate intentional lack of connection for specific nodes, aiding in design organization.

Components Category	Library Name	Cell Name	CDF Parameter: Value
Ground	Basic	gnd	-
No-connection	Analoglib	nocoon	-

Table 3: Component Details

At this stage, your circuit should look approximately identical to the one above. Note that for the sake of this guide, we are developing a low-pass filter (LPF) intended for integration into a larger project. Hence, it is imperative to adhere strictly to the specified requirements outlined below. These requirements are tailored specifically for this project, and it's important to recognize that LPF designs for other applications may have different criteria. **Requirements (provided by Ubilite):**

- **Closed Loop Gain:** 1
- **Quality Factor (Q):** 1.1
- **Cutoff Frequency (Fn):** 10.8 MHz
- **Minimum Signal-to-Noise Ratio (SNR):** 32 dB
- **Load Capacitance:** 3 pF
- **Rejection at 37 MHz:** 22 dB

5.1 Simulation and Optimization

5.1.1 Setting Global Variables

Global variables in Cadence® ensure consistency and enable easy modification of parameters across different design instances. In LPF design, we'll utilize global variables. We'll begin with the following values and adjust them as we go:

Variable Name	Variable Value
R1	1k
R2	1k
R3	1k
C1	1p
C2	1p
AMP	5m
FSIG	1M

Table 4: Global Variables

To add global variables, navigate to the "Global Variable" tab on the left of the screen and click "click to add variable." Enter the name and value according to the table above.

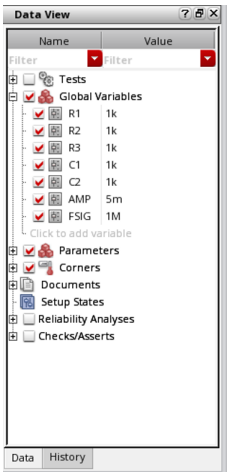


Figure 7: Adding Global Variables

5.1.2 Setting AC and Noise Tests

To set up AC analysis, begin by navigating to the "Test" tab on the left and clicking "click to add test." Select your current cell-view and press enter. Under the "Analysis" tab on the left, click "add analysis." Then, configure the analysis parameters according to the following:

- **Analysis:** AC
- **Sweep Variable:** Frequency
- **Sweep Range:** Start-Stop (Start: 10k, Stop: 1G)
- **Sweep Type:** Logarithmic (Points Per Decade: 100)

The test can be renamed by double-clicking on the top tab (the one with the upward-pointing arrow). In this simulation, We named it `"LPF_AC_tws05"`. To set up the noise/transient analysis, we repeat the process of adding a new test and configure the analysis parameters as follows:

For Noise Analysis:

- **Analysis:** Noise
- **Sweep Variable:** Frequency
- **Sweep Range:** Start-Stop (Start: 10k, Stop: 1G)
- **Sweep Type:** Logarithmic (Points Per Decade: 20)
- **Output Noise:** Voltage
- **Positive Output Node:** /VOP

- **Negative Output Node:** /VOM
- **Input Noise:** None

For Transient Analysis:

- **Analysis:** Trans
- **Stop Time:** 10u
- **Accuracy Defaults:** Conservative

Both Transient and Noise analyses should be included in the same analysis test. In this simulation, we will name it `"LPF_NZ_tws05"`.

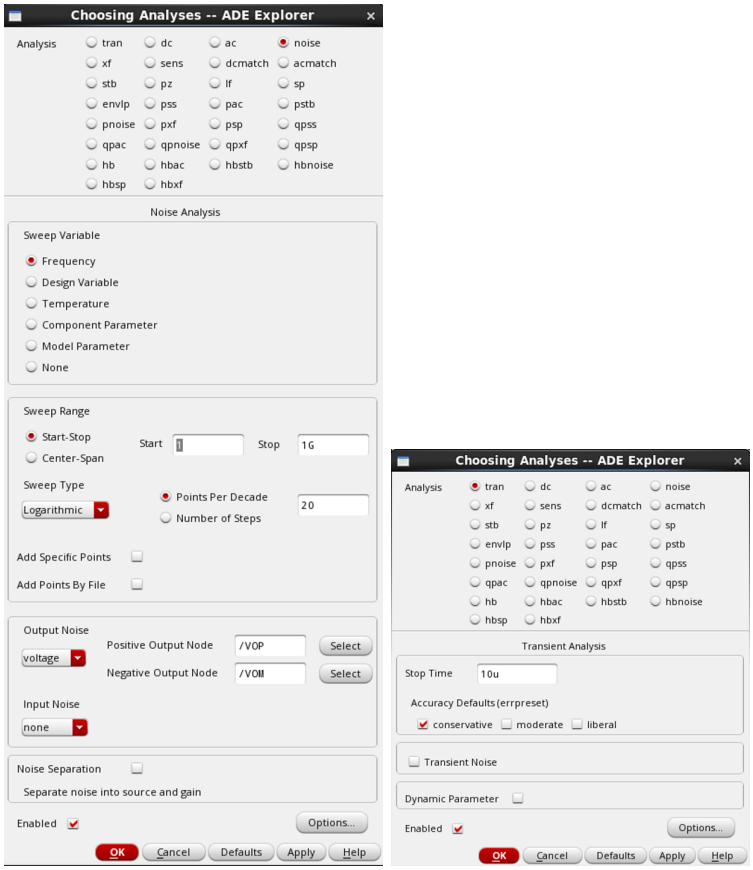


Figure 8: Choosing Analyses for Transient and Noise

5.1.3 Setting AC and Noise Outputs

In Cadence, setting outputs helps define the desired behavior and functionality of the circuits we are working on, ensuring that we are conducting accurate simulations. To set up AC and noise outputs, start by adding a new output through right-clicking and selecting "Add Expression." Choose the appropriate test for the output, specify the desired name, type, details, specifications, and other relevant parameters. For both AC and noise analyses in our LPF, we will use the following output settings:

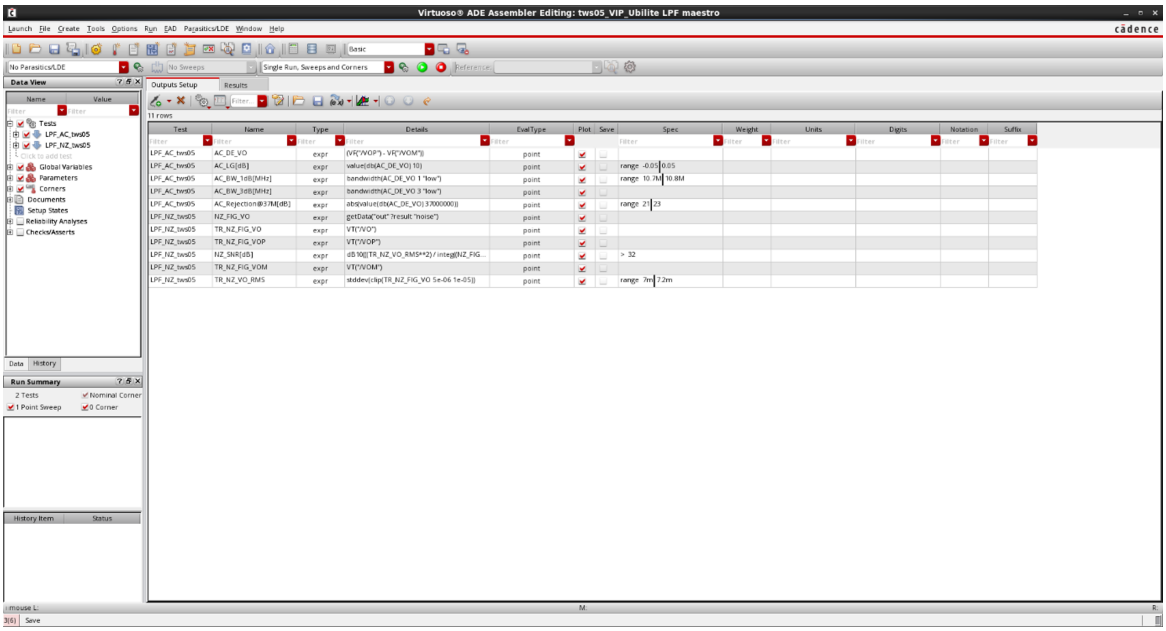


Figure 9: Setting Outputs

5.1.4 Running the Simulation

We can initiate the simulation by pressing the green play button and await the results to appear in the result tab. Upon examination, we may observe that some tests have passed while others have failed, with certain values being close to the desired parameters. Our primary objective is to determine the minimum capacitance/resistance size necessary to achieve the required gain, cutoff frequency f_c, and quality factor (Q), while simultaneously maximizing signal-to-noise ratio (SNR) and rejection. It's essential to ensure that the values of R1, R2, and R3 do not exceed 15k Ohms, while C1 and C2 should remain under 20pF.

