Simulation de circuit avec gEDA et SPICE – HOWTO

Stuart BRORSON [1]

18 janvier 2005

[1]Ce HOWTO a été écrit grâce à Electronicscript, inc. sous GNU Free Documentation licence. L'auteur peut être joint à sdb@electronicscript.com. La version la plus récente de ce document peut être trouvée sur http\protect://www.brorson.com/gEDA/HOWTO/. Cette tradution a été réalisée par Iznogood de la www.iznogood-factory.org.

Table des matières

Abstract

Linux deviendra une plate-forme d'ingénieurs de plus en plus populaire dans le futur. Les applications de CAO de qualité professionnelle pour la conception de circuits sont rendus disponibles par des développeurs de la communauté du logiciel libre. Pour l'électronique, la suite gEDA est le jeu d'outils préféré pour la conception de circuits. La simulation de circuits analogiques utilisant SPICE est maintenant aussi disponible sur Linux. Ce HOWTO décrit la méthode de conception employée pour effectuer des simulations SPICE utilisant les outils gEDA sur Linux.

Chapitre 1

Introduction

Le développement moderne est une discipline utilisant intensivement les ordinateurs. Comme les professionels des autres disciplines d'enginerie, les ingénieurs en électricité et les développeurs électroniques sont des utilisateurs importants de tous types de logiciels de CAO, qui incluent les logiciels pour la conception de circuits et la simulation, de même que la production de PCB et de circuits intégrés. Les ingénieurs électriques ont un nom spécial pour les logiciels de CAO utilisés : EDA, ce qui signifie « Electronic Design Automation (Automatisation de Conception Électronique) ». On peut trouver différents types de logiciels de CAO sous cette rubrique. Par exemple, pendant les phases de dessins d'un projet, un ingénieur électrique utilisera un programme appelé « schematic capture (capture de schémas) » pour saisir son dessin dans l'ordinateur. Un programme de capture de schémas est basiquement un programme de dessin spécialisé incorporant des symboles utilisés dans la création de schémas de circuits. Après avoir dessiné ce schéma, l'ingénieur électrique peut choisir de simuler le comportement de son circuit de manière à vérifier que son schéma fonctionnera comme souhaité. Le programme le plus connu pour ce faire est SPICE (Simulation Program with Integrated Circuit Emphasis), qui a été développé par Berkeley dans le début des années 1970 et qui est largement disponible sous de multiples formes de nos jours. SPICE est maintenant considéré comme un outil d'enginerie fondamental et est une partie essentielle dans l'éventail des ingénieurs les plus actifs.

Le projet gEDA (`http://geda.seul.org`) est un effort open-source pour créer une suite GPL d'une EDA fonctionnant sur GNU/Linux. gEDA s'est développé jusqu'au point d'obtenir des outils de haut niveau de puissance et de qualité ; en utilisant la suite gEDA, vous pouvez maintenant créer des netlists SPICE complexes (fichiers) incorporant les fichiers de modèles des fabricants. Vous pouvez alors utiliser divers simulateurs fonctionnant sous Linux pour effectuer des simulations SPICE de vos netlists. L'objectif de ce document est d'expliquer comment utiliser les outils de gEDA (typiquement fonctionnant sous GNU/Linux) pour effectuer des simulations SPICE. En particulier, ce HOWTO documente l'utilisation de `spice-sdb`, qui est une interface avancée pour le netlister de gEDA (`gnetlist`) utilisé pour créer des netlists SPICE. `Spice-sdb` est fourni avec la suite d'outils de gEDA ; si vous avez installé gEDA, vous êtes déjà prêts à créer des netlists SPICE. Ce HOWTO fournit aussi des conseils sur l'utilisation de ngspice/tclspice et/ou LTSpice pour simuler un circuit netlisté avec `spice-sdb`.

