Étude et mise en œuvre d'une passerelle de communication en environnement hétérogène : Cosimulation MATLAB-CADENCE en vue de l'optimisation automatisée des circuits électroniques analogiques

Diana Carolina HERRERA GAMBA

ECOLE SUPERIEUR DE CHIMIE, PHYSIQUE ET ELECTRONIQUE DE LYON, FRANCE
UNIVERSIDAD INDUSTRIAL DE SANTANDER, COLOMBIE

Année Scolaire 2005-2006

Étude et mise en œuvre d'une passerelle de communication en environnement hétérogène : Cosimulation MATLAB-CADENCE en vue de l'optimisation automatisée des circuits électroniques analogiques

Diana Carolina HERRERA GAMBA

Projet de fin de'études pour l'obtention du diplôme d'ingénieur

Tuteurs :
M. Nacer ABOUCHI
M. Lioua LABRAK
M. Thierry TIXIER

ECOLE SUPERIEUR DE CHIMIE, PHYSIQUE ET ELECTRONIQUE DE LYON, FRANCE
UNIVERSIDAD INDUSTRIAL DE SANTANDER, COLOMBIE
Année Scolaire 2005-2006

Table des matières

Remerciements **v**

Abstract **vi**

Resumen **viii**

Résumé **x**

Introduction **xii**

1 Etude **1**

1.1 Objectifs 1

1.2 Interfaçage : Cadence - Matlab 2

1.2.1 Description Générale et Contraintes 2

1.2.2 Présentation des outils 2

1.2.3 Description de l'interface à réaliser 5

1.2.4 Démarche de réalisation de l'échange de données 6

1.3 Interfaçage et échange de données 7

1.3.1 Définition de la nature des données 7

1.4 Changement de topologie 9

1.4.1 Netlist 10

1.4.2 Détails sur la Netlist 10

1.4.3 Modification de la Netlist 12

1.4.4 Choix des outils 13

Bibliographie 14

2 Réalisation **15**

2.1 Test de l'échange entre deux applications 15

2.1.1 Description des applications réalisées 16

2.1.2 Tests réalisés 18
2.2 Échange de données entre Matlab et Cadence 19
2.2.1 Mise en œuvre de l'échange de données 20
2.3 Extraction des paramètres de performance 23
2.4 Extraction des paramètres du point de fonctionnement 24
2.4.1 Description du Point de Fonctionnement 24
2.4.2 Pourquoi récupérer les paramètres du point de fonctionnement ? 25
2.4.3 Mis en œuvre de l'extraction de paramètres 25
2.5 Changement de topologie 28
2.5.1 Mis en œuvre 28
Bibliographie 29

Conclusion **31**

Perspectives **32**
Bibliographie 34

Annexes **37**

Table des figures

1 Procès d'Optimisation - Localisation de l'objet du travail :*les passerelles* xv

1.1 Schéma Bloc du Système Cadence 3
1.2 Description de l'échange de données 6
1.3 Description du travail à réaliser 8
1.4 Architecture de l'amplificateur à deux étages à compensation de Miller 9
1.5 Netlist du schéma électrique simulé 11
1.6 Symbole du BTS 12

2.1 Algorithme du producteur 16
2.2 Algorithme du consommateur 17
2.3 Schéma de simulation 22
2.4 Fenêtre des résultats du point de fonctionnement pour un transistor 27
2.5 Fichier contenant les paramètres du point de fonctionnement pour un transistor 28
2.6 Partie de la Netlist à modifier 29

2.7 Topologie de l'amplificateur opérationnel 33

Liste des tableaux

2.1 Temps d'exécution en utilisant un fichier placé sur le disque local 19
2.2 Temps d'exécution en utilisant un fichier placé sur le dossier réseau 19
2.3 Définition des variables en accord avec le schéma de simulation 23
2.4 Paramètres du point de fonctionnement échangés entre Matlab et Cadence . . . 26

Remerciements

Je tiens à adresser mes vifs remerciements à l'Equipe de recherche AEME (Architectures Électroniques et Micro Électroniques) de l'École Supérieure de Chimie Physique et Électronique de Lyon (CPE Lyon) de m'avoir accuelli dans leurs locaux ainsi que pour m'avoir encadré et dirigé tout au long de mon stage.

Mes remerciements vont aussi à tous ceux qui m'ont soutenu de près et de loin.

Abstract

TITLE:

STUDY AND ACCOMPLISHMENT OF AN INTERFACE OF COMMUNICATION IN AN HETEROGENEOUS ENVIRONMENT. COSIMULATION BETWEEN MATLAB AND CADENCE IN VIEW OF THE AUTOMATED OPTIMIZATION OF INTEGRATED ANALOG CIRCUITS[*].

AUTHOR:

Diana Carolina HERRERA GAMBA.
Member of the AEME (Electronic and MicroElectronic Architectures) Research Team of CPE (Chemistry, Physics and Electronics) Lyon, France[†].

KEYWORDS:

Design of integrated analog circuits, optimization, automation.

DESCRIPTION:

The design of integrated analog circuits has the goal of being automated. The automated generation of the layout with given specifications is not yet possible and needs a considerable performance for the calculations. Indeed determination of circuits topologies, components size thus their placement on the silicon are as much complex tasks to realize with the awaited specifications.

Presently the design of digital circuits is well automated whereas it is absolutely not the case for analog circuits. With this work we contribute to set up an automated analog circuits synthesis system. In particular we engage in the cosimulation VI Abstract VII Matlab and Cadence with regard to the automated optimization of analog circuits.

[*] Project degree in research category
[†] Faculty of Physical – Mechanical Engineering – Electric and Electronics School - CPE Chemistry, Physics and Electronics, France - ABOUCHI, Nacer, TIXIER, Thierry, LABRAK, Lioua

We have developed a method of communication among two heterogeneous applications.

This approach has been validated and established in the framework of analog circuits' optimization. Thus we have allowed data exchange between Matlab and Cadence (operating point data, specifications, etc.).

Resumen

TITULO:
ESTUDIO Y REALIZACION DE UNA INTERFAZ DE COMUNICACIÓN EN UN MEDIO HETEROGENEO. COSIMULACION MATLAB – CADENCE EN VISTA DE LA OPTIMIZACION AUTOMATIZADA DE CIRCUITOS ELECTRONICOS ANALOGICOS.[*]

AUTORES:
Diana Carolina HERRERA GAMBA.
Miembro del Equipo de Investigación AEME (Arquitecturas Electrónicas y MicroElectrónicas) de la Escuela Superior de Química, Física y Electrónica de Lyon (CPE Lyon), Francia[†].

PALABRAS CLAVES:
Concepción de circuitos electrónicos analógicos, optimización, automatización.

DESCRIPCION:

El diseño de circuitos integrados analógicos necesita ser automatizado. La generación automática del layout a partir de las especificaciones dadas no es todavía posible o necesita una considerable capacidad de cálculo.
En efecto, la determinación de la topología de los circuitos, el tamaño de los componentes además de su posición sobre el silicio son tareas complejas que hay que realizar para responder a las especificaciones esperadas.

Actualmente, la concepción de circuitos digitales está automatizada mientras que este no es el caso para la concepción automática de circuitos analógicos.

[*] Proyecto de grado modalidad investigación
[†] Facultad de ingenierías fisicomecánicas – Escuela de Ingeniería Eléctrica y Electrónica. Escuela Superior de Química, Física y Electrónica de Lyon, Francia - ABOUCHI, Nacer, TIXIER, Thierry, LABRAK, Lioua

Por ejemplo, si la electrónica analógica conforma el 20 % dentro de la arquitectura de un sistema electrónico mixto (analógico y digital), el tiempo de concepción de esta parte corresponde al 80 % del tiempo de concepción global.
Este trabajo contribuye a la realización de un sistema de síntesis automática de circuitos analógicos. En particular, se intervino sobre la cosimulación entre Matlab y Cadence en vista de la optimización automatizada de circuitos analógicos.

Se elaboró una metodología de comunicación entre dos aplicaciones heterogéneas. Esta fue validada y puesta en marcha dentro del contexto de la optimización de circuitos analógicos.
De esta forma, se permitió el intercambio de datos entre Matlab y Cadence (datos del punto de reposo, especificaciones requeridas, etc.).

Résumé

TITRE :
ETUDE ET MISE EN ŒUVRE D'UNE PASSERELLE DE COMMUNICATION EN ENVIRONNEMENT HETEROGENE. COSIMULATION MATLAB - CADENCE EN VUE DE L'OPTIMISATION AUTOMATISEE DES CIRCUITS ELECTRONIQUES ANALOGIQUES.

AUTEUR :
Diana Carolina HERRERA GAMBA.
Membre de l'Équipe de Recherche AEME (Architectures Électroniques et MicroÉlectroniques) de l'École Supérieure de Chimie, Physique et Électronique de Lyon (CPE Lyon), France.

MOTS CLES :
Conception des circuits electroniques analogiques, optimisation, automatisation.

DESCRIPCION :

La conception des circuits intégrés analogiques a besoin d'être automatisée. La génération automatique de dessin des masques, à partir de spécifications données, n'est pas encore possible ou nécessite une puissance de calcul très importante.
En effet, la détermination de la topologie des circuits, la taille des composants ainsi que leur disposition sur le silicium sont autant des taches complexes à réaliser pour répondre aux spécifications attendues.

Actuellement, la conception des circuits numériques est déjà bien automatisée alors que ce n'est pas du tout le cas pour la conception automatique des circuits

analogiques.
Dans ce travail, nous contribuons à la mise en place d'un système de synthèse automatique de circuits analogiques. En particulier, nous intervenons sur la co-simulation Matlab - Cadence en vue de l'optimisation automatisée des circuits analogiques.

Nous avons élaboré une méthodologie de communication entre deux applications hétérogènes. Cette approche a été validée et mise en œuvre dans le cadre de l'optimisation des circuits analogiques. Ainsi nous avons permis l'échange de données entre Matlab et Cadence (données de point de repos, spécifications, etc.)

Introduction

Équipe de recherche AEME

Ce stage de fin d'études a été effectué au sein du laboratoire d'électronique de l'École Supérieure de Chimie Physique et Électronique de Lyon " CPE Lyon " avec l'équipe de recherche AEME (Architectures Électroniques et Micro Électroniques) qui est situé sur le campus universitaire de la DOUA à Villeurbanne (69).

Les travaux de recherche de l'équipe AEME se situent dans les domaines suivants :

- **Conception d'ASICs mixtes dédiés à la capture, au traitement et conditionnement de signaux issus de capteurs (humidité, vidéo, optique, etc.).**
- **Structures intégrées analogiques programmables (Field Programmable Analog Array).**
- **Modélisation de systèmes microélectroniques (VHDL, VHDL-AMS).**
- **Optique intégrée et systèmes optroniques, détecteurs infrarouges.**

Contexte

Pour bien comprendre l'intérêt de travail, il est nécessaire de resituer le contexte dans lequel développent aujourd'hui les ingénieurs de conception de circuits analogiques.

En effet, si la conception automatisée des circuits numériques est déjà bien maîtrisée, il n'en est pas du tout de même pour la conception des circuits analogiques. Pour les circuits numériques, les concepteurs sont aujourd'hui en mesure de réaliser des architectures aussi complexes que celles d'un microprocesseur en un minimum de temps car la synthèse et le dessin des masques se font de manière

totalement automatisés. A la différence, pour la plus simple des structures analogiques le travail peut s'avérer long et difficile ; aucune de deux étapes citées plus haut n'étant automatisée.
Une transposition vers l'analogique des méthodologies et des outils qui ont fait le succès du numérique doit donc être effectuée et de nouveaux outils doivent être développés.

Le but dans les années à venir, est de rendre automatique la conception des circuits analogiques en suivant ces quelques grandes étapes :

- **Choix de la topologie.**
- **Dimensionnement des transistors afin de répondre aux spécifications de l'utilisateur (en utilisant des algorithmes mathématiques d'optimisation).**
- **Dessin des masques d'un circuit.**

Donc l'une des phases la plus critique pour la conception analogique est de dimensionner les transistors afin de répondre aux spécifications de l'utilisateur. Alors, une phase d'optimisation du circuit est réalisée avant de générer les masques pour la fonderie.

Actuellement à l'intérieur de ce groupe de recherche le but est d'automatiser la conception des circuits électroniques analogiques.
Les travaux en cours doivent permettre la synthèse automatique de circuits analogiques destinés à l'interfaçage de capteurs.
De plus il s'agit de concevoir des circuits optimisés en fonction des souhaits de l'utilisateur (par exemple le Gain et la Bande passante dans le cas d'un amplificateur).

L'optimisation est réalisée sous Matlab qui fournira un ensemble de paramètres à un simulateur (le simulateur « Spectre » sous Cadence) afin de vérifier que les spécifications de l'utilisateur sont atteintes. Une fois les dimensions optimales définies il s'agit de générer le dessin des masques tout en respectant les règles de placement routage.

Pour la conception des circuits analogiques nous avons dénoté trois phases : l'ana-

lyse comportementale, l'analyse électrique au niveau schéma et l'analyse physique au niveau layout.
La figure 1 décrit l'approche permettant la synthèse à un niveau d'élaboration donnée (comportementale, schematic, physique). Pour ce niveau il s'agit de sélectionner une topologie parmi des architectures existantes et d'optimiser les dimensions pour répondre aux spécifications de l'utilisateur.

Le travail exposé dans ce rapport intervient au niveau des passerelles entre les différents outils (figure 1).

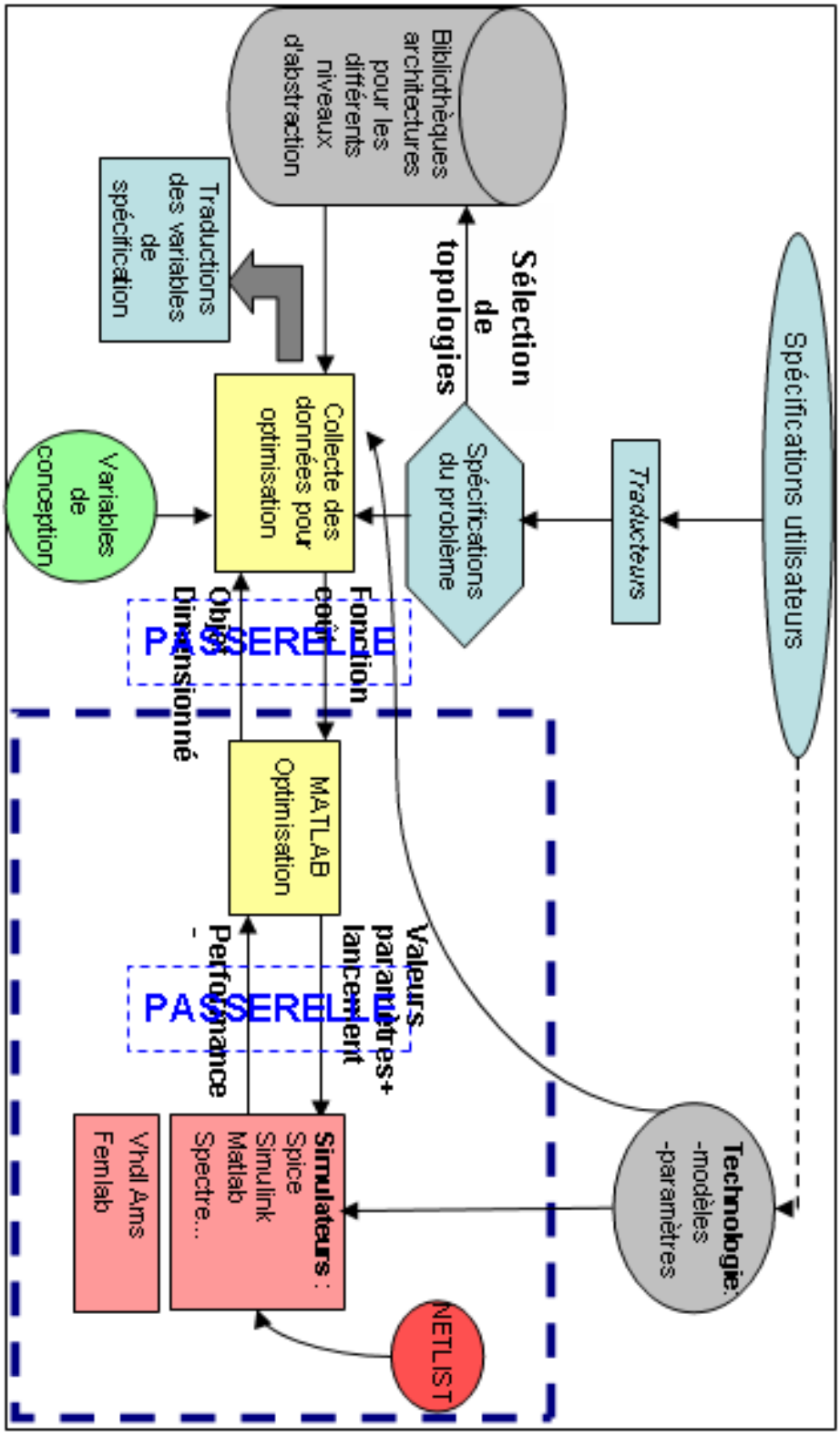


FIG. 1 – Procès d'Optimisation - Localisation de l'objet du travail :*les passerelles*

Chapitre 1

Etude

Sommaire

1.1 Objectifs 1
1.2 Interfaçage : Cadence - Matlab 2
1.2.1 Description Générale et Contraintes 2
1.2.2 Présentation des outils 2
1.2.3 Description de l'interface à réaliser 5
1.2.4 Démarche de réalisation de l'échange de données 6
1.3 Interfaçage et échange de données 7
1.3.1 Définition de la nature des données 7
1.4 Changement de topologie 9
1.4.1 Netlist 10
1.4.2 Détails sur la Netlist 10
1.4.3 Modification de la Netlist 12
1.4.4 Choix des outils 13
Bibliographie 14

1.1 Objectifs

Notre objectif est de proposer une méthode efficace pour mettre en œuvre la cosimulation Matlab Cadence afin d'automatiser le processus d'optimisation et accélérer l'extraction des paramètres de performances. Cela signifie que nous devons en premier lieu réaliser une liaison automatique entre Matlab et Cadence.