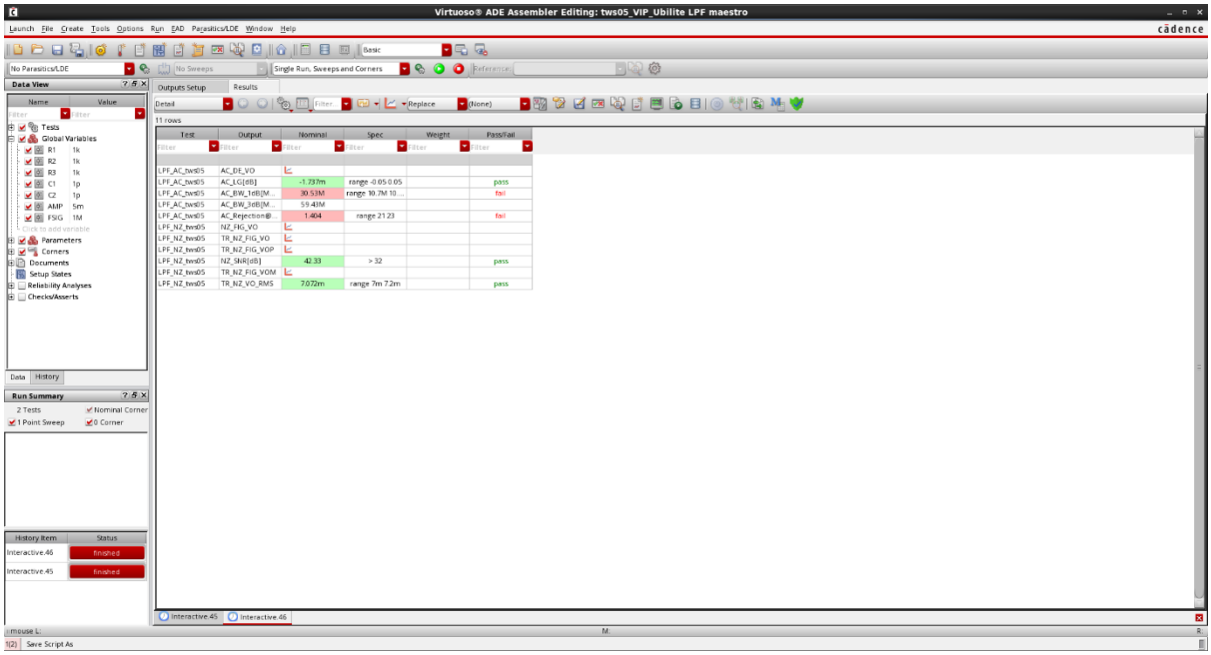


Figure 10: Simulation Results

5.1.5 Reducing Variables

To achieve the desired output, we must regulate the values. Our first step is to minimize the number of variables required for computation, which decreases the time needed to finish our LPF design and

optimization process. This is because the number of sweeps required to complete a simulation is a product of the number of sweeps per parameter. Then, by reducing the number of variables by X (arbitrary component(s)), we will be dividing the total number of sweeps by the number of sweeps of X, thus lowering the number of sweeps and time required to complete the stimulation. To reduce the variables. We can refer to the Laplace Transfer Function of the above circuit:

$$\frac{V_0}{V_i} = -\frac{1}{C_1C_2R_2R_3} \times \frac{1}{s^2 + \frac{1}{C_1R_2R_3}\left(R_3 + R_2\left(1 + \frac{R_3}{R_1}\right)\right)s + \frac{1}{R_1R_2C_1C_2}}$$

Analyzing this transfer function, we can determine the DC gain by setting $s = 0$. The DC gain is calculated as:

$$A_V\,(DC) = -\frac{R_1}{R_3}$$

Considering the specified requirement that the closed-loop gain should be 1 dB, it implies that the gain is 1 in linear scale when expressed as 0 dB. Since the dB scale is logarithmic and 0 dB corresponds to a gain of 1, we can set $A_V\,(DC)$ to 1. This leads us to the conclusion:

$$R_1 = -R_3$$

However, in an inverting amplifier setup, the output signal is phase-shifted by 180 degrees compared to the input signal, which is mathematically represented by a negative sign in the gain equation. However, in real life, resistance cannot be negative. Thus, when encountering an equation like $R_1 = -R_3$, it indicates a phase inversion rather than negative resistance values. The correct interpretation is that the magnitudes of R_1 and R_3 are equal, with both resistors having positive values. Hence:

$$R_1 = R_3$$

Then, now instead of 1k, we can replace the value of R3 with R1.

In addition, after doing several dweeps, we found that

$$R2 = 1.3 \times R1$$

acheives the most optimal result.

We can continue using the same technique to reduce the variables to the following (we use the quality factor and cutoff frequency):

$$R1 = R3$$

$$R2 = 1.3 \times R1$$

$$C_1 = 1.1 \times \frac{\sqrt{\frac{R_1}{R_2}} + 2\sqrt{\frac{R_2}{R_1}}}{67.8584 \times 10^6 \times \sqrt{\frac{R_1}{R_2}}}$$

$$C_2 = \frac{1}{1.1 \times \sqrt{\frac{R_1}{R_2}} + 2\sqrt{\frac{R_2}{R_1}} \times 67.8584 \times 10^6 \times \sqrt{\frac{R_1}{R_2}}}$$

5.1.6 Regulating Variables

To efficiently regulate variables in Cadence, we can utilize the feature to examine results for multiple variables simultaneously. Begin by selecting the three dots next to the value of the variable, such as R1, and choose "Delete Spec." Then , from the dropdown menu, select "From/To" and opt for a step type, like Linear, with a defined range, such as from 1k to 15k with a step size of 1k.

This approach generates multiple simultaneous results within the specified range, aiding in establishing a suitable parameter range.

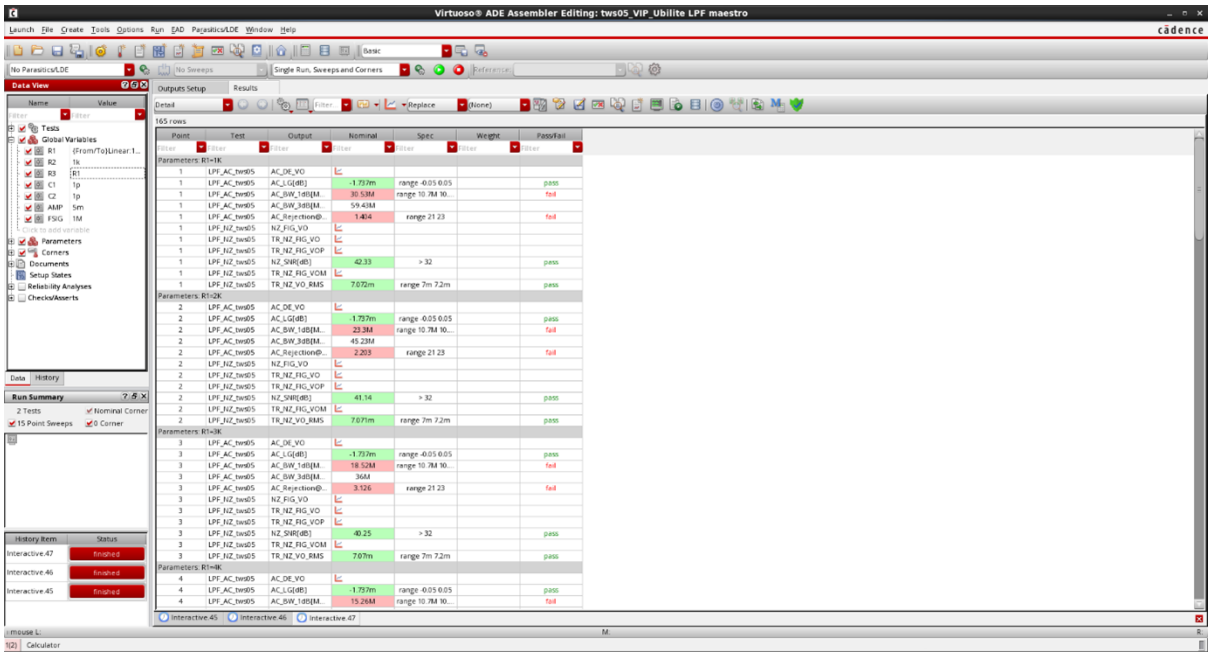


Figure 11: Variable Regulation

After identifying the range, adjust other variables, such as capacitance, to observe their impact on the results. Focusing on the mid of variables and keeping certain variables constant can save time in the analysis process. For instance, set C_1 to 10p (with 20p being the maximum; half of it) while maintaining C_2 constant and observing the outcomes. This systematic approach offers a more efficient to regulate variables compared to randomly selecting them, ultimately providing a comprehensive range of results.

5.1.7 Final Results

After conducting numerous simulations, the following regulated global variables have consistently ensured the success of all outputs:

Variable Name	Variable Value
R1	4.52k
R2	5.83k
R3	4.52k
C1	10.25p
C2	1p
AMP	5m
FSIG	1M

Table 5: Variable Names and Values

It's important to note that minor adjustments to these variables may still achieve successful outcomes. However, our primary objective is to maximize SNR (Signal-to-Noise Ratio)/rejection.

5.1.8 Problems that Might Arise and their Solutions

In Section 4 of our process, setting the AC Magnitude to both 0.5m and -0.5m was a critical step to prevent encountering an "Eval Error" during simulation. In Cadence, an "eval error" indicates a failure in evaluating or executing a specific part of the circuit or simulation setup. These errors can arise due to various issues, such as incorrect setup parameters or inconsistencies in the design.

In our specific case, the potential "Eval Error" was caused by an incorrect setup parameter, particularly in the context of AC analysis simulations. When performing AC analysis, certain equations or expressions within the circuit model may involve division by the AC magnitude. If this magnitude were to become zero, it would result in a division-by-zero error, triggering the "Eval Error."

By setting both positive (0.5m) and negative (-0.5m) AC magnitudes, we ensured that the denominator in any relevant equation did not become zero. This measure effectively prevented division-by-zero errors, thereby avoiding the occurrence of "Eval Errors" during simulation.

5.1.9 Exporting OceanScript File

OceanScript is a scripting language integrated into Cadence software, designed to automate tasks and enhance productivity in electronic circuit. This scripting language allows for the customization and automation of repetitive processes, resulting in faster development and improved design quality.

To export the OceanScript file, navigate to "File", then choose "Save Script", and save the OceanScript in the desired location.

Refer to APPENDIX II - OceanScript 1 for the exported OceanScript file.

5.2 Running the OceanScript Simulation

In Cadence, we can perform few modification to the OceanScript exported in Section 4.1.9 to optimize the LPF.

Using skill language, we replace first the value of R1 with a variable R1VALUE, using the sed command in the SKILL IDE. The modified script will be saved as an .il file and executed in Cadence Virtuoso terminal. The purpose of using the R1VALUE notation is to substitute it with different values of R each time of the iteration.

5.2.1 Create the Ocean Script

Create an Ocean script file named "`ocean4.ocn`" on your desktop. In this script, you will define the required setup and perform the value substitution for R1.

5.2.2 Modify the Ocean Script

Within the Ocean script, locate the line where R1 is defined and replace its value with R1VALUE. It should look like this: `R1 = #R1VALUE#`.