1.0.1 Audience ciblée pour ce HOWTO

Ce HOWTO n'est pas un tutoriel sur la conception de circuits ou sur la simulation avec SPICE. Il est plutôt conçu pour aider l'ingénieur utilisant gEDA à effectuer des simulations SPICE sur la plate-forme Linux. Je suppose donc que vous êtes déjà familier avec la conceptions électronique, les mécanismes de de capture de schémas en utilisant des outils EDA et la simulation SPICE en général. Je suppose aussi que vous êtes raisonnablement familiers avec le système d'exploitation GNU/Linux et son environnement de développement. Finalement, je suppose que vous avez déjà installé gEDA et que vous savez comment l'utiliser. Si vous avez besoin de vous mettre à niveau rapidement avec un de ces sujets, des informations complémentaires sont disponibles sur les sites web suivants :

- Le projet gEDA : http://www.geda.seul.org/ ;
- La syntaxe et les commandes pour SPICE3 : http://newton.ex.ac.uk/teaching/CDHW/Electronics2/userguide/ ;
- Ngspice : http://ngspice.sourceforge.net/ ;
- Tclspice : http://tclspice.sourceforge.net/ ;
- LTSpice : http://www.linear.com/software/ ;
- Spice sur Linux : http://www.brorson.com/gEDA/SPICE/ ;
- Free Dog – Le Groupe des Utilisateurs Free EDA : http://www.freeedaug.org/.

1.0.2 Remerciements

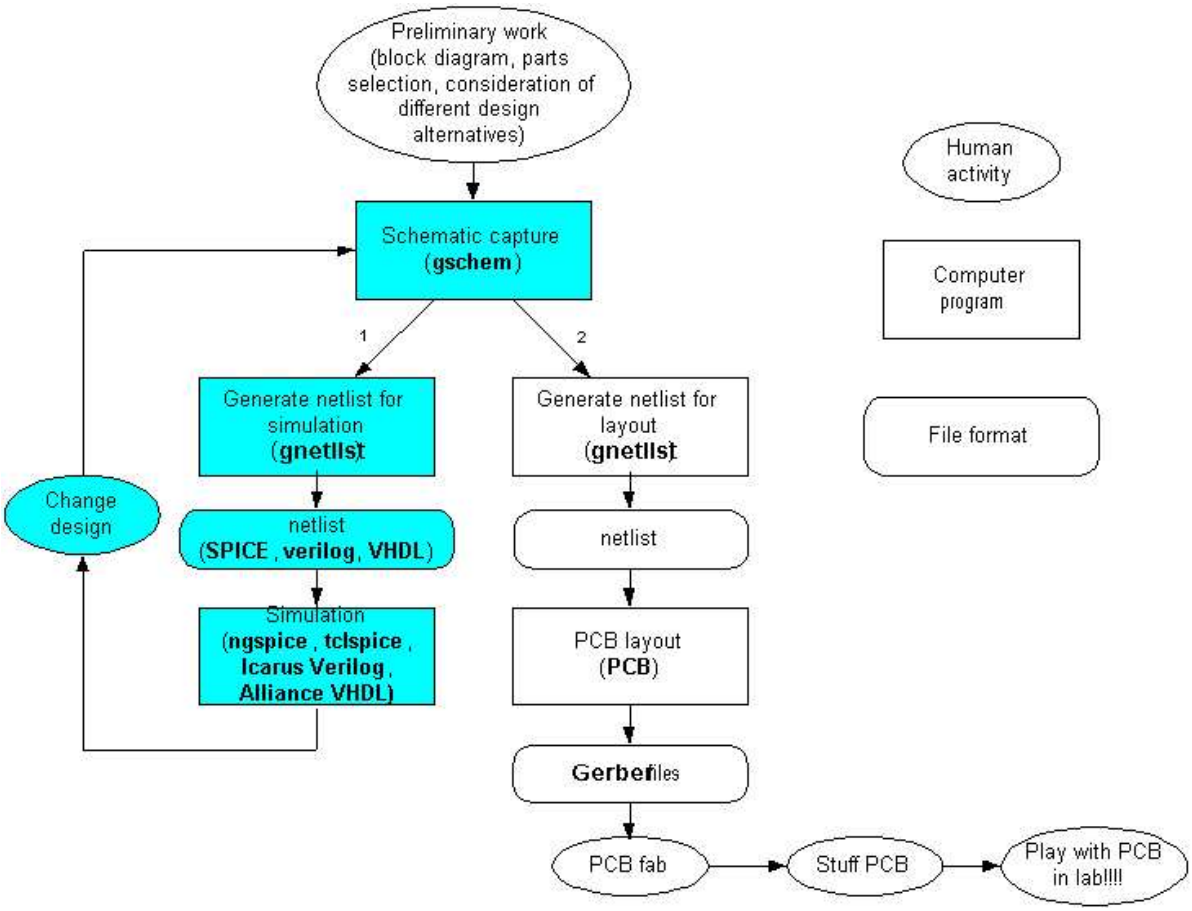
Ce document n'est pas isolé. Plusieurs membres actifs de la communauté EDA libre ont oeuvré pour m'aider à créer ce HOWTO. D'abord et en premier, Paolo Nenzi, l'auteur de ngspice, a pris mon HOWTO original et l'a transformé en document Lyx qui peut ensuite réaliser un DocBook. Merci, Paolo, pour l'aide sur ce HOWTO et, plus important, merci pour le gros travail sur ngspice ! Il y a aussi Ales Hvezda en haut de la liste pour être le moteur derrière le projet gEDA. Sans Ales, rien de tout ceci n'aurait été possible ; sa contribution à gschem est incalculable. Merci, Ales, pour la création de gEDA et de l'avoir distribué sous GPL – vous avez démarré une révolution ! Stefan Jones mérite un grand merci pour son travail sur tclspice, son support gracieux et les efforts d'intégration lorsque j'ai soumis des patches au projet tclspice. Je voudrais aussi remercier W. Kazubski et S. Gieltjes – ils ont écrit les netlisters SPICE originaux sur lesquels j'ai basé gnet-spice-sdb.scm. Je veux aussi remercier Ken Healy pour avoir contribué au patch de tri du netlist et Peter Kaiser pour m'avoir poussé à inclure quelques fonctionnalités utiles pour la simulation de circuits. Je voudrais finalement reconnaître les contributions et les suggestions que j'ai reçu de lecteurs de la liste des utilisateurs de gEDA. La beauté des logiciels libres est qu'il encourage la collaboration, ce qui signifie que le produit final est meilleur que ce qu'un individu pourrait réaliser seul.