Dans un deuxième temps nous devons mettre en œuvre cette interface pour réaliser automatiquement l'opération de dimensionnement.

L'étude actuelle étant menée sur une architecture simple pour la validation du concept, le troisième objectif consiste à la généraliser pour rendre opérationnel le travail réalisé pour d'autres types de structures analogiques. Ceci, consiste à mettre en place une méthode pour effectuer un changement automatique de topologie.

1.2 Interfaçage : Cadence - Matlab

1.2.1 Description Générale et Contraintes

L'objectif de cette première partie est de concevoir une liaison automatique pour l'échange de données entre Matlab et Cadence.

Nous devons tenir compte des contraintes liées à un environnement logiciel hétérogène :
La simulation des circuits analogiques fonctionne sous Cadence, et l'optimiseur sous Matlab.

D'autre part les logiciels utilisés fonctionnent sous des systèmes d'exploitation pouvant être différents (Windows, Linux).

1.2.2 Présentation des outils

MATLAB

MATLAB constitue un logiciel de calcul numérique et scientifique. Il intègre l'analyse numérique, le calcul matriciel, le traitement de signaux et le graphique. A l'origine, le nom MATLAB signifiait "Matrix Laboratory".
Le lecteur qui souhaite avoir plus d'information sur ce sujet peut consulter la référence [1]

Description de l'environnement Cadence

Le logiciel Cadence (figure 1.1) est un outil qui sert pour la simulation et pour le dessin des masques [2]. Il comporte plusieurs outils, tels que :

- **L'outil « Virtuoso » qui permet de réaliser des schémas électriques.**
- **L'outil « Spectre » qui permet d'effectuer des simulations (AC, DC, TRAN,).**

– **L'interface « OCEAN » qui est un environnement de commande basé sur le langage « SKILL »(nous ferons une petite présentation par la suite).**

Comme le montre le schéma bloc de Cadence de la figure 1.1, le langage Skill est au coeur du système.

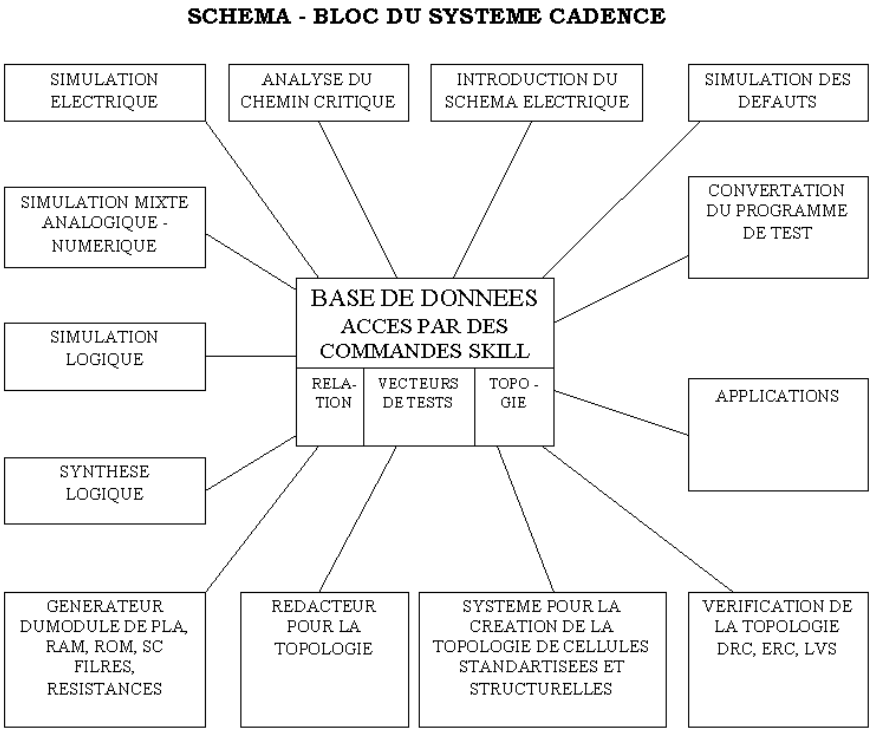


FIG. 1.1 – Schéma Bloc du Système Cadence

Ce sont des commandes Skill qui gèrent l'ensemble de fonctions Cadence. Le langage Skill développé par Cadence est un langage de très haut niveau qui nous permettra de décrire les simulations et de nous fournir le résultat sous forme graphique ou mumérique.

L'interface OCEAN

OCEAN « Open Command Environment for ANalysis » [3] est un utilitaire inclus dans Cadence, basé sur le langage Skill et destiné à commander les simulations.

Cet environnement de commande permet de générer tous les types de simulation

proposés par Cadence (tels que AC, DC, TRAN, etc.) et permet également de générer plusieurs simulations simultanément.
Ainsi, on peut à partir d'un terminal Unix activer l'interface OCEAN et commander Cadence directement.

Introduction au langage Skill

Skill [4] est le langage de commande de l'environnement Cadence.
Même s'il y a plusieurs menus et options, l'environnement Cadence utilise différentes fonctions Skill pour réaliser les tâches.
Dans la plupart des cas les fonctions Skill peuvent : Ouvrir la fenêtre principale (« design window ») de Cadence, Placer un composant, etc. D'autres fonctions Skill permettent de calculer des données de l'environnement.
L'avantage principal du langage Skill c'est qu'il permet d'effectuer avec une seule ligne de commande plusieurs étapes du travail.
Donc pour chaque simulation nous sommes en mesure d'écrire un programme en langage Skill capable de fournir les résultats des simulations sous forme graphique ainsi que sous forme numérique.
Le langage de programmation Skill nous laisse adapter notre environnement de conception selon nos besoins. Il fournit un environnement de programmation sûr et à niveau élevé qui manipule automatiquement les opérations traditionnelles des langages de programmation.
Les programmes en Skill peuvent être immédiatement exécutés dans l'environnement Cadence.
La seule documentation qui existe sur le langage Skill est l'aide proposée par Cadence.
L'environnement cadence permet l'élaboration des programmes en Skill sous une interface personnalisable par l'utilisateur. L'environnement de développement Skill contient des outils puissants de tracé et de mise au point.
Skill accroît l'investissement dans la technologie de Cadence parce qu'il permet de combiner la fonctionnalité existante et ajouter des nouvelles possibilités. Il permet de commander et d'accéder à tous les composants de l'environnement : L'interface de l'utilisateur du système de gestion, la base de données de conception, et les commandes de tous les outils de conception intégrés.

1.2.3 Description de l'interface à réaliser

Il est possible de faire communiquer les deux modules en passant par un fichier.
L'utilisation d'un fichier comme canal de communication est possible dans un environnement hétérogène. En effet un fichier peut être lu par n'importe quelle application, quel que soit le système d'exploitation.

Afin d'assurer l'échange correcte des données à l'aide de ce fichier, il est nécessaire de mettre en place un mécanisme de contrôle. Ce dispositif doit coordonner les applications accédant à la ressource partagée que constitue le fichier.

Une application dite « producteur » crée un fichier contenant des données à transmettre.
Une application dite « consommateur » lit le fichier pour en extraire les données et les exploiter.

Deux règles régissent ce transfert :

1. **Le producteur ne peux pas déposer un message tant que le message précédent n'a pas été consommé.**
2. **Le consommateur attend qu'un message soit produit pour le retirer.**

Habituellement le mécanisme utilisé pour contrôler l'accès à une ressource partagée est un « sémaphore ».
Un sémaphore est un signal échangé entre un producteur et un consommateur leur permettant de se synchroniser au moment de l'utilisation du fichier partagé et évitant tout conflit.

La mise en œuvre de l'échange bidirectionnelle de données entre deux applications nécessite, en plus du fichier contenant les données et du sémaphore, des opérateurs capables de produire (P_{11},...,P_{1i} et P_{21},..., P_{2j}) ou de consommer (C_{11},...,C_{1m} et C_{21},..., C_{2n}) (figure 1.2).

Un sémaphore associé à chaque fichier contrôle l'utilisation.
Les canaux de communication ainsi réalisés sont unidirectionnels.

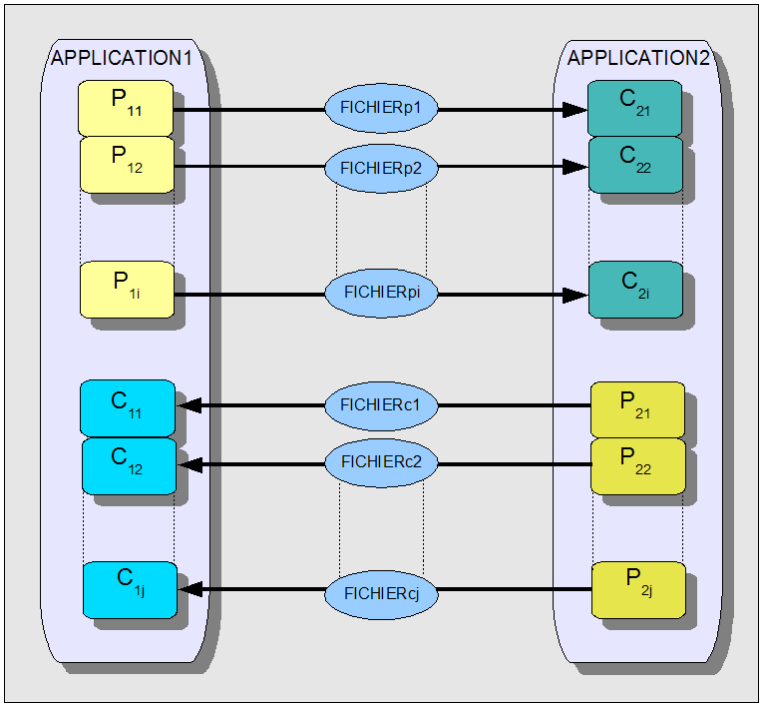


FIG. 1.2 – Description de l'échange de données

Une application est susceptible de produire (ou consommer) plusieurs informations à destination (ou en provenance) d'une autre application.

L'utilisation de plusieurs fichiers, chacun étant dédié à un canal de communication permet de ne pas engendrer des goulots d'étranglement. De plus elle facilite la gestion des données par les opérateurs producteurs et consommateurs : le canal de communication est dédié à un type de donnée en particulier.

1.2.4 Démarche de réalisation de l'échange de données

La création d'un canal de communication nécessite :

- **La création d'un producteur.**
- **La création d'un consommateur.**

Ces deux opérateurs devant être intégrés dans leurs applications respectives, ils doivent être codés dans le langage spécifique à cette application.
Pour évaluer les canaux de communication sans être obligé de programmer le producteurs et consommateurs dans des langages différents, nous choisissons de

coder le producteur et le consommateur en langage C. Cette évaluation doit en particulier nous permettre de mesurer la vitesse de transmission et de valider les algorithmes.

La seconde étape consiste à coder, dans le langage spécifique aux applications utilisés, les algorithmes validés lors de l'étape précédente.

Enfin, les opérateurs producteurs et consommateurs doivent être intégrés aux applications.

1.3 Interfaçage et échange de données :performances et point de fonctionnement

La passerelle que nous souhaitons réaliser s'effectue entre Matlab et Cadence. En effet, durant l'optimisation d'un circuit analogique, le calcul des dimensions des composants se fait sous Matlab et l'analyse des performances du circuit sous Cadence (figure 1.3).
Pour une itération :

- Matlab envoie des variables à Cadence afin d'obtenir les paramètres du point de fonctionnement.
- Matlab envoie des variables à Cadence afin d'obtenir des performances.
- Cadence envoie à Matlab les paramètres du point de repos.
- Cadence envoie à Matlab les performances simulées.

Observation : Pour obtenir un jeu de paramètres satisfaisant les spécifications, un nombre important d'itérations (une centaine) est nécessaire.

1.3.1 Définition de la nature des données

Après avoir analysé les données échangées entre les deux plateformes au cours d'une simulation, nous avons identifié le type des données que nous allons traiter dans notre application finale. Ce sont des données alphanumériques qui peuvent être stockées dans un fichier texte contenant une suite de caractères.

Il faut noter qu'en informatique, l'espace et le retour à la ligne sont considérés comme des caractères au même titre qu'une lettre, un chiffre ou un signe de

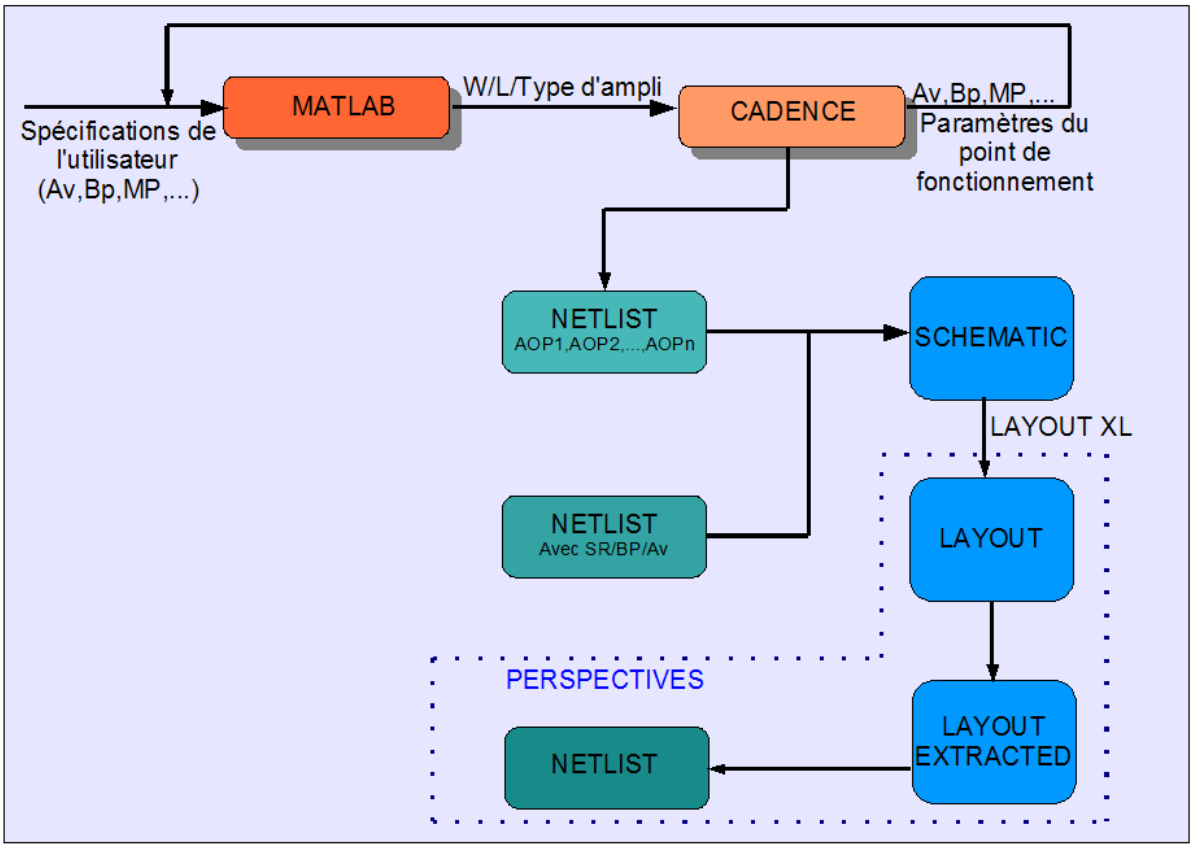


FIG. 1.3 – Description du travail à réaliser

ponctuation.

Les fichiers texte sont utilisés par de nombreux logiciels pour conserver les données de configuration. Ils sont également utilisés pour contenir les textes écrits en langages de programmation.

La plupart des langages de programmation offrent des fonctions prédéfinies pour manipuler du texte brut, ce qui rend la gestion des fichiers textes particulièrement accessible.

Un fichier texte peut simplement contenir du texte dans une quelconque langue. Dans ce cas il ne respecte aucune structure particulière.

La particularité d'un fichier texte est que l'ensemble du fichier respecte un codage de caractères standard.

Nous avons donc choisi d'utiliser des fichiers texte.

1.4 Changement de topologie

Comme nous avons dit précédemment, tous les travaux réalisés ont été basés sur une architecture simple qui est le modèle de l'amplificateur opérationnel qui constitue un des éléments de base des fonctions analogiques.
L'amplificateur utilisé pour ces études a été l'amplificateur à deux étages à compensation Miller (figure 1.4).