5.2.3 Implement the Value Substitution

Ocean script to perform the value substitution using the `sed` command:

```
for (i 1 3 (
    R = 500 + 500 * i
    system("sed -i 's/#R1VALUE#/" R "/g' /home/WIN2K/nbs11/Desktop/ocean4.ocn")
    load("/home/WIN2K/nbs11/Desktop/ocean4.ocn")
    load("/home/WIN2K/nbs11/Desktop/Code.il")
))
```

This code iterates three times, updating the value of R each time (R between 500 and 1500). It uses the `sed` command to replace `#R1VALUE#` with the current value of R in the "`ocean4.ocn`" file.

5.2.4 Execute the Script in Cadence Virtuoso

Open the Cadence Virtuoso console terminal and navigate to the directory where the script files are located. To find the location of the file right click on the file, go to properties, and copy the path. Run the following command to execute the script:

```
load "/home/WIN2K/nbs11/Desktop/ocean4.ocn"
```

This command executes the modified "`ocean4.ocn`" script, which includes the value substitution.

5.2.5 Create a CSV File

In your code, create an empty .CSV file on your desktop.

5.2.6 Read the CSV File

Add read csv file procedure to your code that reads the CSV file you created in step 5. Run and test the code as shown in the picture below. Load the code in the terminal of virtuoso.

We get the output in a list with their variables and values in the csv file as shown below.

By following these steps, you can modify an Ocean script, perform value substitution, execute it in Cadence Virtuoso, and incorporate additional operations, such as creating and reading a CSV file.

```
;==================== Sweeps setup ========================================
ocnxlSweepVar("R1" "#R1VALUE#")
ocnxlSweepVar("R2" "1.35*R1")
ocnxlSweepVar("R3" "R1")
ocnxlSweepVar("C1" "((1.1*((2*((R2/R1)**0.5))+((R1/R2)**0.5)))/(67858401*((R1*R2)**0.5)))")
ocnxlSweepVar("C2" "1.212/(((R1*R2*C1))*(67858401**2))")
ocnxlSweepVar("AMP" "5m")
ocnxlSweepVar("FSIG" "1M")

;==================== Model Group setup ===================================

;==================== Corners setup =======================================

;==================== Checks and Asserts setup ============================
ocnxlPutChecksAsserts(?netlist nil)

;==================== Job setup ===========================================
```

Figure 12: Simulation Results

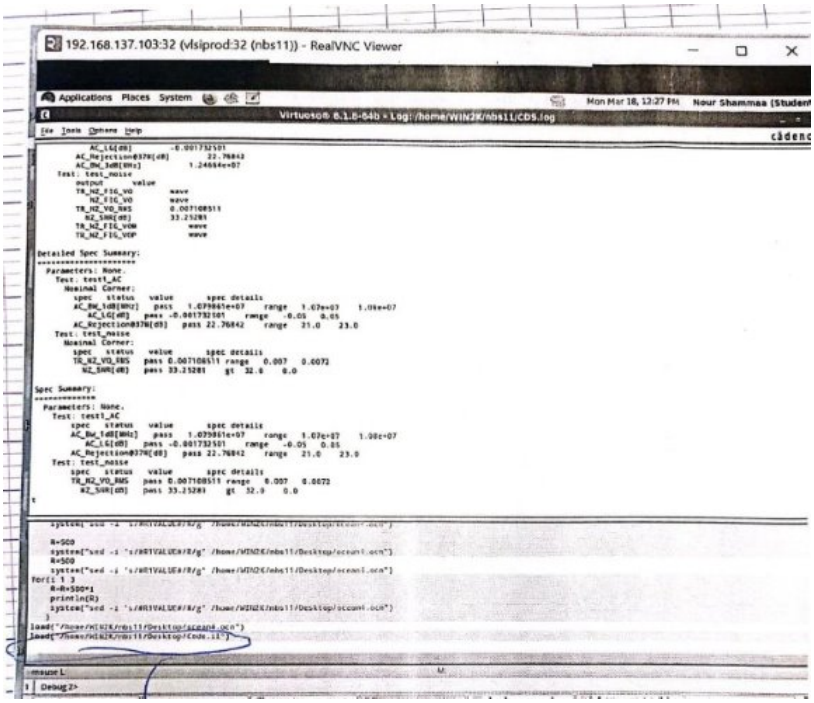


Figure 13: Simulation Log

ocean4.ocn | Code.il

```
for(i 1 3
        R=R+500*i
        println(R)
        system("sed -i 's/#R1VALUE#/R/g'/home/WIN2K/nbs11/Desktop/ocean4.ocn")
        load("/home/WIN2K/nbs11/Desktop/ocean4.ocn")
)

procedure(CCFreadCSV(filename)
        let((paramNames values inp line rows trailCR)
        trailCR=pcreCompile("\n$")
        inp=infile(filename)
        when(inp && gets(line inp)
                line=pcreReplace(trailCR line "" 1)
                paramNames=parseString(line ",")
                while(gets(line inp)
                        line=pcreReplace(trailCR line "" 1)
                        values=parseString(line ",")
                        rows=tconc(rows values)
                )
        )
        when(inp
                 close(inp)
        )
        list(paramNames car(rows))
        )
)
nour=(CCFreadCSV "/home/WIN2K/nbs11/Desktop/nourstest.csv")
println(nour)
```

Figure 14: .il file

5.3 Updated Optimized OceanScript

Refer to APPENDIX II - OceanScript 2 for the updated OceanScript File.

6 Differential Amplifiers

6.1 Introduction and Objectives

A differential amplifier is a 2-input device that amplifies the difference between these two inputs and rejects common mode signals (common signals between its inputs).

In this task, we are required to optimize the given differential amplifier to get the maximum possible gain:

$$20\log(\frac{\Delta Vo}{\Delta Vi})$$

6.2 Design Constraints

As before, it is important to abide by the following design constraints to achieve the required results.

- $V_{DD} = 800mV + / - 50mV$.
- $I_1 = 10\mu A$.
- $A_1 - V_{om} = A_1 - V_{op} = 400mV + / - 100mV$.
- $V_{DS} - V_{DSAT} > 150mV$ for M1, M2.
- $|V_{DS}| - |V_{DSAT}| > 100mV$ for M3, M4.
- $V_{DS} - V_{DSAT} > 100mV$ for M5.

The last three constraints are for the MOSFETs to be in the saturation/subthreshold region, which is necessary for the device to act as an amplifier.

6.3 Steps of Execution

6.3.1 Building the Circuit Schematic

We used real life MOSFETs from the tsmc 22nm library. The rest of the circuit components are the same as those in the previous task.

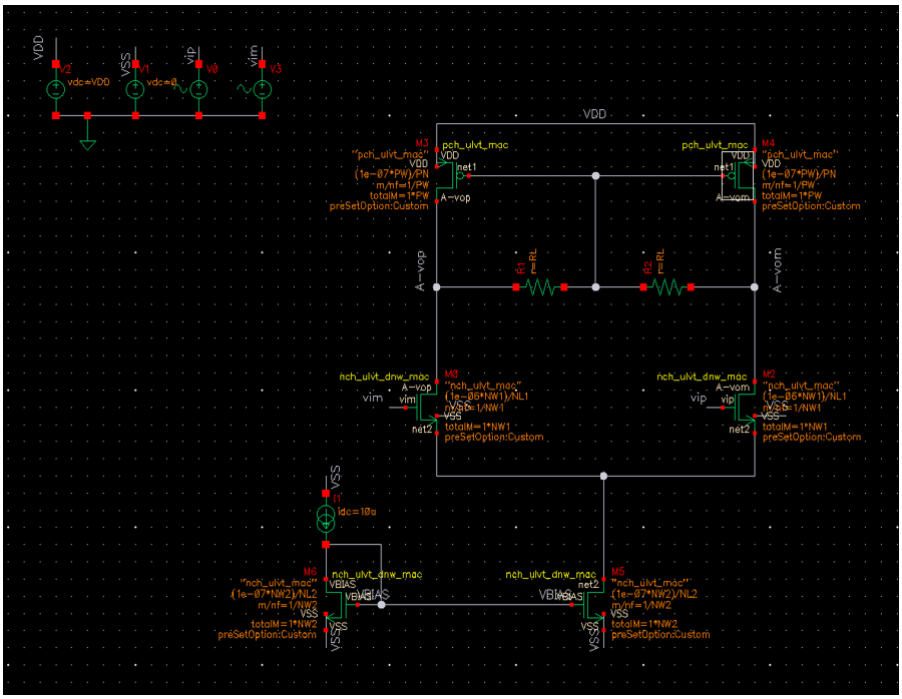


Figure 15: Circuit Schematic

6.3.2 Setting Global Variables

The variables, summarized in the table below, are the supply voltage VDD and the lengths and widths of the MOSFETs. To optimize for the best gain and pass all requirements, we must vary these variables.

Global Variable
VDD
NL1
NW1
NL2
NW2
PN
PW

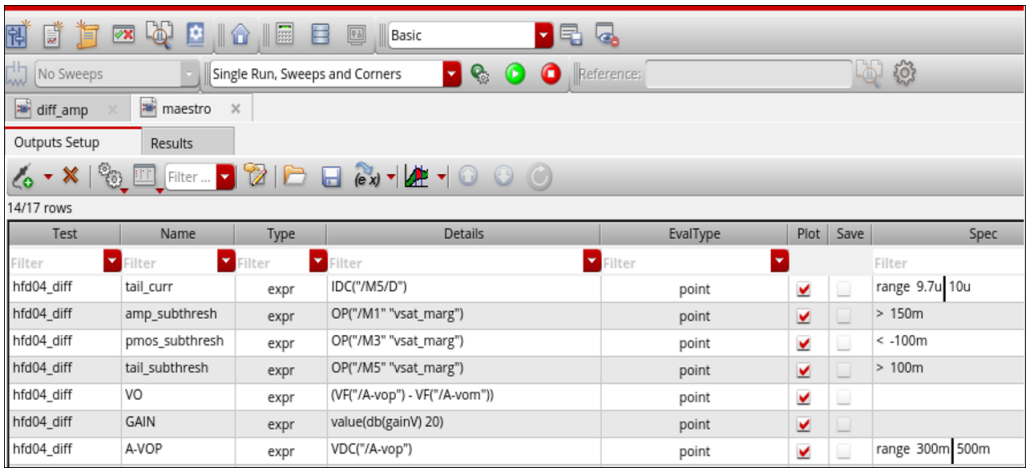
Table 6: Global Variable Names

6.3.3 Setting AC and NZ Analyses

Same as previous tasks.