Chapitre 2

La grande image : le chemin de la conception dans gEDA.

Dans EDA, le concept de « chemin de la conception » est important. gEDA est une suite d'outils utilisés pour faire de la conception électronique – ce n'est pas une simple application. « Chemin de conception » se réfère à l'ordre dans lequel vous utilisez les outils pour atteindre votre but. Selon si vous faite de la conception analogique ou numérique, de la conception de cartes ou de puces, du type de fichiers nécessaires par le fabricant de vos cartes et un nombre d'autres facteurs, vous utiliserez différents outils de la suite gEDA pour réaliser votre objectif.

Le chemin de conception de gEDA est montré sur l'image ci-dessous. Le diagramme montre le cheminement simple adapté à la conception, la simulation et la fabrication de PCB. Comme on peut le voir, l'activité de simulation (avec les blocs bleus) est une boucle. Cela signifie que vous créez votre schéma et vous le simulez jusqu'à ce qu'il se comporte normalement en fonction des spécifications souhaitées.



Le chemin de conception est utilisé dans gEDA. Sous « simulation », vous avez différents types de simulateurs disponible. Dans ce HOWTO, nous nous intéressons seulement dans les variantes de SPICE (e.g. ngspice, tclspice).

2.1 Survol de l'utilisation de SPICE avec gEDA

Conceptuellement, la simulation de SPICE dans gEDA fonctionne suivant les étapes suivantes :

1. Création et rassemblement des symboles de schémas et des fichiers de modèles SPICE. Les fichiers de modèles SPICE sont souvent obtenus des fabricants de composants. Vous pouvez généralement trouver la plupart des modèles en allant sur le site web du fabricant de composants.
2. Capture de schéma en utilisant les symboles et les modèles SPICE créés dans la première étape.
3. La génération de netlist depuis le schéma créé dans la seconde étape.
4. Simulation SPICE des circuits décrits par les netlists créés dans la troisième étape.

Ces étapes sont illustrées par les boîtes bleues dans le diagramme de flux ci-dessus.

Pour créer une netlist SPICE, le netlister (`gnetlist`) réalise des itérations dans tous les schémas et cherche toutes les parts d'un symbole de composant pour créer une partie du code de SPICE. En général, chaque composant peut générer un ou plusieurs lignes de code SPICE. L'information de composant nécessaire par le netlister est disponible en deux endroits :

1. Le symbole lui-même avec l'attribut `device`, qui est attaché lors de la création du symbole et peut être montré typiquement par l'éditeur de symbole.
2. Dans les attributs attachés manuellement au composant pendant la capture de schéma en utilisant `gschem`.