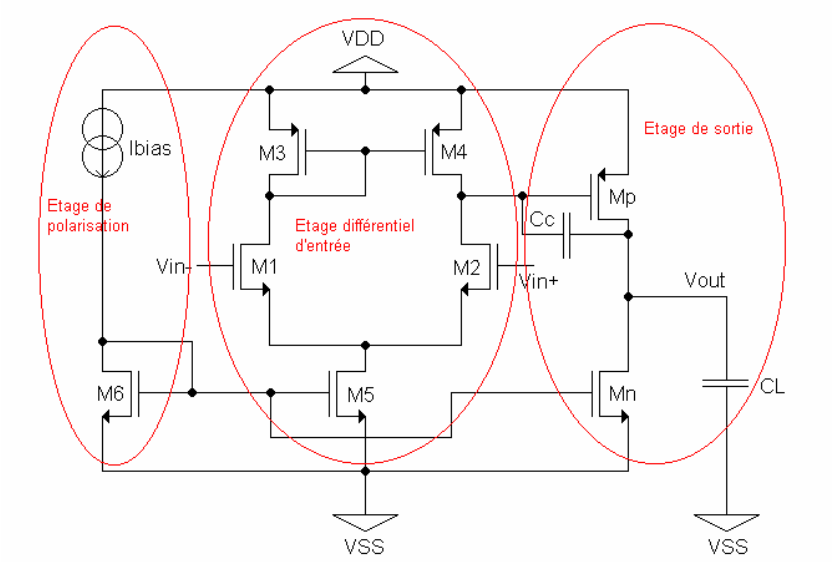


FIG. 1.4 – Architecture de l'amplificateur à deux étages à compensation de Miller

Maintenant, nous voulons rendre opérationnel notre travail pour d'autres circuits (d'autres types d'amplificateurs). Dans cette partie, nous allons expliquer comment nous pouvons changer la topologie des circuits à simuler.

Dans un premier temps nous allons lister les différentes solutions envisagées :

1. **La première idée est de récupérer la netlist afin de la modifier en changeant le nom de la topologie en récrivant d'autre topologie manuellement (à l'intérieur de la netlist).**

2. **La deuxième idée est de rendre automatique le changement de topologie des circuits. Pour cela, nous avons pensé à créer un programme capable d'effectuer à l'intérieur de la netlist les changements dont nous avons besoin.**

Pour l'écriture de ce programme nous avons pensé à utiliser le langage de programmation SKILL ou le langage de programmation P.E.R.L.

1.4.1 Netlist

Définition

Tous les outils d'analyse de design requièrent une description de la connectivité du design qui va être analysé.
Une Netlist est une description textuelle de la connectivité d'un schéma, elle est généré automatiquement pendant le procès de simulation et elle est utilisée plus tard comme une entrée pour le simulateur.
Les noms utilisés dans la Netlist sont fréquemment les noms que le concepteur a placés sur le schéma et ce sont les seuls noms que l'outil d'analyse de design reconnaît.

1.4.2 Détails sur la Netlist

Tout d'abord nous allons voir la première solution envisagée qui consiste à récupérer la netlist afin de la modifier. Cette solution est testée afin de savoir s'il est possible de changer la topologie des circuits.

La netlist est appelée dans le programme principal par la commande suivante :
design(« ~/cds36/Sim/sim_amplivar /spectre/schematic/netlist/netlist »)
Ce chemin représente la direction de la netlist et dépend du schéma électrique simulé (dans ce cas sim_amplivar).
Dans notre cas, nous avons trois schémas de simulation :

1. **sim_amplivar : permet d'effectuer les simulations du gain « G », la fréquence de coupure « fc », le produit Gain Bande " GWB " et la marge de phase « MP ».**
2. **sim_amplivar_mc : permet de simuler le Gain en mode commun « Gmc ».**
3. **sim_amplivar_SR : Permet de simuler le Slew Rate « SR ».**

La figure 1.5 représente un exemple de netlist.

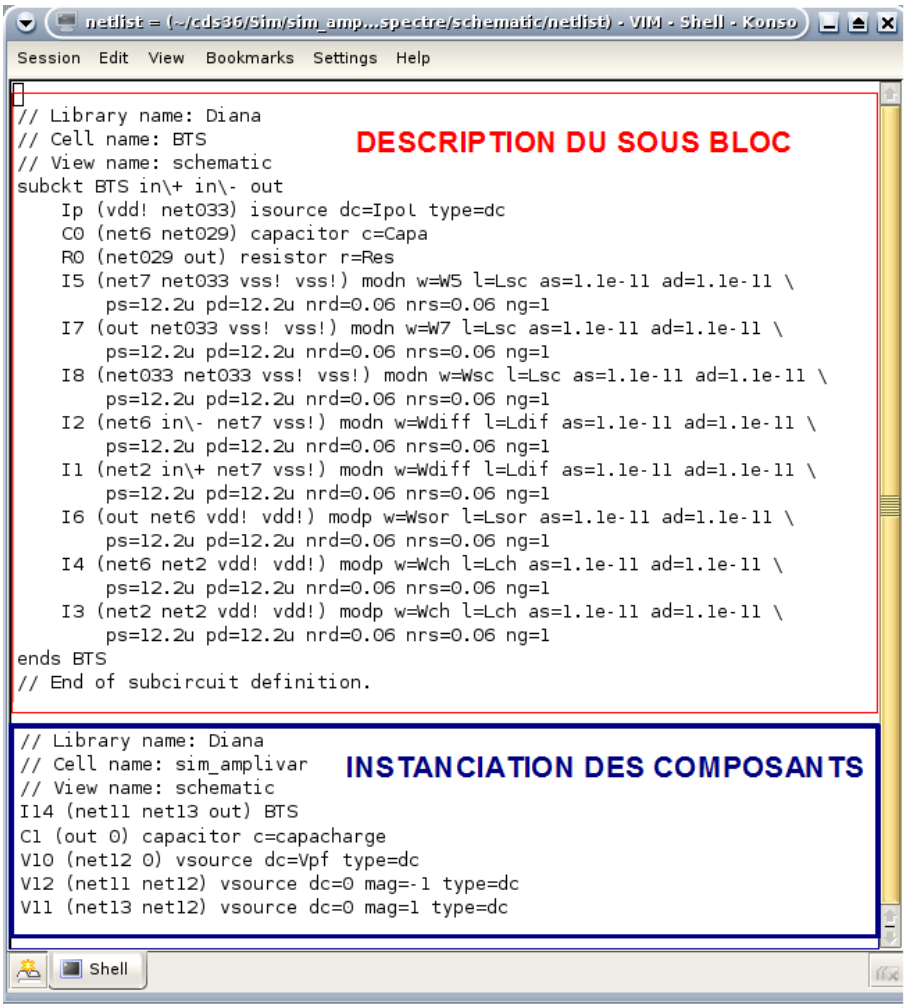


FIG. 1.5 – Netlist du schéma électrique simulé

Nous observons, en premier lieu la description du schéma de l'amplificateur opérationnel utilisé (« le sous bloc », dans ce cas le BTS), c'est-à-dire sa structure ainsi que le détail des composants utilisés pour ce bloc.

La structure pour définir un bloc, est la suivante :

Subckt BTS in\+in\-out

. . .

[Détails du schéma électrique du circuit BTS]

. . .

Ends BTS

D'où :

- **subckt → indique le sous bloc utilisé.**
- **BTS → indique le nom de la cellule utilisée (cellule créé sous Virtuoso).**
- **in\+in\-out → indique le nom des entrées/sorties du bloc, dans ce cas « in\+ et in\- » sont les entrées et « out » est la sortie.**

Voir le symbole (figure 1.6).

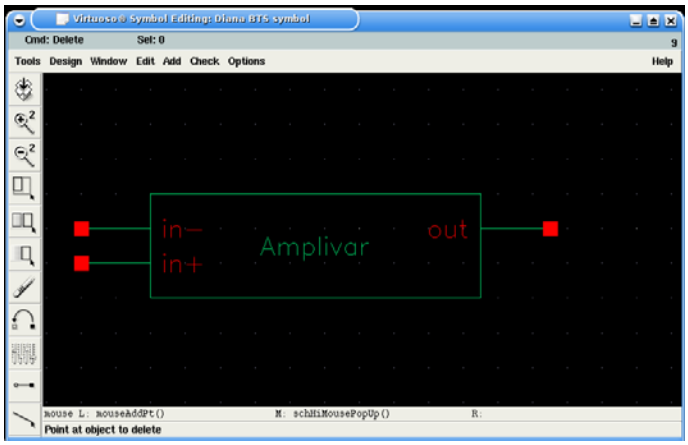


FIG. 1.6 – Symbole du BTS

La netlist est constituée d'une part :

- **des instances des composants (c'est-à-dire le nom des composants)**
- **des nœuds des composants (c'est-à-dire le nom des fils de connexion)**
- **du type des composants.**
- **des différents paramètres des composants.**

Correspondant au sous bloc.

D'autre part, de l'instanciation du circuit à simuler :

- **L'instance du composant**
- **Les nœuds**
- **Le nom du bloc (sous bloc utilisé)**

1.4.3 Modification de la Netlist

La première solution envisagée est de récupérer cette netlist afin de pouvoir la modifier.

Dans la suite, nous allons détailler la méthode énoncée précédemment afin de

pouvoir écrire une nouvelle structure d'amplificateur opérationnel.
Ainsi la prochaine étape est de rendre automatique le changement de topologie des circuits de simulation.
L'idée est que l'utilisateur choisit un amplificateur qui est stocké dans une liste d'amplificateurs disponibles (déjà existants ou réalises préalablement par l'utilisateur sous l'outil Virtuoso de Cadence). Une fois que l'utilisateur a effectué son choix, le but est d'affecter l'amplificateur choisit dans les différents schémas de simulation.

La solution qui est envisagée pour arriver aux résultats demandés (affecter l'amplificateur opérationnel choisit dans tous les schémas de simulation nommés précédemment) est la suivante :
Modifier la netlist en changeant uniquement le nom de l'amplificateur opérationnel dans la netlist. Dans la documentation de Cadence, nous avons remarqué que le rajout d'un composant s'écrit directement dans la netlist.
Donc nous continuons sur cette solution et nous essayons de réaliser ce changement à l'aide d'un programme écrit en langage P.E.R.L. grâce à sa capacité de manipuler des fichiers (notamment pour gérer plusieurs fichiers en même temps)et manipuler des textes (pour réaliser recherches et substitutions).

1.4.4 Choix des outils

Introduction au langage P.E.R.L

P.E.R.L [5] signifie « Practical Extraction and Report Language » que nous pourrions traduire par « langage pratique d'extraction et de génération de rapports ». Il a été Créé en 1986 par Larry Wall (ingénieur système), au départ pour gérer un système de « News » entre deux réseaux.

Caractéristiques du langage P.E.R.L
Le langage P.E.R.L c'est un :

- **Langage de programmation.**
- **Logiciel gratuit.**
- **Langage interprété (Pas de compilation, moins rapide qu'un programme compilé, chaque « script » nécessite d'avoir l'interpréteur P.E.R.L sur la machine pour s'exécuter).**

Bibliographie

[1] ***Matlab*. http ://en.wikipedia.org/wiki/MATLAB.**

[2] Nacer Abouchi and Lioua Labrak. *Prise en Main de l'Outil Cadence*. CPE Lyon, 2006.

[3] Cadence Design Systems, Inc. *OCEAN Reference*. Product Version 5.0.

[4] Cadence Design Systems, Inc. *SKILL Language Reference*. Product Version 06.30.

[5] Randal L. Schwartz and Tom Phoenix. *Introduction à Perl*. Editions O'Reilly, Paris, 2002.

Chapitre 2

Réalisation

Sommaire

2.1 Test de l'échange entre deux applications 15
2.1.1 Description des applications réalisées 16
2.1.2 Tests réalisés 18
2.2 Échange de données entre Matlab et Cadence 19
2.2.1 Mise en œuvre de l'échange de données 20
2.3 Extraction des paramètres de performance 23
2.4 Extraction des paramètres du point de fonctionnement 24
2.4.1 Description du Point de Fonctionnement 24
2.4.2 Pourquoi récupérer les paramètres du point de fonctionnement ? . . 25
2.4.3 Mis en œuvre de l'extraction de paramètres 25
2.5 Changement de topologie 28
2.5.1 Mis en œuvre 28
Bibliographie 29

2.1 Test de l'échange entre deux applications

Comme nous l'avons défini précédemment, nous avons un modèle de sémaphore assez simple, le cas du producteur et du consommateur.
Ce sémaphore a un fonctionnement typique dans lequel nous avons deux applications qui partagent un fichier utilisé comme canal de communication.
Un fichier partagé est une méthode, très simple et efficace, pour faire communiquer et synchroniser deux processus : le premier crée un fichier et écrit à l'intérieur, le deuxième lit son contenu.

Nous pouvons également utiliser l'existence ou la non-existence d'un fichier en guise de sémaphore.

2.1.1 Description des applications réalisées

Nous avons réalisé quelques programmes de test pour vérifier notre hypothèse et bien comprendre le fonctionnement de notre sémaphore. En fait nous avons construit le producteur et le consommateur dans deux environnements différents déjà connus : Visual C++ et Matlab.
Pour commencer, nous avons établi les algorithmes qui doivent être suivis par nos applications :

- **Algorithme du Producteur**

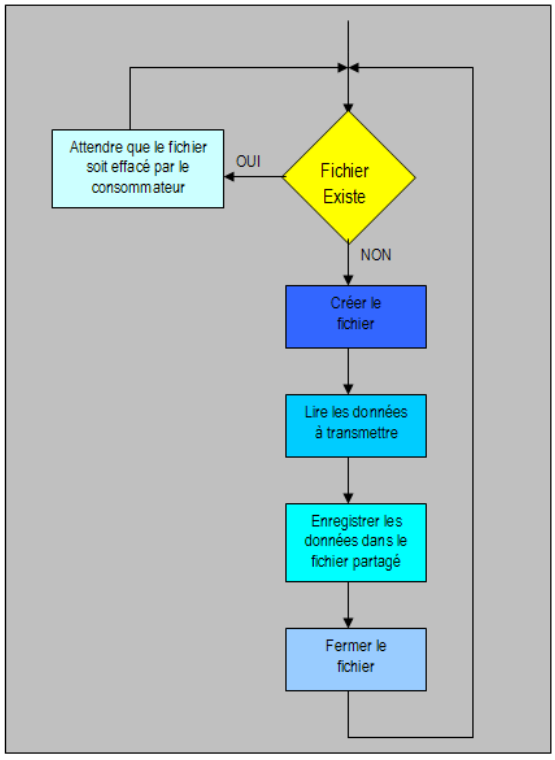


FIG. 2.1 – Algorithme du producteur

i) **Regarde si le fichier existe ou pas.**
ii) **Si le fichier existe, il attend que le fichier soit effacé par le consommateur. En attendant il vérifie continuellement si le fichier existe toujours.**
iii) **Si le fichier n'existe pas, il crée le fichier.**

iv) Lit les données qui doivent être transmis.

v) Enregistre les données obtenues dans le fichier partagé.

vi) Ferme le fichier.

– **Algorithme du Consommateur**

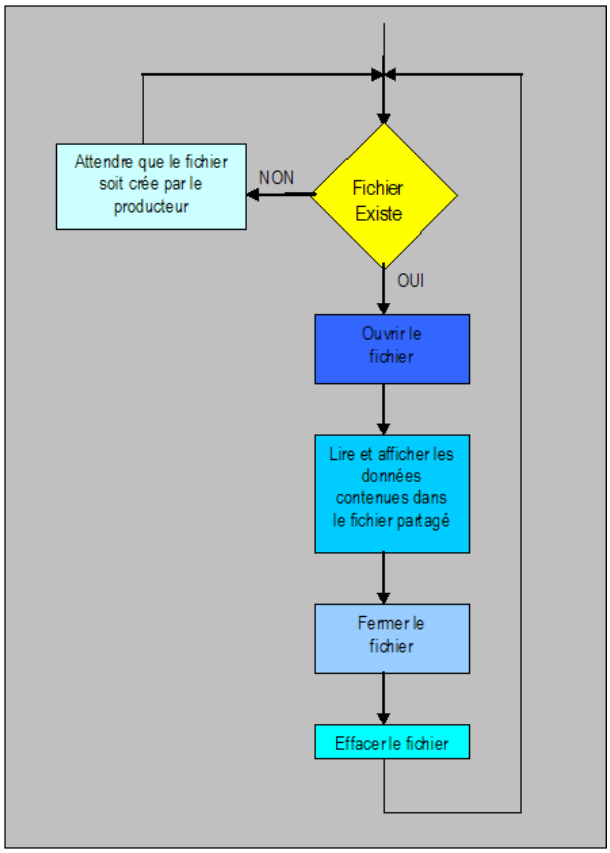


FIG. 2.2 – Algorithme du consommateur

i) Regarde si le fichier existe ou pas.

ii) Si le fichier n'existe pas, il attend que le fichier soit crée par le producteur. En attendant il vérifie continuellement si le fichier existe déjà.

iii) Si le fichier existe, il ouvre le fichier.

iv) Lit les données contenues dans le fichier partagé.

v) Affiche les données lues.

vi) Ferme le fichier.

vii) **Efface le fichier afin de signaler au producteur qu'il peut recommencer à produire des données.**

2.1.2 Tests réalisés

Nous avons testé l'échange de données entre :

a) **Deux applications en Visual C++ sous Windows [8].**

b) **Deux applications en MATLAB sous Windows [10].**
Nous avons écrit deux programmes (le producteur et le consommateur) de test qui suivent l'algorithme présenté précédemment.
Avec l'intention de pouvoir mesurer le temps d'échange de données, nous avons choisi l'heure et la date du moment auquel nous lançons le programme comme donnée pour envoyer vers le consommateur. Pour ça, nous avons ajouté au producteur une fonction qui donne l'heure et la date. Cette donnée est enregistrée dans le fichier et récupérée par le consommateur.

c) **Une application en MATLAB et une application en Visual C++ sous Windows.**
Nous avons déjà crée un producteur et un consommateur sous Visual C++ comme sous Matlab. Pour faire ce test nous devons seulement faire tourner le producteur sous Matlab avec le consommateur sous C++ et le producteur sous C++ avec le consommateur sous Matlab.