6.3.4 Adding Constraints to the Output Setup

As before, we will use the following output settings with specific ranges for both AC and NZ:



Test	Name	Type	Details	EvalType	Plot	Save	Spec
Filter	Filter	Filter	Filter	Filter			Filter
hfd04_diff	tail_curr	expr	IDC("/M5/D")	point	☑	☐	range 9.7u 10u
hfd04_diff	amp_subthresh	expr	OP("/M1" "vsat_marg")	point	☑	☐	> 150m
hfd04_diff	pmos_subthresh	expr	OP("/M3" "vsat_marg")	point	☑	☐	< -100m
hfd04_diff	tail_subthresh	expr	OP("/M5" "vsat_marg")	point	☑	☐	> 100m
hfd04_diff	VO	expr	(VF("/A-vop") - VF("/A-vom"))	point	☑	☐	
hfd04_diff	GAIN	expr	value(db(gainV) 20)	point	☑	☐	
hfd04_diff	A-VOP	expr	VDC("/A-vop")	point	☑	☐	range 300m 500m

Figure 16: Output Constraints

6.3.5 Running the Simulation

We can try out different combinations of variables and use sweeps. You can try sweeping all variables. However, this method takes a very long time since there are way too many variables (7 variables). You can use the equation of the gain derived from the given circuit for an ideal differential amplifier to check how to increase the Gain:

$$g_{m1} \times r_{o1} // (r_{o3} // R_L)$$

Since the gain depends on these variables, add them to the output setup.

After running the simulation again, you will notice that r_{o1} is significantly smaller than r_{o3}. Recall: the parallel equivalent will be smaller than the smallest one. Therefore, increasing r_{o3} would not increase the gain by much. On the other hand, increasing r_{o1} will significantly increase the gain.

hfd04_diff	ro1	221.6K	
hfd04_diff	ro3	1.492M	
hfd04_diff	eq2	103.4K	

Figure 17:

In order to increase r_{o1}, the output resistance of MOSFET M1, you can vary the length and width of that MOSFET. You will also need to play with the other variables in order to satisfy all required conditions. The results are in the below figure.

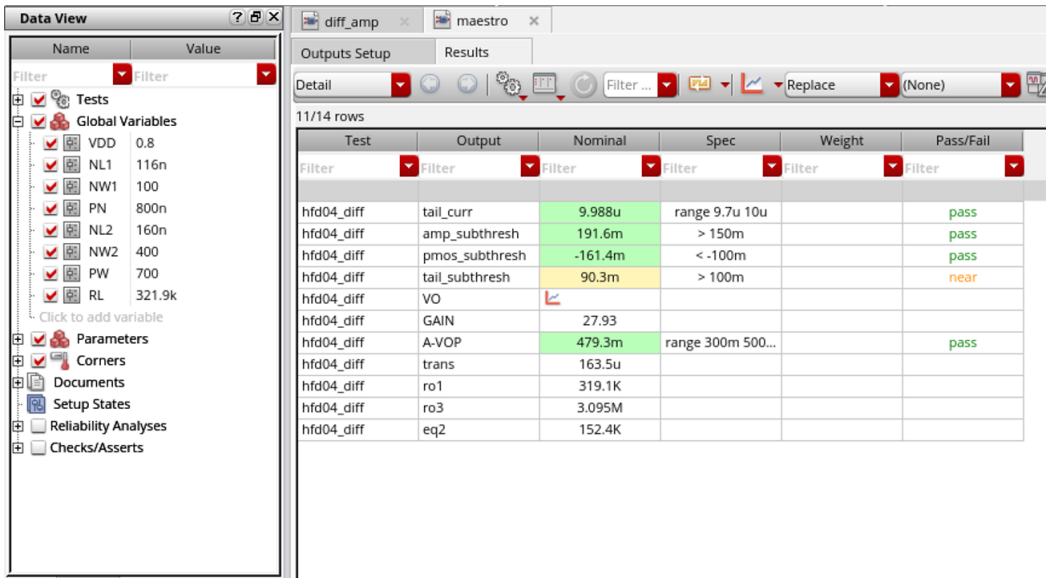


Figure 18: Final Results for M=1

6.4 For M = 4

M is the MOSFET multiplier (number of transistors connected in parallel to form a signal). It is directly related to the Width of transistor. The larger width means more current capacity, and thus more current. So $M = 4$ has more current in M5 MOSFET than $M = 1$.

We have to optimize the same amplifier but after allowing more current. We repeat the process we did before. The final result is in the figure below, where the gain is 24.53 which is acceptable.

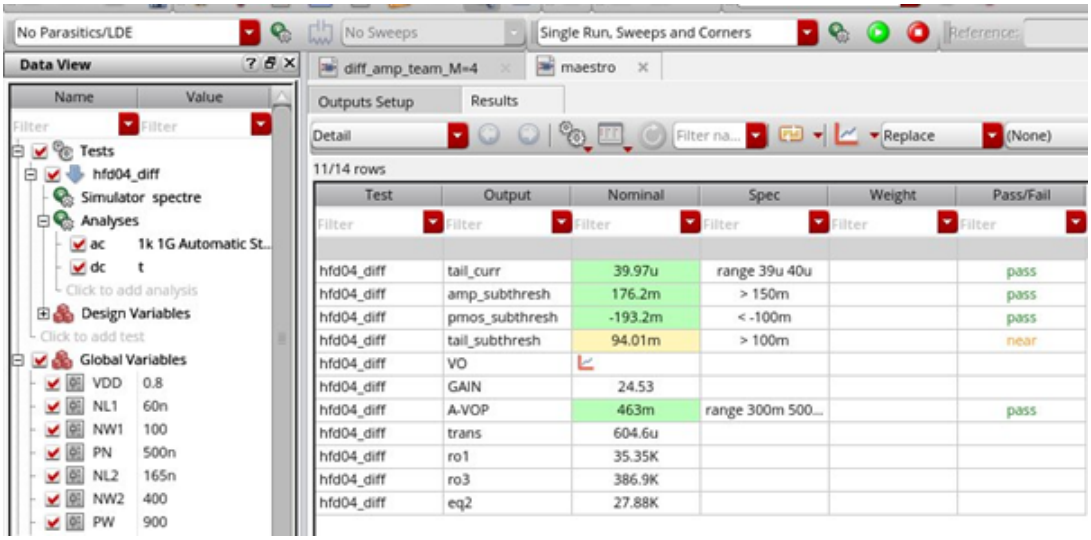


Figure 19: Final Results for M=4

As explained before regarding the oceanscript and csv file saving, here is the output after doing the same steps.