Comme il existe deux endroit où le netlister recherche l'information, *vous devez vous assurer que l'information nécessaire est disponible dans les deux endroits.*

2.2 Résumé détaillé du flux dessin/simulation

Les étapes détaillées nécessaires pour dessiner et simuler un circuit utilisant gEDA ressemble à ceci :

1. Création de symbole de schéma avec l'attribut `device` correct. (Habituellement, les symboles ont déjà été créés avec l'attribut `device` mais si vous avez des problèmes, il est simple de les contrôler.)
2. Capture de schémas en utilisant `gschem`.
3. Assignement des attributs SPICE (`value`, `model`, `file`, `type`, etc.) aux composants utilisant `gschem` ou `gattrib`.
4. Assignement de `refdes` en utilisant i.e. `refdes_renum`.
5. Création de netlist en utilisant : `gnetlist -g spice-sdb`.
6. Contrôle de netlist pour les corrections (ouverture manuelle et inspection de la netlist).
7. Lancer spice en utilisant un simulateur tel que `LTSpice`, `ngspice` ou `tclspice`.
8. Tracé/analyse des résultats (les outils de tracé/analyse sont souvent incorporés dans le simulateur).
9. Si vous n'êtes pas content des performances de votre circuit telles que révélées par la simulation, retournez à la seconde étape, faites les modifications nécessaires en utilisant `gschem` et recommencez.

L'objectif de cet HOWTO est de fournir la compréhension détaillée nécessaire pour naviguer avec succès au travers de ce processus.

Chapitre 3

Travail préliminaire : préparer les symboles et fichiers SPICE.

Lorsque vous créez des symboles de schémas pour les y inclure dans votre dessin, vous devez vous assurer que certains attributs inclus sont correctement configurés. Les étapes soulignées ci-dessous sont réalisées en éditant le symbole lui-même avec l'éditeur de symboles de `gschem` ou en éditant le fichier de symbole avec un éditeur de texte.

3.1 Configurer vos symboles.

3.1.1 Identifier le composant pour le netlister

Le netlister SPICE peut reconnaître tout symbole particulier de deux manières :

1. L'attribut `device` de symbole et
2. Le `refdes` du symbole.

Ces deux attributs sont attachés au symbole lorsqu'il est créé.

Chaque symbole a l'attribut `device` rattaché. L'attribut `device` est la première chose que le netlister examine lors du traitement d'un symbole. Il existe un certain nombre de composants qui sont natifs pour le netlister, signifiant que ce dernier sait exactement comment traiter avec ce type de composants. Les types natifs de composants incluent **RESISTOR**, **CAPACITOR**, **NPN_TRANSISTOR**, etc. La liste complète des composants natifs est présent dans l'annexe A.

L'attribut `device` est caché pendant l'utilisation normale de `gschem`. Plus souvent, le créateur de symboles a déjà donné au symbole l'attribut `device` correct. Néanmoins, étant donné que l'attribut `device` est caché pour l'utilisateur normal, il peut quelque fois y avoir des problèmes avec la création de netlist SPICE lorsqu'il est positionné à une valeur inattendue. Pour voir l'attribut `device` , allez dans l'éditeur de symbole (sélectionnez le symbole à éditer et faites un *Hierarchy* ▷ *down symbol*) et rendez visible les attributs invisible (*Edit* ▷

show/hide inv text). Si l'attribut `device` est incorrect, vous pouvez le changer en éditant le symbole lui-même en utilisant un éditeur de texte.

Si un symbole n'est pas natif (i.e. le netlister ne le reconnaît pas comme un type interne), le netlister s'appuie sur la première lettre de `refdes` pour déterminer comment traiter le symbole. Le préfixe `refdes` est alors construit dans le symbole lorsqu'il est créé. Comme préfixes d'exemples `refdes`, il existe **R** pour les résistances, **C** pour les condensateurs, **Q** pour les transistors, etc. les préfixes `refdes` corrects pour SPICE sont listés dans l'annexe A. Notez que se baser sur `refdes` pour identifier les composants pour SPICE n'est pas totalement sûr – par exemple, le netlister ne peut pas distinguer entre les transistors NPN et PNP basés sur `refdes`. C'est la raison pour laquelle il est toujours mieux d'utiliser les `device` natifs dans vos symboles.