(Voir Listings).
Le principe de fonctionnement du producteur et du consommateur est le même décrit ci-dessus. Nous avons testé le fonctionnement des sémaphores de deux manières différentes :

a) **En utilisant un fichier partagé placé sur le disque local.**

b) **En utilisant un fichier partagé placé sur le dossier réseau.**

Observation des temps d'exécution

Les temps d'exécution obtenus pendant l'échange de données sont enregistrés dans les tableaux 2.1 et 2.2.

a) **En utilisant comme canal un fichier partagé placé sur le disque local (table 2.1).**

	Consommateur Visual C++	**Consommateur MATLAB**
Producteur Visual C++	0.5 s	0.5 s
Producteur Matlab	0.5 s	0.2 s

TAB. 2.1 – Temps d'exécution obtenu pendant l'échange de données en utilisant comme canal un fichier partagé placé sur le disque local

b) En utilisant comme canal un fichier partagé placé sur le dossier réseau (table 2.2).

	Consommateur Visual C++	**Consommateur MATLAB**
Producteur Visual C++	1 s	1 s
Producteur Matlab	1 s	1 s

TAB. 2.2 – Temps d'exécution obtenu pendant l'échange de données en utilisant comme canal un fichier partagé placé sur le dossier réseau

Observations :

- **Quand nous avons utilisé un fichier partagé placé sur le disque local nos avons noté que l'échange de données se réalisait plus rapidement.**
- **Quand nous avons utilisé un fichier partagé placé sur le dossier réseau l'échange de données se réalisait chaque seconde à cause du temps d'actualisation du réseau.**

2.2 Échange de données entre Matlab et Cadence

Maintenant que nous avons réussi à développer et tester l'échange de données entre deux applications en utilisant Visual C++ et Matlab, notre but est de lier notre programme sous Matlab avec Cadence, c'est-à-dire, nous devons créer deux applications comme celles que nous avons crée en Visual C++ et en Matlab mais cette fois ci en Skill.
Nous avons validé notre méthode, nous pouvons donc commencer notre travail en créant le producteur et le consommateur en Skill, qui est le langage de programmation sous Cadence [4].

Nous décidons de faire en premier lieu une découverte du langage Skill.
Si le travail à réaliser ressemble au travail que nous avons déjà fait pour écrire les programmes en Visual C++ et en Matlab, nous pouvons essayer de suivre les algorithmes définis précédemment pour créer notre producteur et notre consommateur en Skill en cherchant des commandes équivalentes [5].
Nous devrons en second lieu définir la structure des données à transmettre.

Pour commencer, nous décidons de tester le transfert des données. En effet même si le programme en Skill n'est pas encore capable d'introduire les données qu'il reçoit de Matlab dans Cadence nous serons sûrs que les données arrivent et que le transfert se réalise.

Par rapport à nos programmes en Matlab, nous devons regarder ce qu'il faudra rajouter pour réussir à échanger des données avec nos programmes en Skill tout en restant générique.

2.2.1 Mise en œuvre de l'échange de données

Nous commençons par établir la communication et l'échange des données entre le producteur et le consommateur en Skill en suivant le même algorithme établi pour le producteur et le consommateur en Matlab.

Pour commencer nous avons cherché les commandes que nous pouvions utiliser pour nos applications. Ensuite, nous avons testé l'échange de données entre le producteur et le consommateur en Skill.

Une fois que nous avons fini d'écrire les programmes en Skill et qu'ils fonctionnent entre eux, nous devons définir exactement quelles sont les données envoyées par Matlab et quelles sont les données que Cadence doit retourner vers Matlab et nous devons ensuite :

- **Tester l' échange de données entre le consommateur en Skill et le producteur en Matlab et vice-versa (Transfert de données dans le sens inverse afin de retourner les données).**
- **Vérifier que les données envoyées soient exactement celles qui sont reçues.**
- **Analyser les temps d'exécution de l'échange de données.**

Quand le transfert se réalise correctement dans les deux sens il faut s'assurer que les données que Matlab envoie soient automatiquement prises en compte par Cadence, pour qu'il puisse lancer une simulation et puis produire les données pour envoyer vers Matlab tout automatiquement.

Définition des données échangées

Comme nous avons dit précédemment, l'optimiseur, composé par des équations et algorithmes en Matlab, pourra donner des valeurs telles que le W (largeur de grille) et le L (longueur de grille) des transistors à partir des spécifications données telles que le Gain et la bande passante [6]. Avec cette définition et en analysant le schéma de simulation utilisé et les résultats obtenus, nous pouvons établir quels sont les paramètres envoyés par Matlab vers Cadence et quels sont les paramètres retournés.

Schéma de simulation

Nous pouvons observer le schéma de simulation utilisé dans la figure 2.3.
En accord avec ce schéma de simulation (figure 2.3) nous avons défini les variables selon la table 2.3.

Données envoyées par Matlab vers Cadence

Les variables définies dans la table 2.3 contiennent les différents paramètres de l'amplificateur [3] pour pouvoir effectuer la simulation. Celles ci sont les données que Matlab doit envoyer vers Cadence.

Données envoyées par Cadence vers Matlab

Une fois la simulation terminée, Cadence doit retourner à Matlab les résultats. Il renvoie alors les données suivantes :

- **La marge de phase.**
- **La bande passante.**
- **Le produit gain-bande.**
- **Le gain (en dB).**
- **Le CMRR (en dB).**
- **Le SR (en dB).**

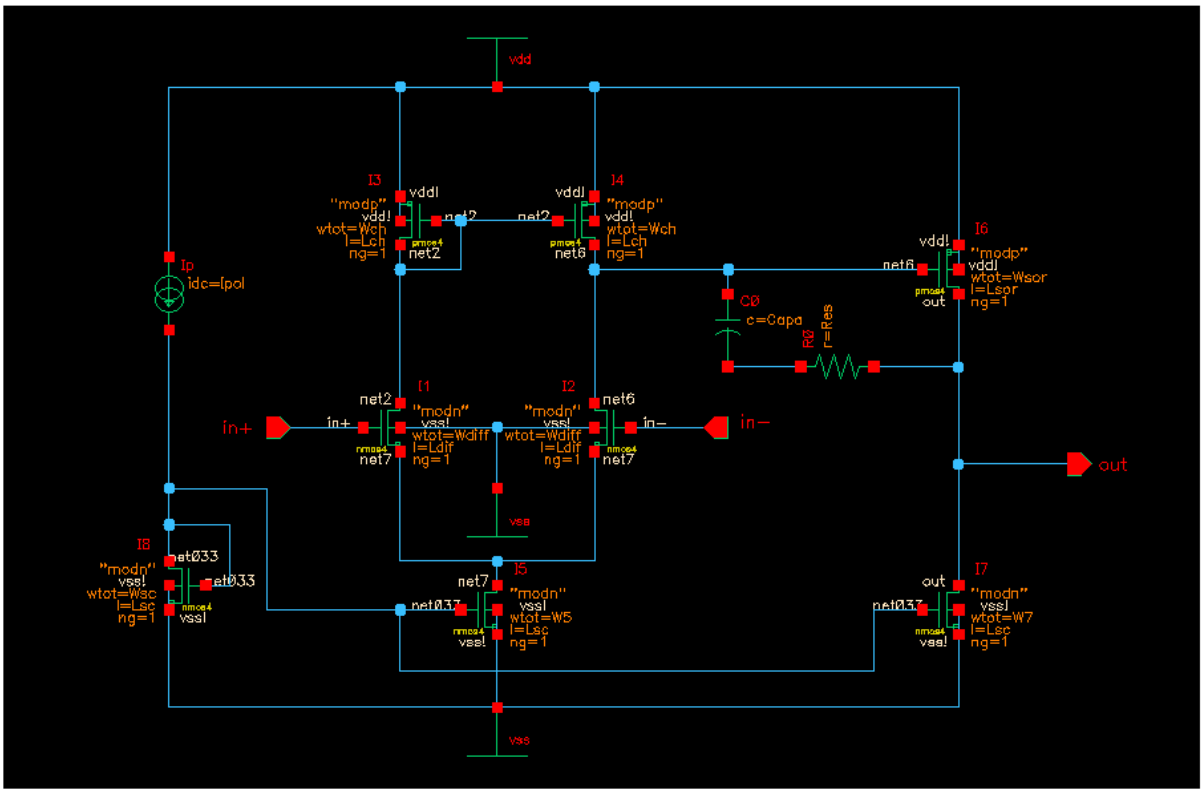


FIG. 2.3 – Schéma de simulation

Test de l'échange de données entre MATLAB et Skill

Nous avons testé l'échange de ces données entre Matlab et Cadence. Pour cela, nous avons du réaliser quelques changements a nos programmes de test en Matlab et en Skill, du fait des variations dans la quantité et la structure des données envoyées et reçus.
Nous avons alors changé la façon d'enregistrer et de lire les fichiers et la disposition des données dans le fichier.

Nous avons réussi à envoyer les paramètres de l'amplificateur de Matlab vers Cadence et séparément nous avons aussi réussi à transmettre les résultats de la

VARIABLES DEFINIES	VARIABLES DU SCHEMA DE SIMULATION
Courant	Ipol (Courant de polarisation)
Resistance	Res
Capacite	Capa (Capacité de Compensation)
W1	Wdiff (W des transistors I1 et I2
W3	Wch (W des transistors I3 et I4)
W5a	W5 (W du transistor I5)
W6	Wsor (W du transistor I6)
W7a	W7 (W du transistor I7)
W8	Wsc (W du transistor I8)
Ldiff	Ldiff
Lch	Lch
Lsc	Lss
Lsor	Lsor
Powersup	Vdd et Vss
Vpf	Tension mode commun
Capacharge	capacharge

TAB. 2.3 – Définition des variables en accord avec le schéma de simulation

simulation de l'amplificateur de Cadence vers Matlab.

2.3 Extraction des paramètres de performance

Maintenant que nous avons réussi à récupérer sous Cadence les données envoyées par Matlab, et à envoyer les résultats de la simulation de Cadence vers Matlab il faut que nous soyons capables de les utiliser, cela signifie que nous devons arriver à introduire dans Cadence les données que nous recevons de Matlab dans les variables correspondantes et puis lancer la simulation désirée.
Nous avons le programme Skill qui lance la simulation. Il s'agit du programme « simuvar_AC .ocn ». Ce programme réalise les simulations de l'amplificateur décrit précédemment.
Nous devons modifier ce programme pour le rendre capable de prendre les nouvelles valeurs des variables et à partir d'elles réaliser la simulation.
Nous avons fait les modifications et nous les avons enregistré dans le programme « simuvar_ACmod.ocn » (c'est le programme « simuvar_AC.ocn » modifié), de

plus il nous suffit d'ajouter à la fin de notre consommateur Skill la commande « load simuvar_ACmod.ocn » pour que notre consommateur juste après avoir reçu les données et les avoir enregistré dans les variables correspondantes puisse lancer la simulation.
Une fois la simulation terminée les résultats sont enregistrés dans le fichier « resultat.il ». Ce sont les données que nous devons retourner à Matlab.
Nous devons lire ces valeurs et les enregistrer dans un autre fichier capable d'être lu par Matlab facilement. Donc nous modifions notre producteur en Skill pour le rendre capable de lire le fichier « resultat.il » extraire les valeurs et créer le nouveau fichier (un fichier texte) qui doit être envoyé vers Matlab ou il sera reçu pour le consommateur Matlab.
En plus, nous ajoutons la commande « load (« producteur.ocn ») » à la fin du programme « simuvar_ACmod.ocn » pour qu'une fois la simulation fini notre producteur soit lancé automatiquement.

Finalement, le procès s'effectue comme nous voulions :

Le consommateur Skill « consommateur.ocn » reçoit le fichier envoyé par Matlab.
Lit les données et puis les introduit dans Cadence et lance le programme « simuvar_ACmod.ocn » qui réalise automatiquement la simulation.
Une fois la simulation finie, les résultats sont enregistrés dans le fichier « resultat.il ».
Avant de finaliser, le programme « simuvar_ACmod.ocn », lance le producteur (« producteur.ocn »), qui prend les données enregistrées dans le fichier « resultat.il » et crée un fichier texte avec les résultats et les envoie vers Matlab.
(Voir Listings).

2.4 Extraction des paramètres du point de fonctionnement

2.4.1 Description du Point de Fonctionnement

Les transistors présents dans un circuit électronique peuvent fonctionner dans des régimes différents. Suivant sont régime de fonctionnement, il sera associé un modèle différent (zone ohmique : résistance commandée - saturation : amplificateur - blocage : interrupteur ouvert).
Le régime de fonctionnement est identifié à partir de l'état statique du transistor.

En particulier, il est donné par le point de repos (« Bias point ») repéré à partir des valeurs des grandeurs électriques statique aux quelles il est soumis (Ids, Vgs pour un transistor MOS par exemple).
L'étude du comportement d'un transistor est donc intimement liée à son état statistique.

2.4.2 Pourquoi récupérer les paramètres du point de fonctionnement ?

L'optimiseur est composé d'algorithmes itératifs décrits en langage Matlab. Ces algorithmes minimisent une fonction coût traduisant la bonne adéquation entre les performances obtenues et souhaitées pour le circuit étudié.
Pour cela, il résout numériquement un jeu d'équations du modèle du circuit pour déterminer la taille W et L des transistors [7, 11]. Pour réaliser le traitement, la connaissance précise du point de repos des transistors permet d'obtenir avec une meilleur précision la valeur des paramètres du modèle, et de ce fait une meilleur convergence.

Ainsi, nous choisissons d'obtenir la valeur du point de repos directement à partir d'une simulation électrique. L'opération de simulation étant réalisé sous Cadence, ce choix impose la mise en place d'un nouveau canal de communication.

Nous avons dressé la liste des paramètres du point de fonctionnement qui doivent être récupérés et renvoyés vers Matlab dans la table 2.4.

2.4.3 Mis en œuvre de l'extraction de paramètres du point de fonctionnement

Pour effectuer la récupération des paramètres du point de fonctionnement, nous utilisons le même principe que pour l'échange des variables entre Matlab et Cadence expliqué précédemment.

En fait, nous conservons le même producteur Matlab (« producteur.m », Voir Listings) qui envoie des valeurs telles que les W et les L des transistors vers Matlab. Nous devons créer un consommateur Cadence capable de recevoir ces données et de lancer une simulation au point de repos (nous effectuons la simulation au point de repos pour vérifier qu'on est bien dans la zone de saturation) [9].

Nom du Paramètre	Description du Paramètre	Unité
IDS	Courant Drain-Source	A
VGS	Tension Grille-Source	V
VDS	Tension Drain-Source	V
VBS	Tension Substrat-Source	V
VTH	Tension de Seuil	V
VDSAT	Tension de saturation Drain-Source	V
GM	Transconductance (dIds/dVgs)	mho
GDS	Transconductance (dIds/dVds)	mho
GMBS	Transconductance (dIds/dVbs)	mho
CBD	Capacité Substrat-Drain	F
CBS	Capacité Substrat-Source	F
CGS	Capacité Grille-Source	F
CGD	Capacité Grille-Drain	F
CGB	Capacité Grille-Substrat	F
PWR	Puissance	W

TAB. 2.4 – Liste des paramètres du point de fonctionnement qui doivent être récupérés et renvoyés vers Matlab

A ce programme qui fait la simulation, nous devons rajouter la commande « SelectResults('dcOpInfo') » pour qu'il puisse donner comme résultat les paramètres du point de fonctionnement.
Une fois la simulation réalisé, nous obtenons pour chaque transistor les paramètres du point de fonctionnement comme nous observons dans la figure 2.4.

Maintenant que nous avons réussi à faire la simulation et obtenir les paramètres nous devons chercher une méthode pour ressembler tous les paramètres de tous les composants dans un fichier pour retourner à Matlab.
Nous ajoutons au consommateur des commandes pour lui rendre capable de récupérer les paramètres de chaque composant, les ressembler et les mettre dans un fichier.
La méthode consiste en envoyer les paramètres de chaque composant dans un fichier différent (figure 2.5).