Figure 20: CSV File Output

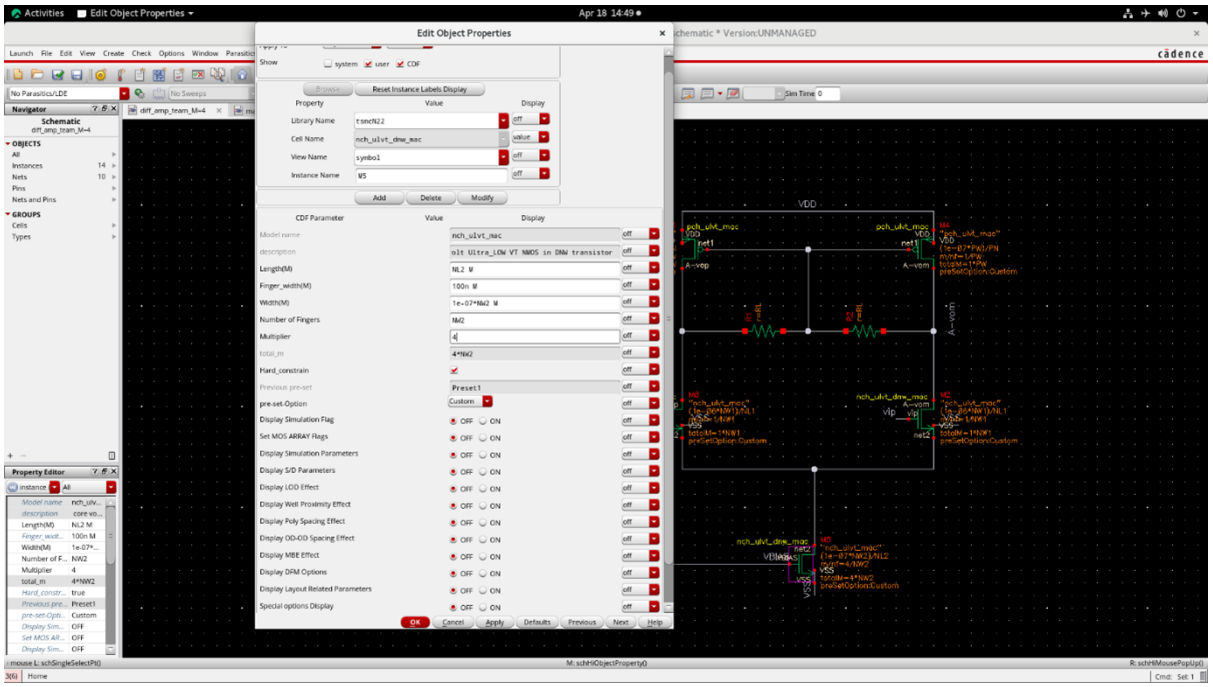


Figure 21: Changing to M=4

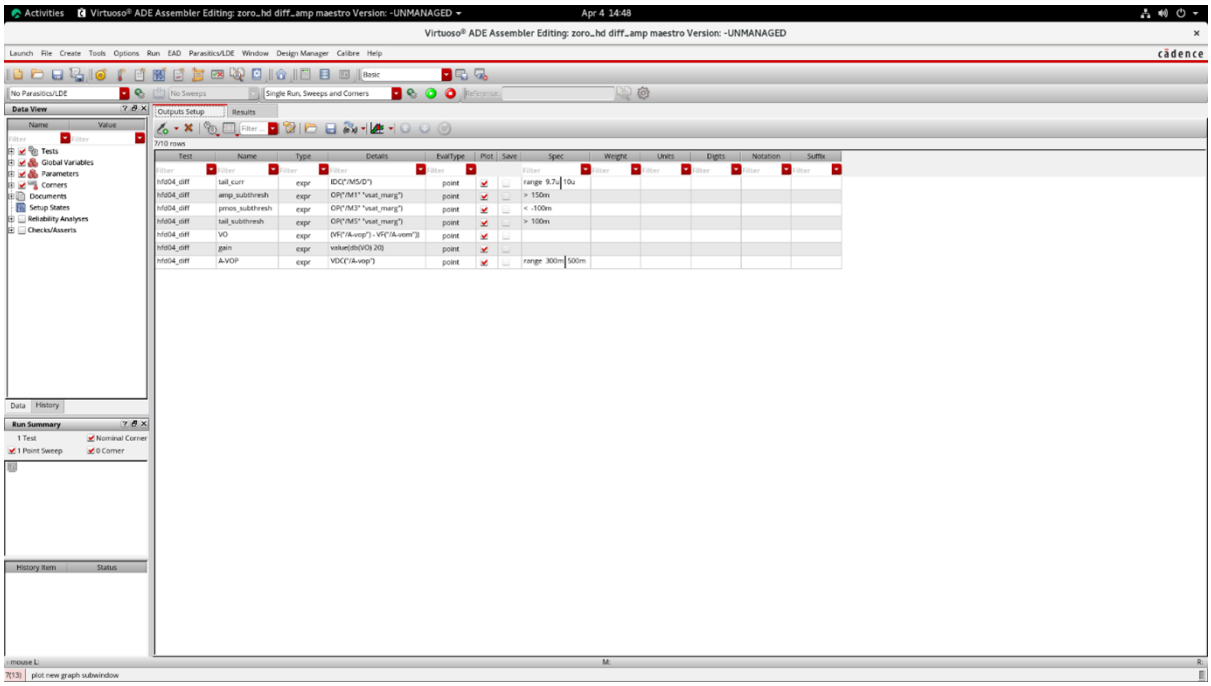


Figure 22: Ouput for M=4

7 Iterative Optimization Algorithms

7.1 Newton's method in optimization

The central problem of optimization is minimization of functions. Let us first consider the case of univariate functions, i.e., functions of a single real variable. We will later consider the more general and more practically useful multivariate case.

Given a twice differentiable function $f : R \to R$, we seek to solve the optimization problem

$$\min_{x \in R} f(x).$$

Newton's method attempts to solve this problem by constructing a sequence $\{x_k\}$ from an initial guess (starting point) $x_0 \in R$ that converges towards a minimizer x^* of f by using a sequence of second-order Taylor approximations of f around the iterates. The second-order Taylor expansion of f around x_k is

$$f(x_k + t) \approx f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2.$$

The next iterate x_{k+1} is defined so as to minimize this quadratic approximation in t, and setting $x_{k+1} = x_k + t$. If the second derivative is positive, the quadratic approximation is a convex function of t, and its minimum can be found by setting the derivative to zero.

7.2 Gradient Descent

Gradient descent is based on the observation that if the multi-variable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point $\mathbf{a}$, then $F(\mathbf{x})$ decreases fastest if one goes from $\mathbf{a}$ in the direction of the negative gradient of F at $\mathbf{a}$, $-\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

for a small enough step size or learning rate $\gamma \in R_+$, then $F(\mathbf{a}_n) \geq F(\mathbf{a}_{n+1})$. In other words, the term $\gamma \nabla F(\mathbf{a}_n)$ is subtracted from $\mathbf{a}_n$ because we want to move against the gradient, toward the local minimum. With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$

We have a monotonic sequence

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$

so, hopefully, the sequence $(\mathbf{x}_n)$ converges to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration.

It is possible to guarantee the convergence to a local minimum under certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ. Those include the sequence

$$\gamma_n = \frac{\left|(\mathbf{x}_n - \mathbf{x}_{n-1})^T [\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})]\right|}{\|\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})\|^2}$$

as in the Barzilai-Borwein method, or a sequence γ_n satisfying the Wolfe conditions (which can be found by using line search). When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

7.3 Gradient Descent vs Newton's method in optimization

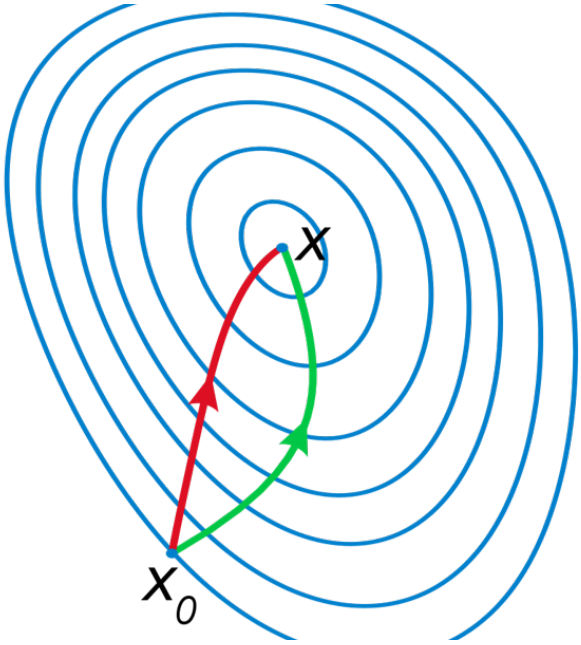


Figure 23: A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information (i.e. the second derivative) to take a more direct route.

7.4 A General Methodology in Circuit Optimization

$$
\begin{aligned}
\vec{x}_{n+1} &= \vec{x}_n - \gamma_n \nabla F(\vec{x}_n) \\
\vec{x}_n &= (R_1, R_2, R_3, C_1, C_2)_n \quad \text{iteration } n \text{ of the optimization loop} \\
(R_1)_{n+1} &= (R_1)_n - \gamma_n \left.\frac{\partial F}{\partial R_1}\right|_n \Rightarrow \text{ set } |(R_1)_{n+1} - (R_1)_n| = \Delta R \\
&\quad \text{find } \gamma_n' \\
(R_2)_{n+1} &= (R_2)_n - \gamma_n \left.\frac{\partial F}{\partial R_2}\right|_n \Rightarrow \text{ set } |(R_2)_{n+1} - (R_2)_n| = \Delta R \\
&\quad \text{find } \gamma_n'' \\
&\quad \vdots \\
(C_2)_{n+1} &= (C_2)_n - \gamma_n \left.\frac{\partial F}{\partial C_2}\right|_n \Rightarrow \text{ set } |(C_2)_{n+1} - (C_2)_n| = \Delta C \\
&\quad \text{find } \gamma_n''' \\
\gamma_n &= \min(\gamma_n \to \gamma_n', \gamma_n \to \gamma_n'', \gamma_n \to \gamma_n''', \gamma_n \to \gamma_n'''')
\end{aligned}
$$

7.5 Pseudo code and Script Analysis

7.5.1 Algorithms

Algorithm 1 Gradient Calculation

```
1:  procedure GradFunction(InputVars)
2:      resultgrad ← [ ]
3:      for each var in InputVars do
4:          neededvar ← variable name
5:          setval ← original value + Δ
6:          Set variable to setval
7:          Load .ocn script
8:          Read CSV output to get fc1
9:          setval ← original value - Δ
10:         Set variable to setval
11:         Load gradient calculation script
12:         Read CSV output to get fc2
13:         partial ← (fc1−fc2)/(2×Δ)
14:         Append ∂F/∂x_i to resultgrad
15:         setval ← variable value
16:     end for
17:     return resultgrad
18: end procedure
```

Algorithm 2 Minimum Absolute Value Function

```
1:  function minimumAbsoluteValue(lst)
2:      minAbs ← absoluteValue of first element of lst
3:      minNumb ← first element of lst
4:      for i from 1 to length of lst − 1 do
5:          nextToCheck ← |lst[i]|
6:          if nextToCheck < minAbs then
7:              minAbs ← nextToCheck
8:              minNumb ← ith element of lst
9:          end if
10:     end for
11:     return minNumb
12: end function
```

Algorithm 3 Optimization Algorithm

procedure OPTIMIZATIONALGO(InputVars, target)
    Load .ocn script
    get initial fc from .csv File
    $\epsilon \leftarrow 1$
    while $|\text{center\_freq} - \text{target}| \geq \epsilon$ **do**
        gamma_list $\leftarrow []$
        Gradf $\leftarrow$ GradFunction(InputVars)
        for each $\partial F/\partial x_i$ **in** Gradf **do**
            if $(center\_freq >= target)$ **then**
                gammanow $\leftarrow -\frac{\Delta x_i}{\partial F/\partial x_i}$
            else
                gammanow $\leftarrow \frac{\Delta x_i}{\partial F/\partial x_i}$
            end if
            Append gammanow to gamma_list
        end for
        gammachosen $\leftarrow$ minimumabsolutevalue(gamma_list)
        for each var **in** InputVars **do**
            setval $\leftarrow$ variable value $-$ gammachosen $\times \partial F/\partial x_i$
            needed Variable $\leftarrow$ setval
            Update variable in InputVars
        end for
        Load updated script
        Calculate new center frequency
        Print results (Gamma, Gradf, InputVars) at current iteration for debugging
    end while
    return InputVars
end procedure

7.6 Mathematical Analysis of the Optimization Algorithm

The optimization algorithm aims to iteratively adjust a set of input variables to minimize the discrepancy between the current center frequency and a desired target frequency. Let $F(\mathbf{x})$ represent the objective function, where $\mathbf{x}$ denotes the vector of input variables.

7.6.1 Objective Function and Optimization Problem Formulation

The optimization problem is formulated as:

$$\begin{aligned} \text{Minimize: } & F(\mathbf{x}) \\ \text{Subject to: } & |\text{center\_freq} - \text{target}| \geq \epsilon \end{aligned}$$

Here, ϵ represents a specified threshold for convergence.