3.1.2 Initialiser l'ordre des broches

Le netlister génère les broches de composants dans l'ordre initialisé par l'attribut `pinseq`. Notez que ce n'est pas le même que l'emplacement physique des broches. Pour initialiser l'attribut `pinseq`, déterminez d'abord l'ordre des broches comme vous le souhaitez. SPICE utilise un ordre de broche spécifique pour plusieurs composants, incluant les diodes et les transistors. Par exemple, les broches d'un transistor bipolaire sont listées dans l'ordre CBE. Autre exemple : si votre symbole est destiné à représenter un CI modélisé avec un `.subckt` de fabricant, l'ordre des connexions dans le sous-circuit est initialisé par la ligne `.subckt` dans le fichier.

Une fois que vous connaissez l'ordre dans lequel vous émettez les broches, initialisez simplement l'attribut `pinseq` avec l'ordre correct pour le composant. Cela assurera que les broches du composant sont émises dans l'ordre correct.

3.2 Configurer vos fichiers SPICE.

Les fichiers contenant des modèles SPICE compliqués ou d'autre code SPICE peuvent être incorporés dans la netlist SPICE finale en incluant les symboles appropriés dans le schéma. Les fichiers de modèle SPICE sont habituellement obtenus des fabricants de composants. Le traitement avec ces fichiers est direct. Quelques éléments doivent être néanmoins gardés en mémoire lors de la préparation de modèles à utiliser pendant la capture de schémas :

- Il est toujours prudent de placer ces fichiers dans un répertoire dédié distinct du répertoire des symboles.
- *Assurez-vous que les fichiers d'assignations de fichiers SPICE correspondent correctement aux broches comme défini dans le symbole du composant !* Il est difficile d'insister plus. L'ordre dans lequel les broches sont listées dans un fichier .subckt ne correspond pas nécessairement à l'ordre physique des broches du composant. Comme décrit ci-dessus, les broches sont émises par le netlister dans l'ordre donné par l'attribut `pinseq`.
- *Assurez-vous que le dernier caractère dans un fichier de modèle SPICE est un retour de chariot.* S'il n'existe pas de retour de chariot alors le composant suivant listé dans la netlist peut être placé sur la même ligne que la dernière ligne du fichier modèle de SPICE.

Chapitre 4

Créer votre circuit : la capture de schéma

La capture de schémas est le processus par lequel on utilise un programme de dessin spécialisé pour créer un diagramme de schéma du circuit en cours de développement. dans l'environnement de gEDA, le programme de capture de schéma est appelé `gschem`. Je suppose que vous savez déjà comment utiliser `gschem`. Si ce n'est pas le cas, consultez la documentation disponible sur le site web de gEDA : http://www.geda.seul.org/ (vous pouvez aussi trouver des traductions normalement à jour sur http://www.iznogood-factory.org/pub/manuels.html). Dans le but de créer des netlists SPICE, vous devez utiliser `gschem` pour attacher les attributs aux composants et vous aurez la possibilité d'incorporer aussi d'autres directives SPICE dans votre netlist. Après avoir réalisé la capture de schéma, vous pouvez créer la netlist de SPICE en lançant le netlister de gEDA, `gnetlist` sur votre dessin.

4.1 Attributs `Gschem` pour la netlist de spice

Il existe plusieurs manières d'associer des attributs spice avec des composants en utilisant `gschem`. La manière que vous choisissez dépend de plusieurs facteurs, incluant le type de composant, la taille et le format du modèle SPICE.

4.2 Attributs de composant et interprétations

Les attributs suivants sont les plus significatifs pour le netlisting SPICE et peuvent être attachés depuis gschem :

- `refdes` : Le désignateur de référence du composant. Les valeurs valides dépendent du type de composant et sont données dans l'annexe A.
- `value` : Pour les passifs, c'est la valeur du composant. Pour les actifs, c'est le type (no. de modèle) du composant (i.e. 2N3904, uA741). Lorsqu'un modèle pour un actif est instantié séparément pour le composant lui-même, l'attribut de `value` contient le nom du modèle spice.
- `model` : Il contient le modèle spice pour le composant en une seule ligne.