Une fois que nous avons les paramètres dans des fichiers, nous pouvons lire les va-

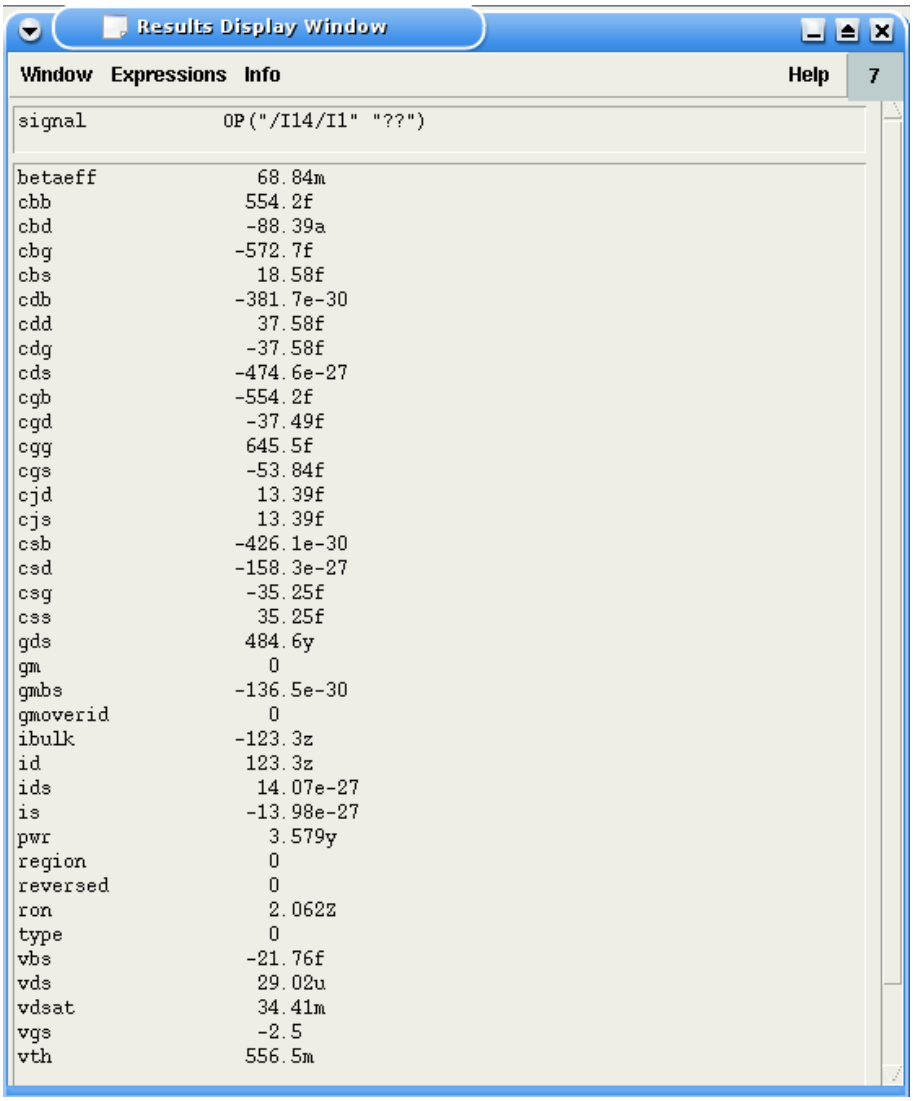


FIG. 2.4 – Fenêtre des résultats du point de fonctionnement pour un transistor

leurs et une fois lues les rassembler toutes et puis convertir les symboles de type p (pico), m (mili), u (micro), y (yocto) (Voir Annexe) en données numériques pour qu'ils puissent être interprétés (Matlab n'accepte pas ce type de symboles) et les insérer dans un fichier texte et l'envoyer vers Matlab (« cproducteurp.ocn » Voir Listings).

Comme nous avons vu, nous avons retourné a Matlab tous les paramètres du point de fonctionnement, même ceux qui ne sont nécessaires pour le traitement. De cette façon l'ensemble des informations liées au point de repos sont disponibles

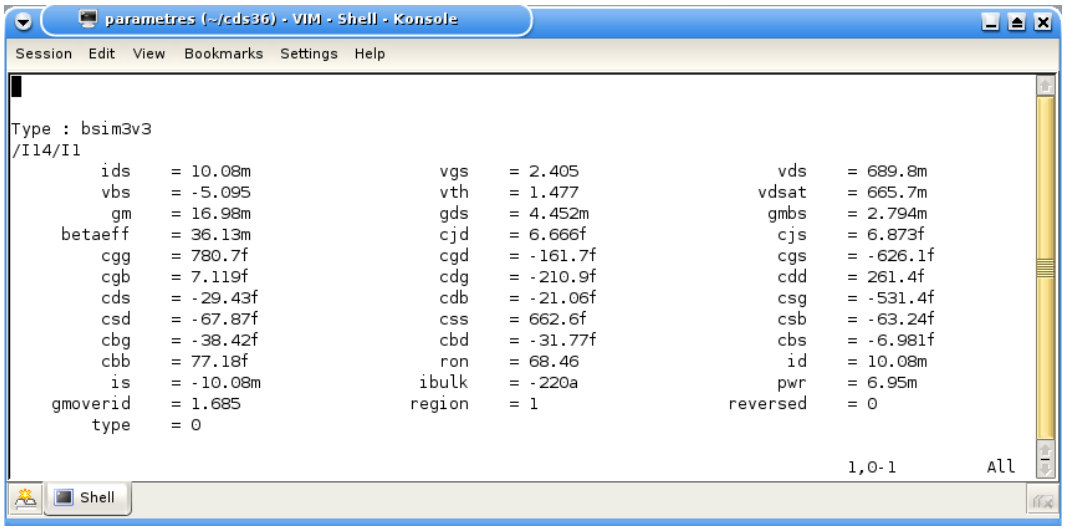


FIG. 2.5 – Fichier contenant les paramètres du point de fonctionnement pour un transistor

pour être utilisées par des traitements plus complets sous Matlab.
Pour recevoir le fichier contenant les paramètres, nous devons créer un consommateur en Matlab (« consommateurp.m » Voir Listings) capable de lire le contenue et introduire en Matlab les paramètres reçus pour être utilisés.

Finalement, l'optimiseur utilise ces données reçues pour produire des nouvelles valeurs de W et de L et la boucle continue.

2.5 Changement de topologie

2.5.1 Mis en œuvre

En profitant des qualités du langage P.E.R.L pour la manipulation des fichiers et des textes, nous avons créé un programme capable d'ouvrir une netlist et cherche dans son contenu l'endroit où il est défini l'amplificateur à utiliser et le remplacer par celui qu'on veut utiliser.

L'utilisateur donnera dans le programme principal (Matlab) le nom de l'amplificateur qu'il souhaite utiliser et celui-là sera directement utilisé et remplacé dans la netlist (figure 2.6).
Nous appelons ce programme depuis le consommateur Skill avec l'aide de quelques commandes en langage Skill . Une fois qu'il a reçu les données provenant de Mat-

lab (entre les quels on trouve en premier le nom de l'amplificateur à utiliser) il lance le programme de remplacement en P.E.R.L (modnetlist.pl) et il remplace automatiquement le nom de l'amplificateur à utiliser dans les netlists de tous les schémas de simulation.

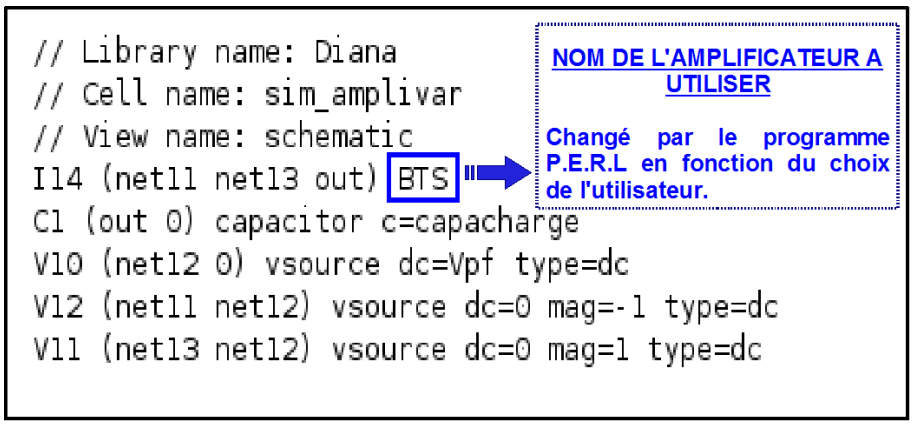


FIG. 2.6 – Partie de la Netlist à modifier

Bibliographie

[1] Phillip E. Allen and Douglas R. Holberg. *CMOS Analog Circuit Design.* Oxford University Press, second edition, 2002.

[2] R. Jacob Baker, Harry W. Li, and David E. Boyce. *CMOS Circuit Design, Layout and Simulation.* IEEE Press, 1998.

[3] J. Blot. *Les transistors, Eléments d'intégration des circuits analogiques.* Dunod, 1995.

[4] Cadence Design Systems, Inc. *Cadence User Interface SKILL Functions Reference.* Product Version 5.0.

[5] Cadence Design Systems, Inc. *Cadence SKILL Language User Guide*, June 2000. Product Version 06.00.

[6] J-M. Fouchet and A. Perez-Mas. *Electronique Pratique.* Dunod, 1996.

[7] Paul R. Gray, Paul J. Hurst, Stephen H. Lewis, and Robert J. Meyer. *Analysis and Design of Analog Integrated Circuits.* John Wiley & Sons Inc., 4th edition, 2001.

[8] **David J. Kruglinski, George Shepherd, and Scot Wingo.** ***Programming Microsoft Visual C++.*** **Microsoft Press, 5th edition, 2001.**

[9] **Jacob Millman and Arvin Grabel.** ***Microélectronique.*** **Ediscience International (McGraw-Hill), 1995.**

[10] **Mohand Mokhtari.** ***Matlab 5.2 et 5.3 et Simulink 2 & 3 pour étudiants et ingénieurs.*** **Springer-Verlag, 2000.**

[11] **Adel S. Sedra and Kenneth C. Smith.** ***Microelectronic Circuits.*** **Oxford University Press, 4th edition, 1997.**

Conclusion

L'aboutissement de ce stage est tout d'abord, la découverte du langage Skill et la reprise des logiciels tels que Visual C++ et Matlab, puis, l'obtention des résultats suivants :

Dans notre projet, nous avons décrit un outil complètement automatique pour la réalisation de l'échange de données entre Matlab et Cadence, appliqué au cas plus concret de l'optimiseur et l'extracteur de performances.
La méthode est fondée sur le principe du sémaphore en utilisant un fichier partagé comme canal de communication.
Cet outil consiste principalement en deux applications en Matlab (producteur et consommateur) et deux applications en Cadence (producteur et consommateur) qui réalisent automatiquement l'échange des données entre elles. Cet outil est fonctionnel et généralisable à d'autres applications.

Nous avons implanté le changement de topologie et maintenant l'optimisation est possible pour plusieurs circuits d'une même famille (d'autres types d'amplificateurs opérationnels). Il suffit de spécifier dans l'optimiseur le nom de l'amplificateur que nous souhaitons utiliser (qui doit avoir été crée précédemment en Cadence avec l'outil « Virtuoso »).
Nous avons réalisé des scripts P.E.R.L capables de lire le contenu d'une netlist d'un schéma de simulation et modifier dans son contenu le type d'amplificateur utilisé.

La récupération des paramètres du point de fonctionnement a été effectué, cela permet à l'optimiseur de réaliser la détermination des tailles des transistors de manière optimale et plus précise.

Perspectives

Les objectifs que nous avons défini au début et tout au long de ce stage pour réussir à effectuer la cosimulation Matlab Cadence en vue de l'optimisation automatisée des circuits électroniques analogiques ont été atteints. Néanmoins, le chemin pour la conception automatique est encore long.
Il faut préciser qu'il y a encore beaucoup de travail à réaliser et que les travaux réalisés pendant ce stage et décrits tout au long de ce rapport peuvent encore être améliorés et modifiés pour élargir leur utilisation à d'autres cas.

Le but du projet global est la conception automatique de circuit jusqu'au dessin des masques (layout, description des géometries des masques). Donc, dans un travail futur le but sera de permettre la génération automatique du layout.

Le dessin des masques se fait dans une fenêtre de type Layout. Pour cela il faut créer une vue layout sur la cellule désirée. L'outil de conception du Layout s'appelle « Virtuoso » [3].
« Virtuoso XL » ou « Layout XL » (anciennement "DLE") de Cadence permet de travailler sur le layout en gardant une connexion permanente avec le schéma. Il génère la vue layout à partir de la vue schématique. Le schéma doit contenir des symboles de cellules paramétrées possédant une vue layout compatible avec LayoutXL.

Nous voulons changer automatiquement les dimensions de la vue existante, en respectant les règles de dessin.
Des travaux sur ce sujet on déjà été entamés ; il s'agit d'un outil efficace pour la réutilisation de dessin des masques analogiques. Il utilise comme entrée un dessin des masques analogique, les tailles des dispositifs W et L (largeur et longueur de grille), pour générer automatiquement un dessin des masques correspondant aux tailles (W', L') des dispositifs sous les nouvelles caractéristiques. Donc, un

nouveau dessin des masques est généré automatiquement, tout en respectant les règles de dessin et en préservant les propriétés du dessin des masques original tel que le placement des dispositif, la symétrie et le routage.
Ce travail est basé sur l'architecture montrée dans la figure 2.7 :

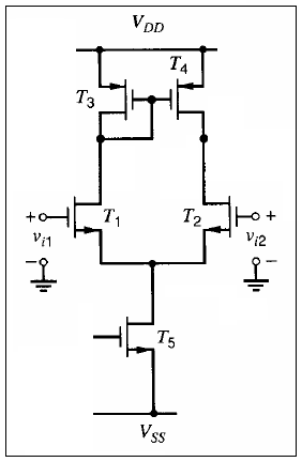


FIG. 2.7 – Topologie de l'amplificateur opérationnel

Comme nous pouvons observer sur la figure il ne permet pas une application sur une architecture de type OTA. Nous avons essayé la reprise de ce travail pour le modifier et le rendre opérationnel pour d'autres types d'amplificateurs, mais il s'agit d'un travail complexe et long.

Nous avons essayé de faire le dimensionnement du layout. Nous avons prospecté la possibilité de réaliser la synthèse automatique à partir de Skill, en utilisant la commande « lxGenFromSource » [2] et de l'outil layout XL de Cadence [3].
Nous avons trouvé une méthode pour réaliser le dimensionnement qui consiste à récupérer le fichier contenant la netlist du layout et de la modifier d'une façon similaire a celle que nous avons utilisée pour le dimmensionnement de la vue schématique. Une fois le dimensionnement du layout fait, nous pouvons générer la vue extracted [1] et faire la simulation de cette vue (simulation « post layout ») en utilisant le simulateur Spectre, afin de vérifier que les spécifications sont toujours atteintes. Cette simulation devrait donner les mêmes résultats que la simulation du schématique. Cette méthode fonctionne mais le passage de paramètres pour réaliser le dimensionement automatique du schématique vers le layout est problématique.

L'outil « Virtuoso XL »génère automatiquement le layout à partir d'une vue schématique mais dans notre cas, nous utilisons des variables au lieu des valeurs numériques fixes et Virtuoso XL ne reconnaît pas les variables que nous avons assignées. Il n'accepte pas de paramètres, pour générer une vue layout a partir d'une vue schématique il a besoin de connaître les valeurs numériques des variables. Il faudra alors développer une méthode pour effectuer ce passage de paramètres.

Dans ce travail nous avons effectué un changement de topologie pour pouvoir utiliser d'autres types d'amplificateurs. Un autre travail à réaliser serait de rendre opérationnel les travaux réalisés pour n'importe quel type de topologie et pas simplement un amplificateur. Pour cela, il suffirait d'établir quels sont les variables que nous voulons transmettre, réaliser les schémas à l'aide de l'outil « Virtuoso » et modifier le nombre de paramètres échanges entre Matlab et Cadence.

Bibliographie

[1] Cadence Design Systems, Inc. *Cell Design Tutorial.* Chapter 4. Verifying the Multiplexer Layout.

[2] Cadence Design Systems, Inc. *Custom Layout Skill Functions Reference.* Product Version 5.0.

[3] Cadence Design Systems, Inc. *Virtuoso XL Layout Editor User Guide.* Chapter 6. Generating Your layout with Virtuoso XL layout Editor.

Bibliographie

[1] *Matlab.* http ://en.wikipedia.org/wiki/MATLAB.

[2] *Preparación de textos con LATEX.* http ://www.uv.es/ ivorra/Latex/LaTeX.pdf.

[3] Nacer Abouchi and Lioua Labrak. *Prise en Main de l'Outil Cadence.* CPE Lyon, 2006.

[4] Phillip E. Allen and Douglas R. Holberg. *CMOS Analog Circuit Design.* Oxford University Press, second edition, 2002.

[5] R. Jacob Baker, Harry W. Li, and David E. Boyce. *CMOS Circuit Design, Layout and Simulation.* IEEE Press, 1998.

[6] J. Blot. *Les transistors, Eléments d'intégration des circuits analogiques.* Dunod, 1995.

[7] Cadence Design Systems, Inc. *Cadence User Interface SKILL Functions Reference.* Product Version 5.0.

[8] Cadence Design Systems, Inc. *Cell Design Tutorial.* Chapter 4. Verifying the Multiplexer Layout.

[9] Cadence Design Systems, Inc. *Custom Layout Skill Functions Reference.* Product Version 5.0.

[10] Cadence Design Systems, Inc. *OCEAN Reference.* Product Version 5.0.

[11] Cadence Design Systems, Inc. *SKILL Language Reference.* Product Version 06.30.

[12] Cadence Design Systems, Inc. *Virtuoso XL Layout Editor User Guide.* Chapter 6. Generating Your layout with Virtuoso XL layout Editor.

[13] Cadence Design Systems, Inc. *Cadence SKILL Language User Guide*, June 2000. Product Version 06.00.

[14] J-M. Fouchet and A. Perez-Mas. *Electronique Pratique.* Dunod, 1996.