7.6.2 Convergence Analysis

Convergence of the algorithm is contingent upon suitable conditions. Under conditions such as convexity of $F(\mathbf{x})$ and appropriate step size selection, convergence to a local minimum can be ensured.

7.6.3 Gradient Calculation

The gradient of $F(\mathbf{x})$ with respect to each input variable is estimated using finite differences:

$$\frac{\partial F}{\partial x_i} \approx \frac{F(\mathbf{x} + \Delta \mathbf{x}_i) - F(\mathbf{x} - \Delta \mathbf{x}_i)}{2\Delta x_i}$$

where $\Delta \mathbf{x}_i$ represents a small perturbation in the i-th variable.

7.6.4 Step Size Selection

The step size, denoted as γ, is determined for each gradient direction:

$$\gamma_i = -\frac{\Delta x_i}{\partial F / \partial x_i}$$

The smallest step size, γ_{chosen}, is selected to update the variables.

7.6.5 Variable Update Rule

The variables are updated using:

$$x_i^{(n+1)} = x_i^{(n)} - \gamma_{\text{chosen}} \cdot \frac{\partial F}{\partial x_i}$$

where t represents the iteration index.

7.6.6 Termination Criteria

The optimization process continues until the absolute difference between the current center frequency and the target frequency falls below the threshold ϵ.

8 Using Skill

8.1 Launch

Open Virtuoso, then head to the tools section and open SKILL IDE:

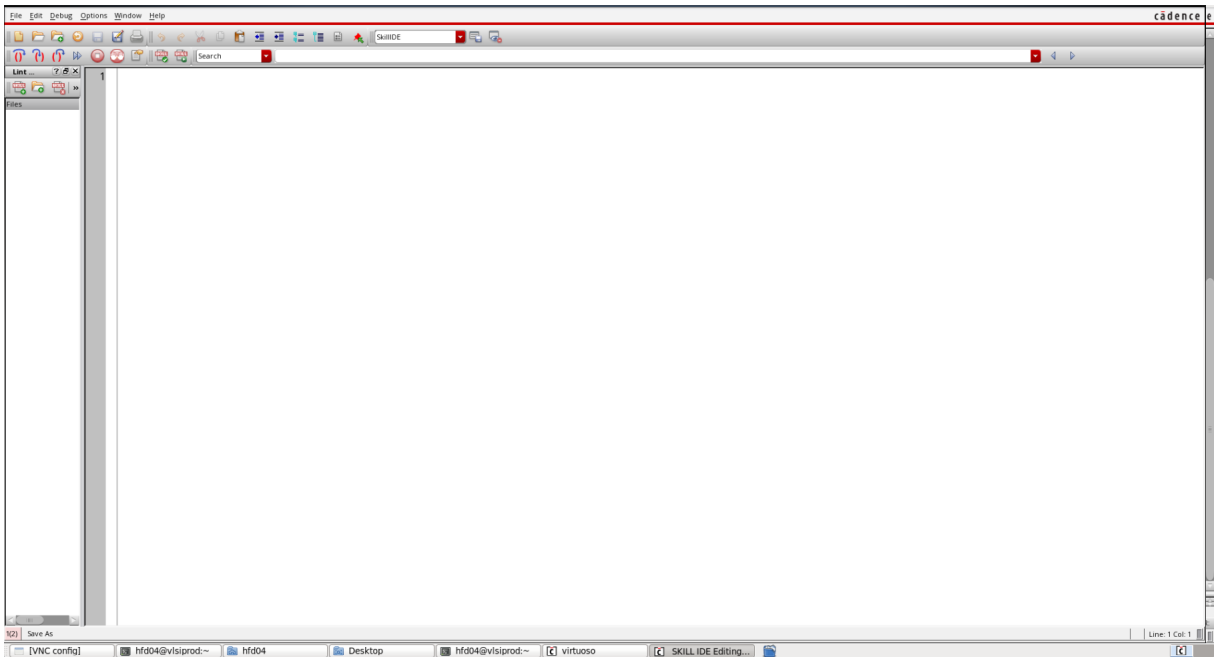


Figure 24: SKILL IDE after opening

Make sure the "SkillIDE" compile option is chosen and not "Debugging".

8.2 SKILL Functions:

We created a `.il` file and include the `readCSV` procedure Prof. Lakkis provided, and we used the reference card to build the Gradient Function:

8.2.1 Gradient Function:

```
(defun GradFunction (InputVars)
   (setq resultgrad '())
  for(i 0 (length(InputVars)-1)
        neededvar=stringToSymbol((nth 2 (nth i InputVars)) )
 setval= (nth 1 (nth i InputVars))+ (car (nth i InputVars))
 set(neededvar setval)
             (load "/home/WIN2K/hfd04/Desktop/GradientFinalOcn.ocn")
             (setq CSVOut (CCFreadCSV "/home/WIN2K/hfd04/Desktop/outputFinalGrad.csv"))
             (setq res1 (nth 1 (nth 0 (nth 1 CSVOut))))
 println(CSVOut)
         setval= -2*(nth 1 (nth i InputVars))+ (car (nth i InputVars))
 set(neededvar setval)
             (load "/home/WIN2K/hfd04/Desktop/GradientFinalOcn.ocn")
             (setq CSVOut (CCFreadCSV "/home/WIN2K/hfd04/Desktop/outputFinalGrad.csv"))
             (setq res2 (nth 1 (nth 0 (nth 1 CSVOut))))
 res1=atof(res1)
 res2=atof(res2)
             partial= (res1-res2)/(2*(nth 1 (nth i InputVars)))
             (push partial resultgrad)
 setval= (nth 1 (nth i InputVars))+ (car (nth i InputVars))
 set(neededvar setval)
   )
 resultgrad = (reverse resultgrad)
             resultgrad
)
```

The algorithm follows the pseudocode in which it takes n input Variables as a list of lists, and computes the partial derivative of the center frequency over each function:

$$\frac{\partial f_c}{\partial x_i} = \frac{f_c(x_i + \Delta x_i) - f_c(x_i - \Delta x_i)}{2\Delta x_i}$$

This is done by utilizing the methods used previously in the report in which the `.ocn` script is edited to save the results in a `.csv` file, and then the `readCSV` function reads file and spits the output

8.2.2 readCSV Function:

```
procedure(CCFreadCSV(filename)
    let((paramNames values inp line rows trailCR)
        ;---------------------------------------------------------------------
        ; pattern for trailing carriage return
        ;---------------------------------------------------------------------
        trailCR=pcreCompile("\n$")
        inp=infile(filename)
        ;---------------------------------------------------------------------
        ; First read the parameter names, then the rows of values.
        ; Uses tconc to ensure lines read in order
        ;---------------------------------------------------------------------
        when(inp && gets(line inp)
            line=pcreReplace(trailCR line "" 1)
            paramNames=parseString(line ",")
            while(gets(line inp)
                line=pcreReplace(trailCR line "" 1)
                values=parseString(line ",")
                rows=tconc(rows values)
            )
        )
        when(inp
```

```
            close(inp)
        )
        list(paramNames car(rows))
    )
)
```

As we can see, the `readCSV` returns a list in which contains the result of the virtuoso simulation saved in the `.csv` file the output of the .ocn file

The Gradient function uses the Partial derivative approximation by first running the `.ocn` file on a chosen variable, getting f_c for $x_i + \Delta x_i$, then for $x_i - \Delta x_i$, thus computing the partial derivative of x_i, after that, it returns x_i to its initial value and does the for x_{i+1}.

8.2.3 MinAbsVal Function:

```
(defun minimumabsolutevalue (lst)

minabs=(abs (nth 0 lst))
minnumb=(nth 0 lst)
    (for i 1 length(lst)-1
nexttocheck=(abs (nth i lst))
if((nexttocheck < minabs) then
minabs=nexttocheck
minnumb=(nth i lst)
)
)
minnumb
)
```

This code implements the algorithm explained before to find γ_i, the learning rate.

8.2.4 Optimization Algorithm Function:

```
(defun OptimizationAlgo (InputVars target)
    (load "/home/WIN2K/hfd04/Desktop/GradientFinalOcn.ocn")
    (setq CSVOut (CCFreadCSV "/home/WIN2K/hfd04/Desktop/outputFinalGrad.csv"))
    (setq center_freq (nth 1 (nth 0 (nth 1 CSVOut))))
center_freq=atof(center_freq)
epsilon = 1
while( (abs (center_freq - target) >= epsilon)
    (setq gamma_list '())
Gradf=(GradFunction InputVars)
for(i 0 (length(Gradf)-1)
if(( (nth i Gradf) != 0) then
                    if (((center_freq) >= target) then
gammanow = (-(nth 1 (nth i InputVars)))/(nth i Gradf)
                    ) if (((center_freq) < target) then
                    gammanow = ((nth 1 (nth i InputVars)))/(nth i Gradf)
                    )
(push gammanow gamma_list)

)
)
gammachosen=minimumabsolutevalue(gamma_list)
for(i 0 (length(Gradf)-1)
neededvar=stringToSymbol((nth 2 (nth i InputVars)) )
setval=(nth 0 (nth i InputVars))-gammachosen*(nth i Gradf)
setf( (nth 0 (nth i InputVars)) setval)
set(neededvar setval)
)
(load "/home/WIN2K/hfd04/Desktop/GradientFinalOcn.ocn")
```

```
            (setq CSVOut (CCFreadCSV "/home/WIN2K/hfd04/Desktop/outputFinalGrad.csv"))
            (setq center_freq (nth 1 (nth 0 (nth 1 CSVOut))))
center_freq=atof(center_freq)
println("Results till now: ")
println("Gamma: ")
println(gamma_list)
println("Gradf: ")
println(Gradf)
println("InputVars: ")
println(InputVars)
)
InputVars
)
```

This is the optimization function that follows the algorithm explained in 6.4, and it utilizes what we learned on reading and loading csv and oceanscript files, and utilizing the `readCSV` function we wrote.