- `file` : Il contient le nom d'un fichier. C'est typiquement un fichier contenant i.e. un .MODEL, .SUBCKT SPICE ou d'autres codes SPICE.
- `model-name` : Contient le nom du modèle spice référé dans un .MODEL ou une définition .SUBCKT. `model-name` est principalement utilisée pour identifier le nom de modèle spice dans le symbole `spice-model-1.sym`. Les composants actifs devraient appeler ce nom dans l'attribut `device` pour associer le composant avec ce modèle ou sous-circuit spice particulier.
- `type` : Il spécifie le type de composant et est utilisé par spice lors de l'interprétation des paramètres du modèle. Les valeurs valides dépendent du composant devant être modélisé.

4.3 Conventions `refdes`

Comme pré-requis pour traiter les attributs liés à SPICE, le netlister SPICE nécessite que tous les composants aient un `refdes` attaché. Le `refdes` peut être attaché soit à la main (ce qui est laborieux) ou en utilisant le programme `refdes_renum` inclus dans la distribution gEDA.

Notez que la première lettre de `refdes` doit correspondre à la lettre appropriée pour la simulation spice. La convention `refdes` est donnée dans l'annexe A.

4.4 Passifs

4.4.1 Passifs de base

Les composants les plus basiques que l'on peut rencontrer dans SPICE sont des composants passifs comme des résistances et des condensateurs qui possèdent des valeurs numériques mais pas d'autres attributs de modèles. Dans ce cas, les attributs suivants peuvent être remplis dans :

- `refdes` : Le `refdes` correct pour le composant.
- `value` : Pour les passifs, c'est la valeur numérique du composant (i.e. 100pF). Pour les actifs, cet attribut peut être rempli mais si aucun attribut de modèle n'est disponible ailleurs dans le schéma, la valeur n'est pas utilisée (dans le netlisting SPICE, en tout cas).

Si seuls les attributs `refdes` et `value` sont rencontrés, le netlister écrira une seule ligne dans le fichier de sortie.

4.4.2 Résistance d'exemple :

- refdes = R2
- value = 220

La ligne SPICE générée : R2 0 4 220

(notez que « 0 » et « 4 » correspondent au noeuds du net pour le composant et sont générés automatiquement par `gnetlist`.)

4.4.3 Condensateur d'exemple :

- refdes = C22
- value = 1UF

La ligne SPICE générée : C22 4 23 1UF

4.5 Passifs avec attributs additionnels

Souvent, les composants passifs ont des attributs additionnels qui leur sont attachés pour les simulations spice. Les exemples de tels attributs sont les coefficients de température (pour les résistances) et les conditions initiales (pour les composants réactifs). Ces composants additionnels peuvent être incorporés dans le fichier SPICE en les attachant simplement à l'attribut `model` du composant. Les attributs nécessaires sont spécifiquement :

- `refdes` : `refdes` de composant correct ;
- `value` : Valeur de composant numérique, comme toujours ;
- `model` : Une chaîne d'une ligne contenant des paramètres additionnels, formatés comme une chaîne SPICE valide.

Cette chaîne est placée après la valeur du composant dans le ligne générée par gnetlist. Il est donc important de formater la chaîne placée dans la ligne `model` devant être du code SPICE valide. Sinon, vous risquerez de faire cafouiller le simulateur SPICE.

4.5.1 Résistance d'exemple :

- refdes = R5
- value = 1MEG
- model = TC=0.001,0.015

Ligne SPICE générée : R2 0 4 220 TC=0.001,0.015

4.6 Passifs pour la conception de semiconducteurs

Les valeurs pour les résistances et les condensateurs sont souvent données comme des dimensions dans les canceptions d'ASIC. SPICE a pris des bibliothèques technologiques la valeur typique par carré et calcule la valeur réelle en Ohm ou Farad d'elle-même. Les attributs suivants sont donc nécessaires :

- refdes : le refdes correct pour le composant
- model-name : correspond au modèle dans la bibliothèque tehnologique
- w, l : les dimensions du composant

La bibliothèque technologique peut être inclus dans une ligne .include dans le fichier d'entrée SPICE.

4.6.1 Exemple de résistance de semiconducteur

- refdes = R6
- model-name = rpoly
- w = 3u
- l = 100u

Ligne SPICE générée : R6 0 5 rpoly w=3 l=100u

Exemple de modèle de résistance de semiconducteur :

- model rpoly R rsh=300

Cela devrait faire partie de la bibliothèque technologique du fabricant d'ASIC.