[15] **Paul R. Gray, Paul J. Hurst, Stephen H. Lewis, and Robert J. Meyer.** ***Analysis and Design of Analog Integrated Circuits.*** **John Wiley & Sons Inc., 4th edition, 2001.**

[16] **David J. Kruglinski, George Shepherd, and Scot Wingo.** ***Programming Microsoft Visual C++.*** **Microsoft Press, 5th edition, 2001.**

[17] **Jacob Millman and Arvin Grabel.** ***Microélectronique.*** **Ediscience International (McGraw-Hill), 1995.**

[18] **Mohand Mokhtari.** ***Matlab 5.2 et 5.3 et Simulink 2 & 3 pour étudiants et ingénieurs.*** **Springer-Verlag, 2000.**

[19] **Randal L. Schwartz and Tom Phoenix.** ***Introduction à Perl.*** **Editions O'Reilly, Paris, 2002.**

[20] **Adel S. Sedra and Kenneth C. Smith.** ***Microelectronic Circuits.*** **Oxford University Press, 4th edition, 1997.**

Annexes

Schéma de l'amplificateur

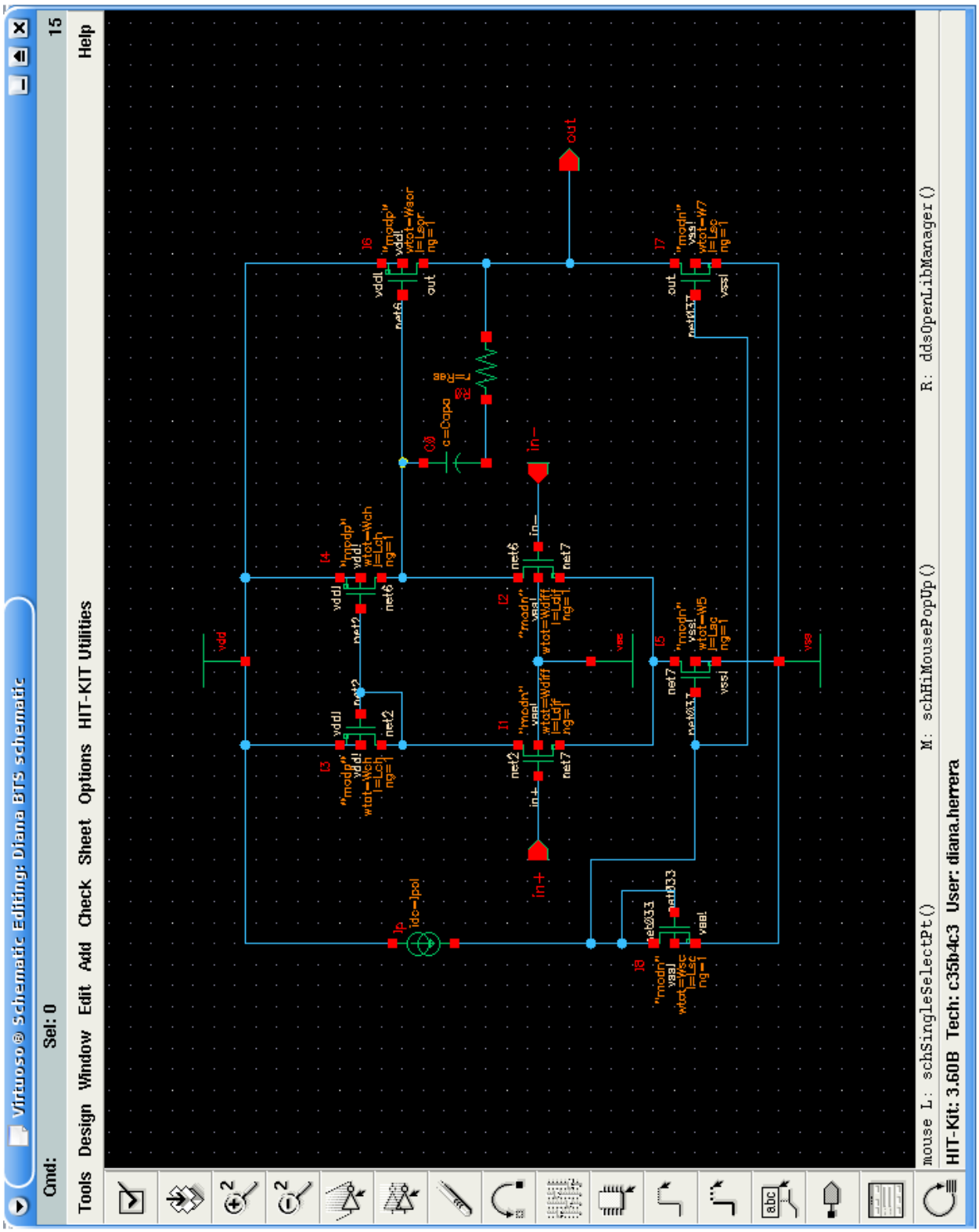


Schéma de simulation du gain, de FC du GBW (Gain Bandwidth Product) et de la MP(Marge de Phase)

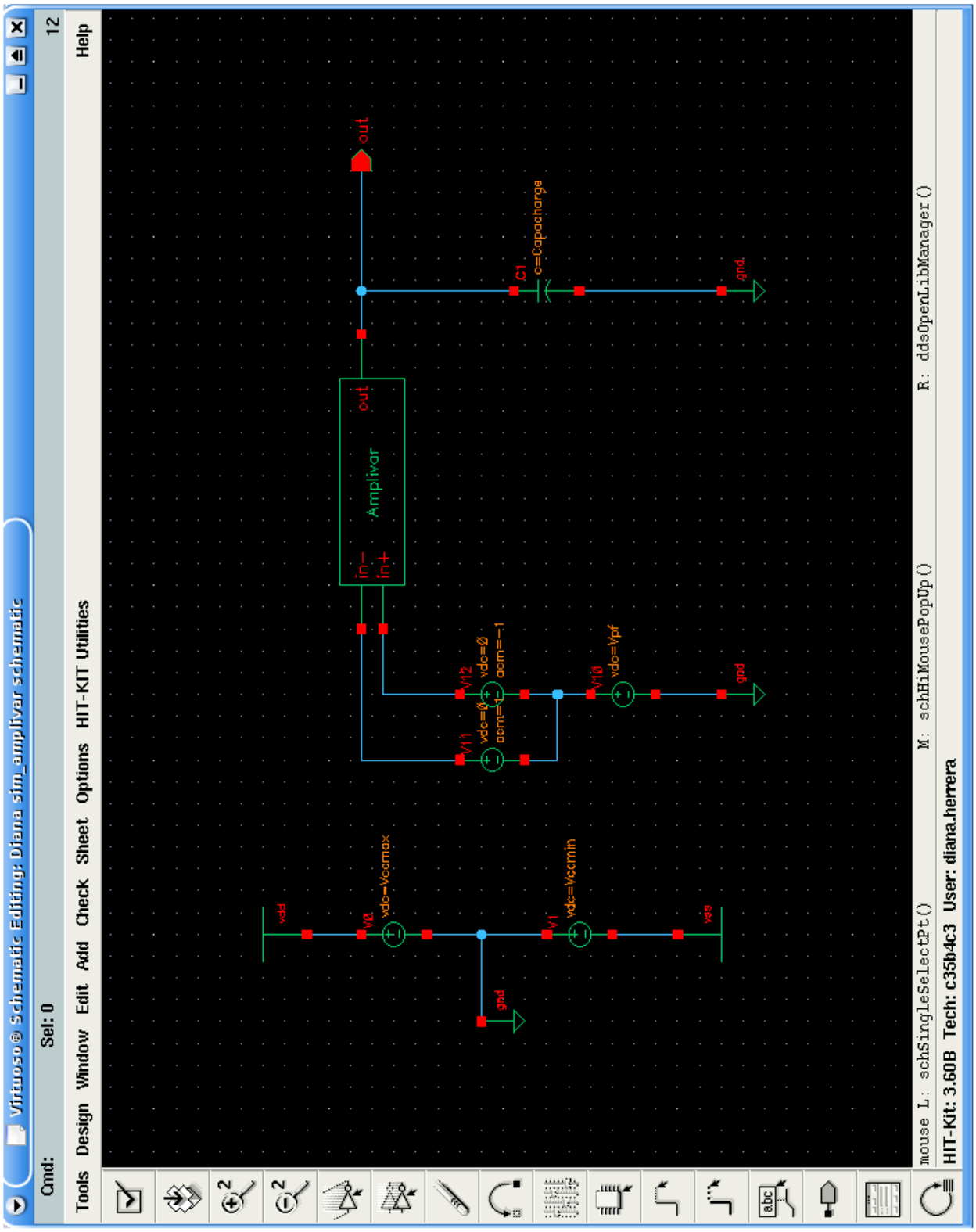


Schéma de simulation du Gain en Mode commun

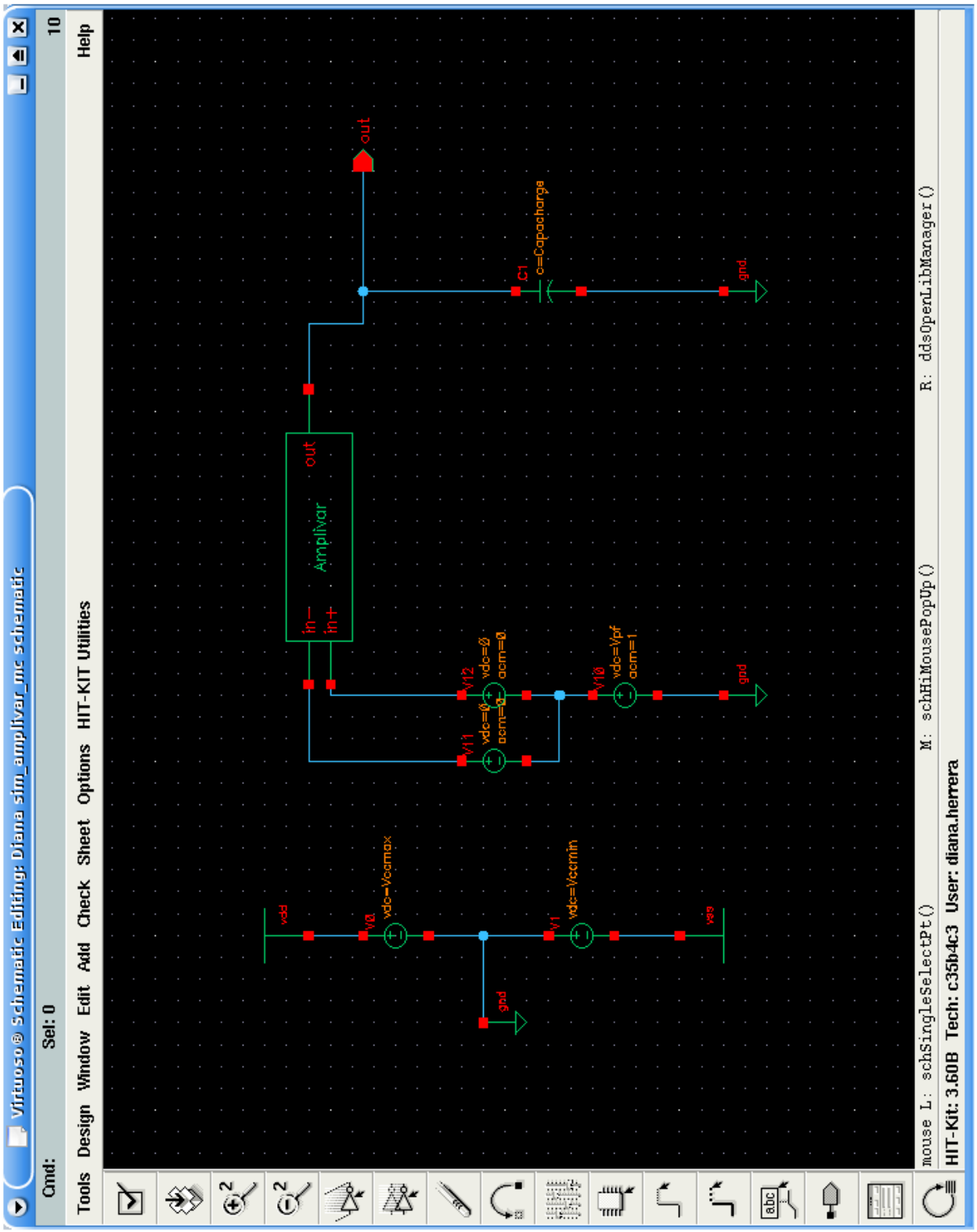
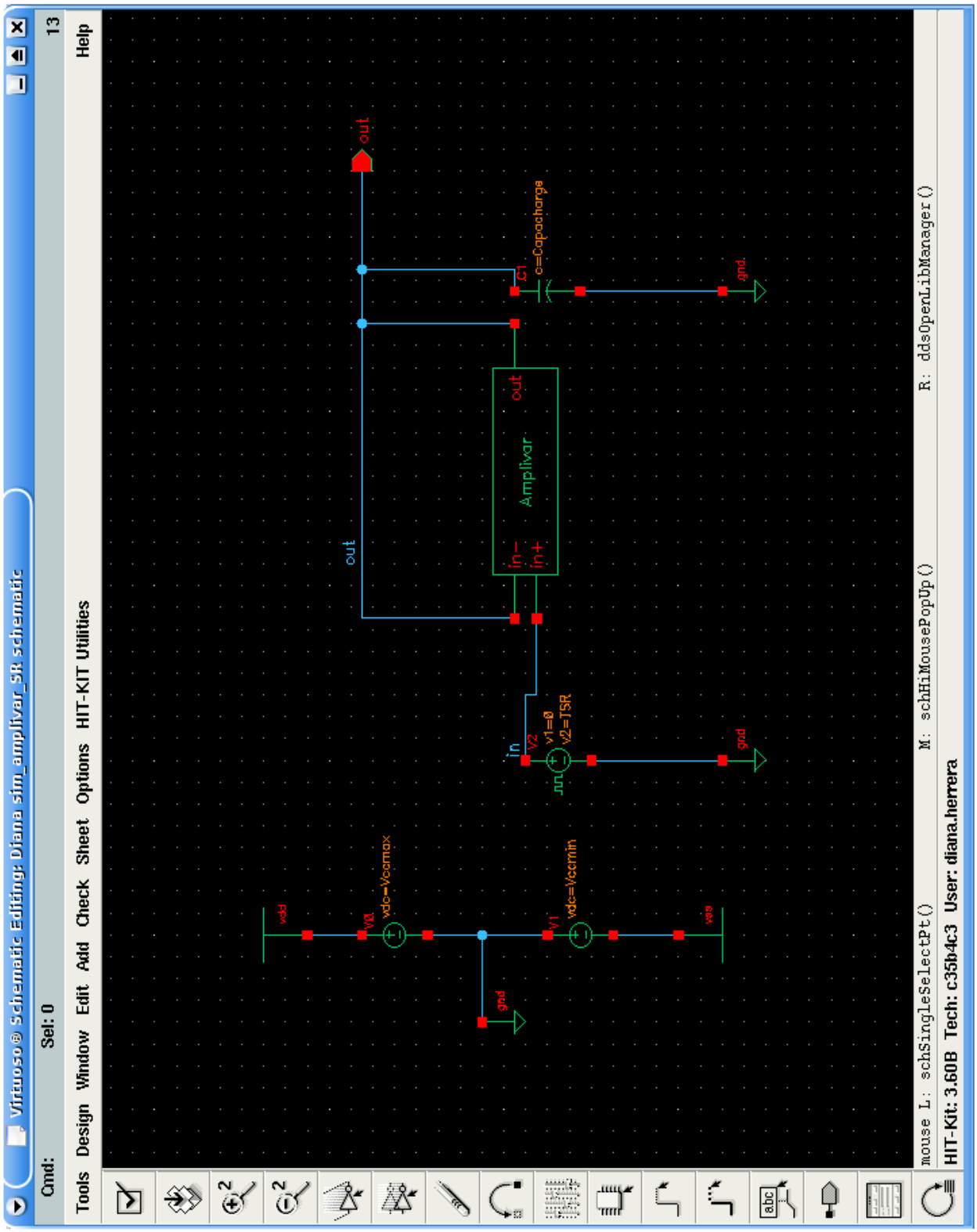


Schéma de simulation du SR



Listings

../Annexes/producteur.cpp 44
../Annexes/consommateur.cpp 46
../Annexes/producteurt.m 49
../Annexes/consommateurt.m 50
../Annexes/producteur.m 52
../Annexes/consommateur.m 54
../Annexes/consommateurp.m 55
../Annexes/consommateur.ocn 58
../Annexes/simuvar_ACmod.ocn 61
../Annexes/producteur.ocn 66
../Annexes/cproducteurp.ocn 69
../Annexes/modnetlist.pl 77

Programmes Visual C++

Programmes de test pour valider le concept de sémaphore et l'échange de données entre deux applications.

producteur.cpp : envoie des données (l'heure et la date) vers un consommateur écrit en Visual C++ ou Matlab

consommateur.cpp : reçoit des données envoyés par un producteur écrit en Visual C++ ou Matlab.

producteur.cpp

```
1 // Producteur.cpp : Defines the entry point for the console application.
  //
3
  #include "stdafx.h"
5 #include <stdio.h>
  #include <stdlib.h>
7 #include <iostream.h>
  #include <time.h>
9
  tm *  heure(void)
11 {
  //Pour savoir l'heure et la date
13 struct tm *newtime;
  time_t aclock;
15 time( &aclock );
  //La fonction local time donne la date et l'heure
17 newtime = localtime( &aclock );
  printf( "Le heure et date sont: %s", asctime( newtime ) );
19 return(newtime);

21 }

23 FILE* enregistrer()
  {
25
        FILE *fp;
27
        tm * heuredate;
29      fp = fopen("H:\\diana\\fichier.txt","a");//Ouvrir le fichier

31      //nous appelons la fonction heure
        heuredate=heure();
33      fputs(asctime(heuredate),fp);//Ecrit le texte dans le fichier

35      //cin.get();
        if (ferror(fp)){ //verifier s'il y a des erreurs
37              printf("Erreur pendant l'ecriture\n");
                clearerr(fp);
39      }
```

```
        fclose(fp);//Fermeture du fichier
41
      //Pour verifier s'il y a des erreurs lors de la fermeture du fichier
43        if(fclose(fp)==0){
                printf("Erreur dans la fermeture du fichier\n");
45
          }
47        return(fp);
          }
49

51 void main(int argc, char* argv[])
   {
53 FILE *fp;

55        do{
          fp=enregistrer();
57
          }
59        while(fp==NULL);
          //cin.get();
61 }
```

consommateur.cpp

```
1 // Consommateur.cpp : Defines the entry point for the console application.
  //
3
  #include "stdafx.h"
5 #include <stdio.h>
  #include <stdlib.h>
7 #include <iostream.h>

9 int main(int argc, char* argv[])
  {
11        FILE *fp=NULL;
          int i=0;
13        char list[30];
          int  dhread;
15        while (1)


17
          //FILE *stream=0;
19


21        //char buffer[81];

23        //Ouverture du fichier en lecture
          fp= fopen("H:\\diana\\fichier.txt","r");
25
          //Si le fichier existe
27        if(fp!=NULL)
          {
29        printf("fichier trouve\n");

31        //Lecture du fichier
          dhread = fread(list, sizeof( char ),50,fp );
33      printf( "Contenu du fichier = %.50s\n", list );

35      //Fermeture du fichier
          fclose(fp);
37
          //while (!ferror(fp) && !feof(fp))
39        //       buffer[i++]=fgetc(fp);
```

```
        //buffer[--i]='\0';
41
        if(ferror(fp))
43      perror("Erreur dans la lecture du fichier\n");

45      if(fclose==NULL){
        printf("Erreur dans la fermeture du fichier\n");
47      }


49
        //Suppression du fichier
51      char filename[] = "H:\\diana\\fichier.txt";
        remove(filename);
53      }

55      //Si le fichier n'existe pas
        else
57      fp = fopen("H:\\diana\\fichier.txt","r");
        printf("recherche fichier\n");
59

61      //return;
}
```

Programmes Matlab

Programmes de test pour valider le concept de sémaphore et l'échange de données entre deux applications.

producteurt.m : envoie des données (l'heure et la date) vers un consommateur écrit en Visual C++ ou Matlab

consommateurt.m : reçoit des données envoyés par un producteur écrit en Visual C++ ou Matlab.