8.3 Summary of SKILL Syntax:

- `defunc` is used to define a function.
- `atof()` changes the variable from string type to integer type.
- `nth i ()` represents a for loop.
- `car` returns the first element of the list.
- `gets` reads a line from the input and stores it as a string in the variable.
- `tconc` is a data structure that facilitates efficient addition of elements to the end of a list.

9 Testing and Results

Initial Values:

$R_1 = 3000\Omega$, $R_2 = 2000\Omega$, $R_3 = 3000\Omega$, $C_1 = 10pF$, $C_2 = 2pF$.

In addition, the step sizes are $\Delta R_i = 500\Omega$ and $\Delta C_i = 50fF$

Results:

$R_1 = 3000\Omega$, $R_2 = 2000\Omega$, $R_3 = 3000\Omega$, $C_1 = 10.000325pF$, $C_2 = 2.2176pF$.

We notice that the final values are close to the obtained values.

10 Future Work

To enhance the robustness and efficiency of our optimization methods, further research is recommended in several areas. Firstly, the implementation of Accelerated Proximal Gradient Descent (APGD), also known as Fast Iterative Shrinkage-Thresholding Algorithm (FISTA), promises significant improvements in the speed of convergence over traditional gradient methods. This approach, noted for its efficacy in handling sparse recovery problems, will be explored to determine its applicability and performance in our context. Additionally, incorporating Line Search methods will be investigated to determine the most effective step size dynamically, ensuring faster convergence rates without sacrificing stability.

Moreover, as we accumulate more data from our ongoing experiments, it will be feasible to integrate Machine Learning (ML) techniques to predict and optimize outcomes. The proposed steps to build our ML model include defining the problem, building the dataset, training the model, evaluating its performance, and ultimately deploying the model for real-world applications. This integration aims to

leverage predictive analytics to fine-tune our design parameters further and achieve optimal configurations automatically.

Exploring these advanced methods will enable us to tackle more complex problems and enhance the system's overall effectiveness and efficiency. The pursuit of these avenues will also open up collaborative opportunities with other research groups and industry partners who are working on similar challenges.

11 Conclusion

In summary, our project aims to automate and optimize the design process for SOC Wireless Transceivers. By focusing on the second-order low pass filter and the first stage operational amplifier, we strive to achieve exceptional performance within the 2.5-7GHz frequency range. Through careful analysis of optimization simulations, and employing appropriate optimization algorithms, we have made significant strides in improving the design efficiency of these components and achieving the ideal frequency responses.

12 APPENDIX I: Cadence Hotkeys

To facilitate the workflow in Cadence®, it is advised to leverage keyboard shortcuts. These shortcuts are designed to streamline various tasks within the software, saving time and effort while navigating through different functionalities. A summary of such keyboard shortcuts are represented in the tables below:

Hotkey	**Action**
Ctrl+R	Open selected view for read
Ctrl+O	Open selected view for editing

Table 7: Library Manager Hotkeys

Hotkey	**Action**
W	Add a wire
I	Add an instance
P	Add a pin
L	Label to a wire
E	Display options
Q	Open property dialogue box
{ or Shift+Z	Zoom-out by 2x
} or Ctrl+Z	Zoom-in by 2x
C	Copy
M	Move
S	Stretch
F3	Open dialogue box for move/stretch mode
F2	Save
F8	Check and save
U	Undo
Shift+U	Redo
Delete	Delete an object
Ctrl+D	Deselect all
F3	Save
F5	Open
Tab	Pan
F	Fit
Shift+X	Descend to edit by one
B	Go one level up
Shift+B	Return to top
X	Descend to read by one level
Ctrl+R	Redraw the window
Shift+V	World view
Ctrl+W	Close the window
Shift+Q	Properties of the whole cell view

Table 8: Schematic Diagram Hotkeys

Hotkey	**Action**
F	Fit
Shift+F	Show all levels as flat
Ctrl+F	Hide hierarchy, show outline
R	Rectangle
Q	Property of an object
Ctrl+Z	Zoom in
Shift+Z	Zoom out
F2	Save
T	Tap
P	Path
Ctrl+A	Select all
Ctrl+D	Deselect all
C	Copy
M	Move
S	Stretch
K	Ruler
I	Add an instance
U	Undo
Shift+U	Redo
Shift+R	Reshape
Shift+C	Chop
Shift+M	Merge all rectangles
E	Display options
F6	Redraw

Table 9: Layout Tool Hotkeys

13 APPENDIX II: OceanScript Samples

13.1 OceanScript 1:

```
;====================Set to Maestro mode assembler ============================
ocnSetXLMode("assembler")
ocnxlProjectDir( "~/simulation" )
ocnxlTargetCellView( "tws05_VIP_Ubilite" "LPF" "maestro" )
ocnxlResultsLocation( "" )
ocnxlSimResultsLocation( "" )
ocnxlMaxJobFail( 20 )

;====================== Tests setup ========================================

;---------- Test "LPF_AC_tws05" -------------
ocnxlBeginTest("LPF_AC_tws05")
simulator( 'spectre )
design( "tws05_VIP_Ubilite" "LPF" "schematic")
analysis('ac ?start "10k"  ?stop "1G"  ?dec "100"  )
desVar(   "AMP" "" )
desVar(   "C1" "" )
desVar(   "C2" "" )
desVar(   "FSIG" "" )
desVar(   "R1" "" )
desVar(   "R2" "" )
desVar(   "R3" "" )
desVar(   "V" "" )
envOption(
'analysisOrder  list("pz" "dcmatch" "stb" "tran" "envlp" "ac" "dc"
 "lf" "noise" "xf" "sp" "pss" "pac" "pstb" "pnoise" "pxf" "psp" "qpss" "qpac"
 "qpnoise" "qpxf" "qpsp" "hb" "hbac" "hbstb" "hbnoise" "hbxf" "sens" "acmatch")
)
option( 'pzSeverity  "None"
'noiseSeverity  "None"
'spSeverity  "None"
'acSeverity  "None"
'dcOpSeverity  "None"
'dcSeverity  "None"
'tranSeverity  "None"
)
temp( 27 )
ocnxlOutputExpr( "(VF(\"/VOP\") - VF(\"/VOM\"))" ?name "AC_DE_VO" ?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "bandwidth(AC_DE_VO 1 \"low\")" ?name "AC_BW_1dB[MHz]" ?plot t ?evalType 'point)
ocnxlOutputExpr( "value(db(AC_DE_VO) 10)" ?name "AC_LG[dB]" ?plot t ?evalType 'point)
ocnxlOutputExpr( "abs(value(db(AC_DE_VO) 37000000))" ?name "AC_Rejection@37M[dB]" ?plot t ?evalType
ocnxlOutputExpr( "bandwidth(AC_DE_VO 3 \"low\")" ?name "AC_BW_3dB[MHz]" ?plot t ?evalType 'point)
ocnxlEndTest() ; "LPF_AC_tws05"

;---------- Test "LPF_NZ_tws05" -------------
ocnxlBeginTest("LPF_NZ_tws05")
simulator( 'spectre )
design( "tws05_VIP_Ubilite" "LPF" "schematic")
analysis('noise ?start "1"  ?stop "1G"  ?dec "20"
?p "/VOP"  ?n "/VOM"  ?oprobe ""  ?iprobe ""  )
analysis('tran ?stop "10u"  ?errpreset "conservative"  )
desVar(   "AMP" "" )
desVar(   "C1" "" )
desVar(   "C2" "" )
desVar(   "FSIG" "" )
```

```
desVar(   "R1" "" )
desVar(   "R2" "" )
desVar(   "R3" "" )
desVar(   "V" "" )
envOption(
'analysisOrder  list("pz" "dcmatch" "stb" "tran" "envlp"
 "ac" "dc" "lf" "noise" "xf" "sp" "pss" "pac" "pstb" "pnoise" "pxf"
 "psp" "qpss" "qpac" "qpnoise" "qpxf" "qpsp" "hb" "hbac" "hbstb"
 "hbnoise" "hbxf" "sens" "acmatch")
)
option( 'pzSeverity  "None"
'noiseSeverity  "None"
'spSeverity  "None"
'acSeverity  "None"
'dcOpSeverity  "None"
'dcSeverity  "None"
'tranSeverity  "None"
)
temp( 27 )
ocnxlOutputExpr( "VT(\"/VO\")" ?name "TR_NZ_FIG_VO" ?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "getData(\"out\" ?result \"noise\")" ?name "NZ_FIG_VO"
?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "stddev(clip(TR_NZ_FIG_VO 5e-06 1e-05))"
?name "TR_NZ_VO_RMS" ?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "dB10(((TR_NZ_VO_RMS**2) / integ((NZ_FIG_VO**2) 10000 15000000 \" \")))"
?name "NZ_SNR[dB]" ?plot t ?evalType 'point)
ocnxlOutputExpr( "VT(\"/VOM\")" ?name "TR_NZ_FIG_VOM" ?plot t ?evalType 'point)
ocnxlOutputExpr( "VT(\"/VOP\")" ?name "TR_NZ_FIG_VOP" ?plot t ?evalType 'point)
ocnxlEndTest() ; "LPF_NZ_tws05"

;====================== Specs ===================================================
ocnxlPutRangeSpec( "LPF_AC_tws05" "AC_LG[dB]" "0.05" "-0.05" )
ocnxlPutRangeSpec( "LPF_AC_tws05" "AC_BW_1dB[MHz]" "10.8M" "10.7M" )
ocnxlPutRangeSpec( "LPF_AC_tws05" "AC_Rejection@37M[dB]" "23" "21" )
ocnxlPutRangeSpec( "LPF_NZ_tws05" "TR_NZ_VO_RMS" "7.2m" "7m" )
ocnxlPutGreaterthanSpec( "LPF_NZ_tws05" "NZ_SNR[dB]" "32" )

;====================== Sweeps setup ============================================
ocnxlSweepVar("R1" "4.52k")
ocnxlSweepVar("R2" "5.83k")
ocnxlSweepVar("R3" "R1")
ocnxlSweepVar("C1" "10.25p")
ocnxlSweepVar("C2" "1p")
ocnxlSweepVar("AMP" "5m")
ocnxlSweepVar("FSIG" "1M")

;====================== Model Group setup =======================================

;====================== Corners setup ===========================================

;====================== Checks and Asserts setup ================================================
ocnxlPutChecksAsserts(?netlist nil)

;====================== Job setup ===============================================
ocnxlJobSetup( '(
"blockemail" "1"
"configuretimeout" "300"
"defaultcpuvalue" "1"
```

```
"defaultmemoryvalue" "1000"
"distributionmethod" "Local"
"jobruntype" "ICRP"
"estimatememoryvalue" ""
"estimationsimulationmode" "0"
"lingertimeout" "300"
"maxjobs" "1"
"name" "Maestro Default"
"preemptivestart" "1"
"providecpuandmemorydata" "1"
"reconfigureimmediately" "1"
"runpointsvalue" "5"
"runtimeout" "-1"
"scaleestimatedbycpu" "100"
"scaleestimatedbymemory" "100"
"showerrorwhenretrying" "1"
"showoutputlogerror" "0"
"startmaxjobsimmed" "1"
"starttimeout" "300"
"usesameprocess" "1"
"warndisklow" "0"
"warnthresholdvalue" "100"
) )

;====================== Disabled items ========================================

;====================== Run Mode Options ======================================

;====================== Starting Point Info ===================================

;====================== Run command ===========================================
ocnxlRun( ?mode 'sweepsAndCorners ?nominalCornerEnabled t
?allCornersEnabled t ?allSweepsEnabled t)
ocnxlOutputSummary(?exprSummary t ?specSummary t ?detailed t ?wave t)
ocnxlOpenResults()

;====================== End XL Mode command ===================================
ocnxlEndXLMode("assembler")
```