4.7 Transistors et diodes

Les transistors et les diodes sont généralement accompagnées par un modèle spécifique du composant. Chaque modèle tente de saisir les éléments dynamiques non linéaires détaillés particuliers du composant ; sinon, la simulation SPICE est sans objet. Le modèle SPICE peut être soit une ligne unique, à paramètres courts ou un jeu de paramètres SPICE sur plusieurs lignes. Une chaîne de paramètres typique d'une ligne est une courte liste de paramètres décrivant une diode petits signaux. Les modèles multi-lignes typiques proviennent des fabricants de composants, qui fournissent souvent les modèles pour leurs composants dans un fichier texte. Comme il existe deux grands formats d'informations pour SPICE, il y a deux approches pour incorporer ces paramètres dans le schéma :

4.7.1 Chaîne de paramètres SPICE à ligne unique

Pour incorporer une chaîne de paramètres SPICE à ligne unique dans la netlist, les attributs suivants doivent être attachés au composant :

- `refdes` : `refdes` de composant correct ;
- `value` : Le numéro de modèle ou le numéro de part du composant ;
- `model-name` : Le nom que vous voulez donner au modèle SPICE. C'est habituellement le numéro de modèle ou le numéro de part du composant. Si vous avez déjà attaché un attribut `value` au composant, ce paramètre est optionnel.
- `model` : Chaîne à ligne unique qui contient des paramètres additionnels. Ne placez pas les paramètres du modèle entre parenthèses – `gnetlist` le fera pour vous.

4.7.2 Diode d'exemple :

- refdes = D5
- model-name = 1N1004
- model = IS=0.5UA RS=6 BV=5.20

Ligne SPICE générée :

D5 2 6 1N1004 MODEL 1N1004 D (IS=0.5UA RS=6 BV=5.20)

4.7.3 Fichier de modèle SPICE

Pour incorporer un fichier rempli de paramètres SPICE dans la netlist, les attributs suivants doivent être attachés au composant :

- `refdes` : `refdes` de composant correct ;
- `value` : Le numéro de modèle ou le numéro de part du composant ;
- `model-name` : Le nom que vous voulez donner au modèle SPICE. C'est habituellement le numéro de modèle ou le numéro de part du composant. Si vous avez déjà attaché un attribut `value` au composant, ce paramètre est optionnel.
- `file` : Le nom de fichier de modèle SPICE que vous souhaitez incorporer dans la netlist. Ce nom de fichier peut être soit un chemin relatif, soit absolu mais il est probablement mieux d'utiliser un chemin absolu pour éviter les problèmes si jamais vous déplacez le répertoire de votre schéma.

Notez que vous avez besoin de vos assurer que le nom de modèle de votre fichier de modèle SPICE est le même que les attributs `value` ou `model-name` que vous avez attaché au composant. Il est aussi une bonne idée de vérifier que l'assignement des broches dans le fichier de modèles correspond aux assignements de broches faits par le symbole du composant.

4.8 Actifs – circuits intégrés

Les circuits intégrés sont incorporés dans la netlist similairement aux transistors et aux diodes. Comme tel, vous pouvez incorporer l'information spice soit comme chaîne de paramètre sur une ligne unique ou comme un fichier modèle.

4.8.1 Les paramètres SPICE à ligne unique

Pour incorporer des paramètres SPICE sur une ligne unique dans la netlist, les attributs suivants doivent être attachés au composant :

- `refdes` : `refdes` de composant correct ;
- `value` : Le numéro de modèle ou le numéro de part du composant ;
- `model-name` : Le nom que vous voulez donner au modèle SPICE. C'est habituellement le numéro de modèle ou le numéro de part du composant. Si vous déjà attaché un attribut `value` au composant, ce paramètre est optionnel.
- `model` : Chaîne à ligne unique qui contient des paramètres additionnels. Ne placez pas les paramètres du modèle entre parenthèses – `gnetlist` le fera pour vous.

4.8.2 Fichier .MODEL ou .SUBCKT SPICE

Pour incorporer des paramètres SPICE sur une ligne unique dans la netlist, les attributs suivants doivent être attachés au composant :

- `refdes` : Composant refdes correct. *Notez que si le fichier contient un .MODEL, le* `refdes` *ne doit pas démarrer avec U ; si le fichier contient un .SUBCKT, le refdes doit démarrer avec X.* Le netlister contrôle le type de fichier et tente de « faire la bonne chose » mais les problèmes peuvent survenir si vous ne suivez pas cette règle.
- `value` : Le numéro de modèle ou le numéro de part du composant ;
- `model-name` : Le nom que vous voulez donner au modèle SPICE. C'est habituellement le numéro de modèle ou le numéro de part du composant. Si vous avez déjà attaché un attribut `value` au composant, ce paramètre est optionnel.
- `file` : Le nom du fichier contenant les .MODEL ou .SUBCKT de SPICE que vous souhaitez intégrer dans la netlist. Ce nom de fichier peut spécifier soit un chemin relatif soit absolu mais il est sûrement mieux d'utiliser un chemin absolu pour éviter les problèmes si vous déplacez le répertoire des schémas.