Programmes pour l'échange de données entre Matlab et Cadence

producteur.m : envoie les données issues de l'optimisation vers Cadence.

consommateur.m : reçoit les performances produites par Cadence lors de la simulation.

consommateurp.m : reçoit les paramètres du point de fonctionnement envoyés par Cadence.

producteurt.m

```
% Programme du Producteur

while 1
%Pour tester l'existence du fichier:
pf=exist('H:/diana/fichier.txt','file')

%Si le fichier n'existe pas:
if(pf ==0)

%Pour obtenir la date et l'heure
dateheure=datestr(now)

%Ouvrir le fichier en ecriture
fp= fopen('H:/diana/fichier.txt','w');

%Pour ecrire dans le fichier la date et l'heure obtenues:
fwrite(fp,dateheure,'integer*4');
disp('Fichier créé');

%Fermeture du fichier
stat = fclose('all');

%Pour verifier s'il y a des erreurs lors de la fermeture du fichier
if(stat == -1)
   disp('Erreur dans la fermeture du fichier')
end

pause(1)

%Si le fichier existe:
else
disp('attend que le cons efface le fichier')
pause(1);
end

end
```

consommateurt.m

```
% Programme du Consommateur
while 1

%Pour tester l'existence du fichier:
pc=exist('H:/diana/fichier.txt')

%Si le fichier n'existe pas
if(pc==0)
%pfc=exist('H:/diana/fichier.txt')
disp('attend que le producteur cree le fichier')
pause(1);

%Si le fichier existe
else
disp('fichier trouvé')

%Ouverture du fichier en lecture
fid=fopen('H:/diana/fichier.txt','r');

tline=fgets(fid)

%Pour verifier s'il y a des erreurs lors de la lecture du fichier
if(tline==-1)
    disp('Erreur dans la lecture du fichier')
end

%Fermeture du fichier
status=fclose('all');

%Pour verifier s'il y a des erreurs lors de la fermeture du fichier
if(status== -1)
   disp('Erreur dans la fermeture du fichier')
end

%Pour effacer le fichier
delete('H:/diana/fichier.txt')

pause(1);
```

```
40 end

42 end
```

producteur.m

```
% Programme du Producteur

%while 1

%Test de l'existence du fichier :
pf=exist('H:/fichierP.txt','file');

%Si le fichier n'existe pas:
if(pf ==0)

%Pause d'une minute pour que Cadence puisse finir la simulation et donner
%les resultats (curves, etc)
pause(60)

%Données à envoyer vers Cadence
disp('Données a envoyer ves Cadence')
x;
powersup=5;
resistance=0;
Vpf=2.5;
capacharge=2e-12;

%Ouverture du fichier pour ecrire les données
fp= fopen('H:/fichierP.txt','w');

%Ecriture des données dans le fichier
fprintf(fp,'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d
x, powersup, resistance, Vpf, capacharge)
disp('Fichier créé');
pause(1)

%Fermeture du fichier
stat = fclose('all');

%Pour afficher les erreurs lors de la fermeture du fichier
if(stat == -1)
    disp('Erreur dans la fermeture du fichier')
end
```

```
40 pause(1)

42 %Si le fichier existe :
   else
44 %Message d'attente
   disp('attend que le cons efface le fichier')
46 pause(1)
   end
48
   %end
```

consommateur.m

```
%Programme du Consommateur

%while 1
%Test de l'existence du fichier :
pc=exist('H:\fichiers.txt')

%Si le fichier n'existe pas
if(pc==0)
disp('attend que le producteur cree le fichier')
pause(1)

%Si le fichier existe
else
disp('fichier trouvé')

%Lecture du fichier
SPECC = dlmread('H:\fichiers.txt', '')
%Nous utilisons "dlmread" Pour lire un fichier avec valeurs numeriques
%et créer une matrix avec ses elements.

%Nous assignons à chaque valeur lu la variable qui correspond.
disp('Résultats de la simulation effectué en Cadence:')
GB=SPECC(1,1)
margedephase=SPECC(1,2)
bandepassante=SPECC(1,3)
Gaindiff=SPECC(1,4)
SR=M(1,5)
CMRR=SPECC(1,6)

%Pour effacer le fichier
delete('H:\fichiers.txt')

%pause(1)

end
%end
```

consommateurp.m

```
%Programme du Consommateur

%while 1
%Test de l'existence du fichier:
pc=exist('H:\fichiercp.txt')

%Si le fichier n'existe pas
while pc==0
disp('attend que le producteur cree le fichier')
pc=exist('H:\fichiercp.txt')
%pause(1)
end

%Si le fichier existe
disp('fichier trouvé')

%Lecture du fichier
M = dlmread('H:\fichiercp.txt', '');
%Le fichier "fichiercp.txt" contient une file et 296 columnes
%Pour lire un fichier avec valeurs numeriques et créer une matrix avec
%ses elements
%On peut assigner chaque valeur lu à sa variable correspondante
disp('Parametres')
ids    =M(1,1),M(1,38),M(1,75),M(1,112),M(1,149),M(1,186),M(1,223),M(1,260)
vgs      =M(1,2),M(1,39),M(1,76),M(1,113),M(1,150),M(1,187),M(1,224),M(1,261
vds    =M(1,3),M(1,40),M(1,77),M(1,114),M(1,151),M(1,188),M(1,225),M(1,262)
vbs    =M(1,4),M(1,41),M(1,78),M(1,115),M(1,152),M(1,189),M(1,226),M(1,263)
vth      =M(1,5),M(1,42),M(1,79),M(1,116),M(1,153),M(1,190),M(1,227),M(1,264
vdsat  =M(1,6),M(1,43),M(1,80),M(1,117),M(1,154),M(1,191),M(1,228),M(1,265)
gm     =M(1,7),M(1,44),M(1,81),M(1,118),M(1,155),M(1,192),M(1,229),M(1,266)
gds      =M(1,8),M(1,45),M(1,82),M(1,119),M(1,156),M(1,193),M(1,230),M(1,267
cgd    =M(1,14),M(1,51),M(1,88),M(1,125),M(1,162),M(1,199),M(1,236),M(1,273)
cgs    =M(1,15),M(1,52),M(1,89),M(1,126),M(1,163),M(1,200),M(1,237),M(1,274)
cdb    =M(1,20),M(1,57),M(1,94),M(1,131),M(1,168),M(1,205),M(1,242),M(1,279)
pwr    =M(1,33),M(1,70),M(1,107),M(1,144),M(1,181),M(1,218),M(1,255),M(1,292
region=M(1,35),M(1,72),M(1,109),M(1,146),M(1,183),M(1,220),M(1,257),M(1,294


pause(1)
```

```
40 fclose('all')

42 %Pour effacer le fichier
   delete('H:\fichiercp.txt')
44
   %pause(1)
46


48 %end
```

Programmes Skill

consommateur.ocn : reçoit les variables envoyés pas Matlab, lance le programme qui modifie les netlists (modnetlist.pl) et une fois la netlist est modifiée lance le programme qui réalise les simulations (simuvar_ACmod.ocn).

simuvar_ACmod.ocn : réalise les simulations pour obtenir les performances. Lance le programme « producteur.ocn ».

producteur.ocn : envoie vers Matlab les performances obtenues lors des simulations.

cproducteurp.ocn : reçoit les variables envoyés par Matlab pour obtenir les paramètres du point de fonctionnement. Il réalise la simulation au point de repos et il envoie vers Matlab les paramètres obtenus lors de cette simulation.

consommateur.ocn

```
;n=1
;while(n=1

;Tester l'existence ou la non existence du fichier
if(isFileName("fichierP.txt")==t

;Si le fichier existe
then

        ;Afficher que le fichier existe
        println("Le fichier existe")

        ;Tester si le fichier peut etre lu
        if(isReadable("fichierP.txt")==t

        ;Si le fichier peut etre lu
        then

        ;Afficher le fichier peut etre lu
        println("On peut lire le fichier")

        ;Ouverture du fichier en lecture
        inPort = infile("fichierP.txt")
        fscanf( inPort "%s %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f"
        ampli courant capacite W1 W3 W5a W6 W7a W8 Ldif Lch Lsc Lsor
        powersup resistance Vpf capacharge)

        ;Option pour plus de parametres
        ;fscanf( inPort "%s %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
        ;%f %f %f %f %f %f " ampli courant capacite W1 W2 W3 W4 W5a W6 W7a
        ;Ldif L2 Lch L4 Lsc Lsor L7 L8 powersup resistance Vpf capacharge)

        ;Liste de parametres lus
        param = (list courant capacite W1 W3 W5a W6 W7a W8 Ldif Lch Lsc Lso
        powersup resistance Vpf capacharge)

        ;Nom de l'amplificateur choisi par l'utilisateur
        typeamplificateur = (list amplif)
        println(param)
```

```
40
        printf("\nType amplificateur = %s" amplif)
42      printf("\ncourant = %f" courant)
        printf("\nresistance = %f" resistance)
44      printf("\ncapacite = %f" capacite)
        printf("\nW6 = %f" W6)
46      printf("\nLdif = %f" Ldif)
        printf("\nLch = %f" Lch)
48      printf("\nLsc = %f" Lsc)
        printf("\nLsor = %f" Lsor)
50      printf("\nW3 = %f" W3)
        printf("\nW8 = %f" W8)
52      printf("\nW7a = %f" W7a)
        printf("\nW1 = %f" W1)
54      printf("\nW5a = %f" W5a)
        printf("\npowersup = %f" powersup)
56      printf("\nVpf = %f" Vpf)
        printf("\ncapacharge = %f" capacharge)
58
        ;Fermeture du fichier
60      close( inPort )

62      ;Effacer le fichier
        deleteFile("fichierP.txt")
64
        ;Modifier toutes les netlists d'accord a l'amplificateur choisi
66      ;avant de lancer les simulations

68      cid=ipcBatchProcess(strcat( "~/modnetlist.pl " "/linux-nas/users
        /ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar/spectre/schematic
70      /netlist/netlist "  amplif ) "" "/tmp/netlist.log")

72      cid=ipcBatchProcess(strcat( "~/modnetlist.pl " "/linux-nas/users
        /ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar_mc/spectre/schematic
74      /netlist/netlist "  amplif ) "" "/tmp/netlist_mc.log")

76      cid=ipcBatchProcess(strcat( "~/modnetlist.pl " "/linux-nas/users
        /ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar_SR/spectre/schematic
78      /netlist/netlist "  amplif ) "" "/tmp/netlist_SR.log")

80      ;Chargement du programme pour lancer la simulation.
```

```
        load("simuvar_ACmod.ocn")
82
        ;Si le fichier ne peut pas etre lu
84      else
        ;Afficher que le fichier ne peut pas etre lu
86      println("On ne peut pas lire le fichier")
                )
88
        ;Si le fichier n'existe pas
90      else
                ;Message d'attente
92              println("En attendant que le producteur cree le fichier")
        )
94 ;)
```

simuvar_ACmod.ocn

```
simulator( 'spectre )
global Ipol
design(  "/linux-nas/users/ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar
/spectre/schematic/netlist/netlist")


modelFile(
    '("/cds/local/AMS_3.60_CDS/spectre/c35/mcparams.scs" "")
    '("/cds/local/AMS_3.60_CDS/spectre/c35/cmos53.scs" "cmostm")
    '("/cds/local/AMS_3.60_CDS/spectre/c35/res.scs" "restm")
    '("/cds/local/AMS_3.60_CDS/spectre/c35/cap.scs" "captm")
    '("/cds/local/AMS_3.60_CDS/spectre/c35/bip.scs" "biptm")
    '("/cds/local/AMS_3.60_CDS/spectre/c35/ind.scs" "indtm")
)


;Déclaration des variables du schéma de simulation
;desVar(          "Ipol" courant)
;desVar(          "Res" resistance)
;desVar(          "Capa" capacite)
;desVar(          "Wsor" W6)
;desVar(          "Lsor" Lsor)
;desVar(          "Wch" W3)
;desVar(          "Wsc" W8)
;desVar(          "W7" W7a)
;desVar(          "Wdiff" W1)
;desVar(          "W5" W5a)
;desVar(          "Ldif" Ldif    )
;desVar(          "Lsc" Lsc      )
;desVar(          "Lch" Lch   )
desVar(    "Vccmin" powersup)
desVar(    "Vccmax" powersup)
desVar(    "Vpf" Vpf)
desVar(    "capacharge" capacharge)


analysis('ac ?start "1"  ?stop "1G" ?dec "1500"  )
```

```
analysis('dc ?saveOppoint t ?dev "/V10"  ?param "dc"  ?start "-5" ?stop "5"
)

envOption( 'autoDisplay  nil )

temp( 27 )
run()
awvResetAllWindows() ;réinitialisation des fenêtres graphiques

selectResult( 'ac )
Adm = vm( "/out" )
plot(Adm ?expr '("Adm"))
addSubwindow()

Gdm = dB20(vm( "/out" ))
plot( Gdm ?expr '("Gdm"))
addSubwindow()

margedephase = phaseMargin( v("/out"))
printf( "la marge de phase=%L\n", margedephase )

bandepassante = bandwidth( v("/out") 3 "low")
printf( "la bande passante =%L\n", bandepassante )

GB = gainBwProd( v("/out"))
printf( "le produit gain-bande =%L\n", GB )

Gaindiff = ymax( dB20(vm( "/out" )))
printf( "le Gain (en db) =%L\n", Gaindiff )


selectResult( 'dc )
CMR = getData("/out")
plot(CMR ?expr '("CMR"))
addSubwindow()

results()
selectResults('dcOpInfo)
```

```
;report(?param "region")
;report(?param "vdsat")


design("/linux-nas/users/ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar_mc
/spectre/schematic/netlist/netlist")

;Déclaration des variables du schéma de simulation
desVar(    "Ipol" courant)
desVar(    "Res" resistance)
desVar(    "Capa" capacite)
desVar(    "Wsor" W6)
desVar(    "Lsor" Lsor)
desVar(    "Wch" W3)
desVar(    "Wsc" W8)
desVar(    "W7" W7a)
desVar(    "Wdiff" W1)
desVar(    "W5" W5a)
desVar(    "Ldif" Ldif    )
desVar(    "Lsc" Lsc      )
desVar(    "Lch" Lch   )
desVar(    "Vccmin" powersup)
desVar(    "Vccmax" powersup)
desVar(    "Vpf" Vpf)
desVar(    "capacharge" capacharge)

analysis('ac ?start "1"  ?stop "1G" ?dec "1500" )


temp( 27 )
run()

selectResult( 'ac )
Acm = vm( "/out")
plot( Acm ?expr '("Acm"))
addSubwindow()

Gcm = dB20(vm( "/out" ))
```

```
120 plot( Gcm ?expr '("Gcm"))
    addSubwindow()
122
    plot( db20(Adm / Acm ) )
124 addSubwindow()
    CMRR = ymax(db20(Adm / Acm ))
126 printf( "le CMRR(en db) =%L\n", CMRR )


128
    design( "/linux-nas/users/ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar_SR
130 /spectre/schematic/netlist/netlist")


132
    analysis('tran ?stop "10m"  )
134
    ;Déclaration des variables du schéma de simulation
136 desVar(    "Ipol" courant)
    desVar(    "Res" resistance)
138 desVar(    "Capa" capacite)
    desVar(    "Wsor" W6)
140 desVar(    "Lsor" Lsor)
    desVar(    "Wch" W3)
142 desVar(    "Wsc" W8)
    desVar(    "W7" W7a)
144 desVar(    "Wdiff" W1)
    desVar(    "W5" W5a)
146 desVar(    "Ldif" Ldif    )
    desVar(    "Lsc" Lsc      )
148 desVar(    "Lch" Lch   )
    desVar(    "Vccmin" powersup)
150 desVar(    "Vccmax" powersup)
    desVar(    "Vpf" Vpf)
152 desVar(    "capacharge" capacharge)
    desVar(    "TSR" powersup)
154
    temp( 27 )
156 run()

158 selectResult( 'tran )
    SRi = getData("/in")
160 SRo =getData("/out")
```

```
plot(SRi ?expr '("SRi"))
plot(SRo ?expr '("SRo"))
SR = slewRate(VT("/out") 1 nil 3 nil 10 90)
printf( "le SR(en db) =%L\n", SR )

;Méthode permettant d'afficher les résultats sous forme de liste
L = list( SR CMRR GB Gaindiff margedephase bandepassante)

;myresults = makeTable( "atable" 0 )
;myresults[1] = 'SR
;myresults[2] = 'CMRR
;M = printstruct( myresults)

;Méthode permettant d'afficher les résultats dans un fichier
R = outfile( "/linux-nas/users/ETI/ETI3/diana.herrera/resultat.il" )

fprintf( R "%L\n" L)
fprintf( R  "le produit gain-bande =%L\n", GB )
fprintf( R   "la marge de phase=%L\n", margedephase )
fprintf( R "la bande passante =%L\n", bandepassante  )
fprintf( R  "le Gain (en db) =%L\n", Gaindiff )
fprintf( R   "le SR(en db) =%L\n", SR )
fprintf( R   "le CMRR(en db) =%L\n", CMRR  )

close(R)

load("producteur.ocn")
```