13.2 OceanScript 2

```
;====================Set to Maestro mode assembler ============================
ocnSetXLMode("assembler")
ocnxlProjectDir( "/local/ilakkis" )
ocnxlTargetCellView( "zoro_sl" "inductorTestBench" "maestro" )
ocnxlResultsLocation( "" )
ocnxlSimResultsLocation( "" )
ocnxlMaxJobFail( 20 )

;====================== Tests setup ===========================================

;---------- Test "IND_AC" -------------
ocnxlBeginTest("IND_AC")
simulator( 'spectre )
design( "zoro_sl" "inductorTestBench" "schematic")
modelFile(
    '("/data/tsmc/crn22ull/1P7M5X1U_UT_ALRDL/msrf/pdk
```

```
    /tn22crsp004w1_1_3_1p1a/tsmcN22/../models/tsmcN22.scs" "tsmcN22stdcell")
    '("/data/tsmc/crn22ull/1P7M5X1U_UT_ALRDL/msrf/pdk
    /tn22crsp004w1_1_3_1p1a/tsmcN22/../models/tsmcN22.scs" "tsmcN22io25")
    '("/data/tsmc/crn22ull/1P7M5X1U_UT_ALRDL/msrf/pdk
    /tn22crsp004w1_1_3_1p1a/tsmcN22/../models/tsmcN22.scs" "tsmcN22io25ln")
    '("/data/tsmc/crn22ull/1P7M5X1U_UT_ALRDL/msrf/pdk
    /tn22crsp004w1_1_3_1p1a/tsmcN22/../models/tsmcN22.scs" "pre_tttt")
)
analysis('ac ?freq "2.5G"  ?param "TK_CAP"  ?start "100f"
?stop "2p"  ?step "5f"  )
desVar(   "FRF" 2.5G )
desVar(   "TK_CAP" 1p )
envOption(
'firstRun  t
'analysisOrder  list("pz" "dcmatch" "stb" "tran"
 "envlp" "ac" "dc" "lf" "noise" "xf" "sp" "pss" "pac"
 "pstb" "pnoise" "pxf" "psp" "qpss" "qpac" "qpnoise"
 "qpxf" "qpsp" "hb" "hbac" "hbstb" "hbnoise" "hbxf" "sens" "acmatch")
)
option( 'pzSeverity  "None"
'noiseSeverity  "None"
'spSeverity  "None"
'acSeverity  "None"
'dcOpSeverity  "None"
'dcSeverity  "None"
'tranSeverity  "None"
)
saveOption( ?infoOptions list(list("modelParameter" "models" "rawfile" "" "" "" t)
list("element" "inst" "rawfile" "" "" "" t)
list("outputParameter" "output" "rawfile" "" "" "" t)
list("designParamVals" "parameters" "rawfile" "" "" "" t)
list("primitives" "primitives" "rawfile" "" "" "" t)
list("subckts" "subckts" "rawfile" "" "" "" t)
list("asserts" "assert" "rawfile" "" "" "" nil)
list("extremeinfo" "all" "logfile" "" "yes" "" nil)
list("allcap" "allcap" "file" "" "" "" nil)
list("simStatistics" "simstat" "rawfile" "" "" "" nil)
list("<Click_To_Add>" "none" "rawfile" "" "" "" nil) ) )
save( 'v "/vop" "/vom" )
save( 'i "/ICn/PLUS" "/ICp/PLUS" )
temp( 27 )
ocnxlOutputExpr( "db((VF(\"/vop\") - VF(\"/vom\")))"
?name "FIG_AC_vo" ?save t ?evalType 'point)
ocnxlOutputExpr( "xmax(FIG_AC_vo 1)" ?name "AC_COPT" ?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "(0.5 * (IF(\"/ICn/PLUS\") - IF(\"/ICp/PLUS\")))"
?name "FIG_AC_io" ?save t ?evalType 'point)
ocnxlOutputExpr( "value((FIG_AC_io / (VF(\"/vop\") - VF(\"/vom\"))) AC_COPT)"
?name "AC_YMAX" ?save t ?evalType 'point)
ocnxlOutputExpr( "(1.0 / real(AC_YMAX))" ?name "AC_RP" ?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "(2.0 * pi * VAR(\"FRF\"))" ?name "omega" ?save t ?evalType 'point)
ocnxlOutputExpr( "(1.0 / (AC_COPT * ((2.0 * pi * VAR(\"FRF\"))**2)))"
?name "AC_LP" ?plot t ?save t ?evalType 'point)
ocnxlOutputExpr( "(AC_RP / (AC_LP * omega))" ?name "AC_Q" ?plot t ?evalType 'point)
ocnxlOutputSignal( "/vo")
ocnxlOutputSignal( "/vop" ?save t)
ocnxlOutputSignal( "/vom" ?save t)
ocnxlOutputTerminal( "/ICn/PLUS" ?save t)
ocnxlOutputTerminal( "/ICp/PLUS" ?save t)
```

```
ocnxlEndTest() ; "IND_AC"

;====================== Sweeps setup ========================================

;====================== Model Group setup ===================================

;====================== Corners setup =======================================

;====================== Checks and Asserts setup ============================================
ocnxlPutChecksAsserts(?netlist nil)

;====================== Job setup ===========================================
ocnxlJobSetup( '(
"autoresume" "0"
"blockemail" "1"
"communicationtimeout" "180"
"configuretimeout" "300"
"defaultcpuvalue" "1"
"defaultmemoryvalue" "1000"
"distributionmethod" "Local"
"jobruntype" "ICRP"
"estimatecpuvalue" ""
"estimatememoryvalue" ""
"estimationsimulationmode" "0"
"lingertimeout" "300"
"maxjobs" "7"
"name" "Maestro Default"
"preemptivestart" "1"
"providecpuandmemorydata" "1"
"reconfigureimmediately" "1"
"runpointsvalue" "5"
"runtimeout" "-1"
"scaleestimatedbycpu" "100"
"scaleestimatedbymemory" "100"
"showerrorwhenretrying" "1"
"showoutputlogerror" "0"
"startmaxjobsimmed" "1"
"starttimeout" "300"
"userquotaspace" "0"
"userquotavalue" ""
"usesameprocess" "1"
"warndisklow" "0"
"warnthresholdvalue" "100"
) )

;====================== Disabled items ======================================

;====================== Run Mode Options ====================================

;====================== Starting Point Info =================================

;s2pfile=infile("/data/user/ilakkis/UBILITE/active/results/sandbox/s2pfileName.txt")
;gets(datatxt s2pfile)
;datatxt=strcat("results_" datatxt)
;datatxt=strcat( datatxt ".csv")
;datatxt=strcat("/data/user/ilakkis/UBILITE/active/results/current/" datatxt)
;ocnPrint(datatxt)
```

```
;====================== Run command ==========================================
ocnxlRun( ?mode 'sweepsAndCorners ?nominalCornerEnabled t
?allCornersEnabled nil ?allSweepsEnabled t)
ocnxlOutputSummary(?exprSummary t ?specSummary t ?detailed t ?wave t)
ocnxlOpenResults()
ocnxlExportOutputView( "/data/user/ilakkis/results.csv" "Detail-Transpose")
;ocnxlExportOutputView( datatxt "Detail-Transpose")

;====================== End XL Mode command ==================================
ocnxlEndXLMode("assembler")
;exit
/*****************************************************************
*                                                                *
*                     CCFreadCSV(fileName)                       *
*                                                                *
*  Read a CSV file - returns a list of param names (the first  *
*     row), and then a list of list of values (in order).      *
*    Does not attempt to handle quoted strings - leave that    *
*                as an exercise for the reader.                  *
*                                                                *
*****************************************************************/

procedure(CCFreadCSV(fileName)
    let((paramNames values inp line rows trailCR)
        ;---------------------------------------------------------------
        ; pattern for trailing carriage return
        ;---------------------------------------------------------------
        trailCR=pcreCompile("\n$")
        inp=infile(fileName)
        ;---------------------------------------------------------------
        ; First read the parameter names, then the rows of values.
        ; Uses tconc to ensure lines read in order
        ;---------------------------------------------------------------
        when(inp && gets(line inp)
            line=pcreReplace(trailCR line "" 1)
            paramNames=parseString(line ",")
            while(gets(line inp)
                line=pcreReplace(trailCR line "" 1)
                values=parseString(line ",")
                rows=tconc(rows values)
            )
        )
        when(inp
            close(inp)
        )
        list(paramNames car(rows))
    )
)
```