producteur.ocn

```
;Test de l'existence du fichier contenant les resultats
if(isFileName("/linux-nas/users/ETI/ETI3/diana.herrera/resultat.il")==t

;Si le fichier existe
then
        ;Afficher que le message existe
        println("On a trouve les resultats")

        ;Ouvrir le fichier pour le lire
        inPortp = infile("resultat.il")
        fscanf( inPortp "%f %f %f %f %f %f" GB margedephase bandepassante
        Gaindiff SR CMRR)

        ;Liste avec les parametres a retourner a Matlab
        param2 = list(GB margedephase bandepassante Gaindiff SR CMRR)

        ;Pour remplacer les nil par zero
        GB=list(0 GB nil)
        println(GB)
        rplaca(member(nil GB) "nan")
        GB=nth(1 GB)
        println(GB)

        margedephase=list(0 margedephase nil)
        println(margedephase)
        rplaca(member(nil margedephase) "nan")
        margedephase=nth(1 margedephase)
        println(margedephase)

        bandepassante=list(0 bandepassante nil)
        println(bandepassante)
        rplaca(member(nil bandepassante) "nan")
        bandepassante=nth(1 bandepassante)
        println(bandepassante)

        Gaindiff=list(0 Gaindiff nil)
        println(Gaindiff)
        rplaca(member(nil Gaindiff) "nan")
        Gaindiff=nth(1 Gaindiff)
```

```
40          println(Gaindiff)

42          SR=list(0 SR nil)
            println(SR)
44          rplaca(member(nil SR) "nan")
            SR=nth(1 SR)
46          println(SR)

48          CMRR=list(0 CMRR nil)
            println(CMRR)
50          rplaca(member(nil CMRR) "nan")
            CMRR=nth(1 CMRR)
52          println(CMRR)

54          println(GB)
            println(margedephase)
56          println(bandepassante)
            println(Gaindiff)
58          println(SR)
            println(CMRR)
60
            ;Remplacement de la liste param2
62          param2 = list(GB margedephase bandepassante Gaindiff SR CMRR)
            println(param2)
64


66          ;println(param2)
            ;printf("\nGB = %g" GB)
68          ;printf("\nmargedephase = %g" margedephase)
            ;printf("\nbandepassante = %g" bandepassante)
70          ;printf("\nGaindiff = %g" Gaindiff)
            ;printf("\nSR = %g" SR)
72          ;printf("\nCMRR = %g" CMRR)

74          ;Fermer le fichier
            close( inPortp )
76
            ;datav=strcat("GB" "margedephase" "bandepassante" "Gaindiff" "SR"
78          "CMRR")

80          printf("\n0n prepare les resultats pour les envoyer vers Matlab")
```

```
82          ;Création du fichier
            myPortp = outfile("/linux-nas/users/ETI/ETI3/diana.herrera
84          /fichiers.txt")

86          ;Ecriture des données dans le fichier
            ;fprintf( myPortp "%g %g %g %g %g %g" GB margedephase bandepassante
88          Gaindiff SR CMRR )
            ;println( myPortp "%f %s %f %f %s %f" GB margedephase bandepassante
90          Gaindiff SR CMRR )
            ;fprintf( myPortp "%f %s %f %f %f" GB margedephase bandepassante
92          Gaindiff CMRR )

94          ;fprintf( myPortp "%t %t %t %t %t %t" GB margedephase bandepassante
            Gaindiff SR CMRR )
96          ;datav=strcat(GB " " margedephase " " bandepassante " " Gaindiff "
            SR " " CMRR)
98          ;println( param2 myPortp)
            ;fprintf(myPortp datav)
100         ;Ecriture des données dans le fichier

102         ;fprintf( myPortp param2)
            ;Fermeture du fichier
104         close(myPortp)

106         ;Effacer le fichier
            ;deleteFile("resultat.il")
108

110 ;Si le fichier n'existe pas
    else
112
            ;Afficher que le fichier n'existe pas
114         println("Il n'y a pas encore de resultats")

116         ;Message d'attente
            println("En attendant que les resultats soient produits")
118         )
```

cproducteurp.ocn

```
;n=1
;while(n=1

;Tester l'existence ou la non existence du fichier
if(isFileName("fichierP.txt")==t

        then

                ;Afficher que le fichier existe
                println("Le fichier existe")

                ;Tester si le fichier peut etre lu
                if(isReadable("fichierP.txt")==t

                ;Si le fichier peut etre lu
                then

                        ;Afficher que le fichier ne peut pas etre lu
                        println("On peut lire le fichier")

                ;Ouverture du fichier en lecture
                inPort = infile("fichierP.txt")
                fscanf( inPort "%f %f %f %f %f %f %f %f %f %f %f %f %f %f %
                %f %f %f %f %f %f %f" courant capacite W1 W2 W3 W4 W5a W6 W
                W8 Ldif L2 Lch L4 Lsc Lsor L7 L8 powersup resistance Vpf
                capacharge)

                ;Liste de parametres lus
                param = (list courant capacite W1 W2 W3 W4 W5a W6 W7a W8 Lc
                L2 Lch L4 Lsc Lsor L7 L8 powersup resistance Vpf capacharge

                ;Afficher les parametres lus
                println(param)

                ;Fermeture du fichier
                close( inPort )

                ;Effacer le fichier
                deleteFile("fichierP.txt")
```

```
40

42 ;Realisation de la simulation au point de repos

44 simulator( 'spectre )

46 design("/linux-nas/users/ETI/ETI3/diana.herrera/cds36/Sim/sim_amplivar
   /spectre/schematic/netlist/netlist")
48
   resultsDir("/linux-nas/users/ETI/ETI3/diana.herrera/cds36/Sim
50 /sim_amplivar/spectre/schematic")

52 modelFile(
       '("/cds/local/AMS_3.60_CDS/spectre/c35/mcparams.scs" "")
54     '("/cds/local/AMS_3.60_CDS/spectre/c35/cmos53.scs" "cmostm")
       '("/cds/local/AMS_3.60_CDS/spectre/c35/res.scs" "restm")
56     '("/cds/local/AMS_3.60_CDS/spectre/c35/cap.scs" "captm")
       '("/cds/local/AMS_3.60_CDS/spectre/c35/bip.scs" "biptm")
58     '("/cds/local/AMS_3.60_CDS/spectre/c35/ind.scs" "indtm")
   )
60
   ;Déclaration des variables du schéma de simulation
62 desVar(    "Ipol" courant)
   desVar(    "Res" resistance)
64 desVar(    "Capa" capacite)
   desVar(    "Wsor" W6)
66 desVar(    "Lsor" Lsor)
   desVar(    "Wch" W3)
68 desVar(    "Wsc" W8)
   desVar(    "W7" W7a)
70 desVar(    "Wdiff" W1)
   desVar(    "W5" W5a)
72 desVar(    "Ldif" Ldif    )
   desVar(    "Lsc" Lsc      )
74 desVar(    "Lch" Lch   )
   desVar(    "Vccmin" powersup)
76 desVar(    "Vccmax" powersup)
   desVar(    "Vpf" Vpf)
78 desVar(    "capacharge" capacharge)

80 ;analysis('dc ?saveOppoint t ?dev "/V10" ?param "dc" ?start "-5" ?stop "5")
```

```
analysis('dc ?saveOppoint t)
temp(27)
run()

;openResults("/linux-nas/users/ETI/ETI3/diana.herrera/cds36/Sim
;/sim_amplivar/spectre/schematic/psf")
openResults()
results()
selectResult( 'dcOpInfo)

;Recuperation des parametres du point de fonctionnement
for( i 1 8
nu=i
println(i)
comp=concat( "" "/" "I14" "/" "I" nu)
println(comp)
a=substring(comp 1 5)
println(a)
b=substring(comp 6 8)
println(b)
c=strcat(a b)
println(c)

;Enregistrement des parametres de chaque transistor dans un fichier
report(?output "parametres" ?name c)

;Ouverture du fichier en lecture
inPortp = infile("~/cds36/parametres")


;Lecture du fichier
fscanf( inPortp "%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
%s" Type dp Typev composant ids eq1 idsv vgs eq2 vgsv vds eq3 vdsv vbs
eq4 vbsv vth eq5 vthv vdsat eq6 vdsatv gm eq7 gmv gds eq8 gdsv gmbs eq9
gmbsv betaeff eq10 betaeffv cjd eq11 cjdv cjs eq12 cjsv cgg eq13 cggv
cgd eq14 cgdv cgs eq15 cgsv cgb eq16 cgbv cdg eq17 cdgv cdd eq18 cddv cds
```

```
eq19 cdsv cdb eq20 cdbv csg eq21 csgv csd eq22 csdv css eq23 cssv csb eq24 csbv cbg eq25 cbgv cbd eq26 cbdv cbs eq27 cbsv cbb eq28 cbbv ron eq29 ronv id eq30 idv is eq31 isv ibulk eq32 ibulkv pwr eq33 pwrv gmoverid eq34 gmoveridv region eq35 regionv reversed eq36 reversedv type eq37 typev)

;Fermeture du fichier
close( inPortp )

;Ordonner les paramtres dans une liste
para=list(Type dp Typev composant ids eq1 idsv vgs eq2 vgsv vds eq3 vdsv vbs eq4 vbsv vth eq5 vthv vdsat eq6 vdsatv gm eq7 gmv gds eq8 gdsv gmbs eq9 gmbsv betaeff eq10 betaeffv cjd eq11 cjdv cjs eq12 cjsv cgg eq13 cggv cgd eq14 cgdv cgs eq15 cgsv cgb eq16 cgbv cdg eq17 cdgv cdd eq18 cddv cds eq19 cdsv cdb eq20 cdbv csg eq21 csgv csd eq22 csdv css eq23 cssv csb eq24 csbv cbg eq25 cbgv cbd eq26 cbdv cbs eq27 cbsv cbb eq28 cbbv ron eq29 ronv id eq30 idv is eq31 isv ibulk eq32 ibulkv pwr eq33 pwrv gmoverid eq34 gmoveridv region eq35 regionv reversed eq36 reversedv type eq37 typev)

println(para)

d=concat( "param" nu )
println(d)
e=strcat(idsv " " vgsv " " vdsv " " vbsv " " vthv " " vdsatv " " gmv " " gdsv " " gmbsv " " betaeffv " " cjdv " " cjsv " " cggv " " cgdv " " cgsv " " cgbv " " cdgv " " cddv " " cdsv " " cdbv " " csgv " " csdv " " cssv " " csbv " " cbgv " " cbdv " " cbsv " " cbbv " " ronv " " idv " " isv " " ibulkv " " pwrv " " gmoveridv " " regionv " " reversedv " " typev)

println(e)
set( d e )
)

data=strcat(param1 " " param2 " " param3 " " param4 " " param5 " " param6 " " param7 " " param8)
println(data)


;Pour remplacer les symboles des unites
;Deca
rexCompile("da")
data=rexReplace(data "10" 0)
```

```
    ;Yotta
164 rexCompile("Y")
    data=rexReplace(data "e24" 0)
166 ;Zetta
    rexCompile("Z")
168 data=rexReplace(data "e21" 0)
    ;Exa
170 rexCompile("E")
    data=rexReplace(data "e18" 0)
172 ;Tera
    rexCompile("T")
174 data=rexReplace(data "e12" 0)
    ;Giga
176 rexCompile("G")
    data=rexReplace(data "e9" 0)
178 ;Mega
    rexCompile("M")
180 data=rexReplace(data "e6" 0)
    ;Kilo
182 rexCompile("K")
    data=rexReplace(data "e3" 0)
184 rexCompile("k")                    ;Plus utilise
    data=rexReplace(data "e3" 0)
186 ;Hecto
    rexCompile("h")
188 data=rexReplace(data "e2" 0)

190 ;rexCompile("_")
    ;data=rexReplace(data "1" 0)
192 ;Deci
    rexCompile("d")
194 data=rexReplace(data "e-1" 0)
    ;Centi
196 rexCompile("c")
    data=rexReplace(data "e-2" 0)    ;Plus utilise
198 ;rexCompile("%")
    ;data=rexReplace(data "e-2" 0)
200 rexCompile("c")
    data=rexReplace(data "e-2" 0)
202 ;Milli
    rexCompile("m")
```

```
204 data=rexReplace(data "e-3" 0)
    ;Micro
206 rexCompile("u")
    data=rexReplace(data "e-6" 0)
208 ;Nano
    rexCompile("n")
210 data=rexReplace(data "e-9" 0)
    ;Pico
212 rexCompile("p")
    data=rexReplace(data "e-12" 0)
214 ;Femto
    rexCompile("f")
216 data=rexReplace(data "e-15" 0)
    ;Atto
218 rexCompile("a")
    data=rexReplace(data "e-18" 0)
220 ;Zepto
    rexCompile("z")
222 data=rexReplace(data "e-21" 0)
    ;Yocto
224 rexCompile("y")
    data=rexReplace(data "e-24" 0)
226
    println(data)
228
    ;Ouverture du fichier
230 myPortc = outfile("/linux-nas/users/ETI/ETI3/diana.herrera/fichiercp.txt")

232 ;Ecriture des données dans le fichier
    fprintf( myPortc data)
234
    ;Fermeture du fichier
236 close(myPortc)

238         ;Si le fichier ne peut pas etre lu
            else
240                 ;Afficher que le fichier ne peut pas etre lu
                    println("On ne peut pas lire le fichier")
242                 )

244 ;Si le fichier n'existe pas
```

```
    else
246                 ;Message d'attente
                    println("En attendant que le producteur cree le fichier")
248         )
    ;)
```

Programmes Perl

modnetlist.pl : modifie le contenu de la netlist. Il remplace le nom de l'amplificateur utilisé par celui choisi par l'utilisateur.

modnetlist.pl

```
#!/usr/bin/perl -W

# Nous lissons les arguments reçus qui specifient :
# ARGV[0] le chemin de la netlist a modifier
# ARGV[1] le nom de l'amplificateur souhaite
if (@ARGV>0) {

# Nous voulons lire le fichier bloc par bloc
# Pour définir les blocs, nous definissons le separateur
$/ = "// Library name:";

# Ouverture du fichier en lecture
open FILE, "<$ARGV[0]" or
die "On ne peut pas ouvrir le fichier netlist: $!\n";

# Lecture du fichier bloc par bloc
@blocs = <FILE>;

# Fermeture du fichier
close FILE;

# Nous devons modifier seulement le dernier bloc de la netlist
# (la partie d'instanciation des composants), alors:

# Pour compter le nombre de blocs dans la netlist
# print "Nous avons " . scalar(@bloques) . " blocs\n";
$numblocs = scalar(@blocs);

# En perl, les rangements commencent par 0
# Le premier bloc (bloc 0) contient seulement la chaine de caracteres que
# nous utilisons comme separateur de blocs, alors nous devons pas le
# prendre en compte parce qu'il contient pas d'information utile.
# Donc, notre dernier bloc serait, le nombre total de blocs moins 1

# Pour trouver la position du dernier bloc
$a=1;
$arg=$numblocs-$a;

# Pour modifier le contenu du fichier
```

```
# On edit le dernier paragraphe en utilisant le parametre reçu
$blocs[$arg] =~ s/BTS/$ARGV[1]/g;

# Pour afficher le résultat obtenu (La netlist modifiee)
print @blocs;

# Maintenant pour modifier le contenu du fichier et remplacer l'ancienne
# netlist par la netlist modifie
# Nous ouvrons le fichier en effasant son contenu
open FICH,">$ARGV[0]" or
die "On ne peut pas ouvrir le fichier netlist: $!\n";
# Pour reécrire chaque bloc de la nouvelle netlit dans le fichier
foreach my $bloc (@blocs){
print FICH "$bloc";
}

# Fermeture du fichier
close(FICH);

}
# Si nous ne pouvons pas lire les arguments, un message d'erreur s'affiche
# nous sortons du programme
else {
        print "Erreur : 1 argument requis\n";
};
```