Notez que vous avez besoin de vous assurer que le nom de modèle du fichier modèle de SPICE est le même que celui des l'attributs `value` ou `model-name` que vous avez attaché au composant. Il est aussi une bonne idée de vérifier

que l'assignation de broche dans le fichier modèle correspond l'assignation de broches faite pas le symbole du composant.

4.9 Sources indépendantes

Il existe deux sources indépendantes : les sources de tension et les sources de courant. Pour incorporer une netlist SPICE, elles fonctionnent de la même manière. Pour incorporer une source indépendante dans votre netlist SPICE, faites ce qui suit :

1. Placez la source indépendante sur votre schéma. (Faites *Add* ▷ *Component* ▷ *spice* ▷ *<independent source name>.sym*)
2. Double cliquez sur le bloc et ajoutez/éditez les attributs suivants :
   - `refdes` : V ? ou I ?
   - `value` : Une chaîne à ligne unique au format SPICE décrivant la source.

4.10 Sources dépendantes

Il existe quatre sources dépendantes :
This section remains TBD.

4.11 Composants SPICE

4.11.1 Bloc de modèle Spice

Dans certaines situations, vous pouvez souhaiter embarquer un bloc de modèle spice directement dans votre schéma. C'est fait lorsque vous avez plusieurs composants avec un attribut "value" appelant un modèle spice. Selon si le bloc spice est à une ligne ou à plusieurs, vous pouvez embarquer le code d'une des deux manières suivantes :

4.11.2 Le modèle SPICE à ligne unique :

1. Placez un bloc de modèle spice sur votre schéma. (Faites *Add*▷ *Component*▷ *spice*▷ *spice-model-1.sym*)
2. Double cliquez sur le bloc et ajoutez/éditez les attributs suivants :
   - `refdes` : A ?
   - `model` : nom du modèle (i.e. le nom du modèle utilisé dans les composants étant modélisés.)
   - `type` : Un des types de composants spice valide est défini dans les spec. spice.
   - `value` : Le modèle spice à ligne unique correspondant.

4.11.3 Le modèle SPICE multiligne :

1. Placez un bloc de modèle spice sur votre schéma. (Faites *Add*▷ *Component*▷ *spice*▷ *spice-model-1.sym*)
2. Double cliquez sur le bloc et ajoutez/éditez les attributs suivants :

- `refdes` : A ?
- `model-name` : nom de modèle
- `file` : Le nom de fichier contenant le code modèle SPICE (i.e. .MODEL ou .SUBCKT).

4.11.4 Bloc Include

Le bloc include place une directive .INCLUDE dans votre netlist.

1. Place un bloc de modèle spice sur votre schéma. (Faites *Add*▷ *Component*▷ *spice*▷ *spice-include-1.sym*)
2. Double cliquez sur le bloc et ajoutez/éditez les attributs suivants :
   - `refdes` : A ?
   - `file` : Le nom du fichier à inclure.

4.11.5 Bloc de directive SPICE

Placer un bloc de directives SPICE dans votre schéma crée un bloc arbitraire de code SPICE dans la netlist. La directive peut être soit une définition contenue dans un fichier, soit une chaîne à ligne unique contenue dans l'attribut `model`. Le netlister fera simplement un vidage du contenu de la chaîne ou du fichier dans votre version de netlist. Voici un exemple de situations où les include sont utiles :

- Définition de .TEMP
- Définition de .IC
- Autres définition SPICE pour lequel `gschem` ne possède pas de symbole.

Pour placer une directive SPICE dans votre schéma, faites :

1. Placez un bloc de directive SPICE dans votre schéma. (Faites *Add*▷ *Component*▷ *spice*▷ *spice-directive-1.sym*)
2. Double cliquez sur le bloc et ajoutez/éditez les attributs suivants :
   - `refdes` : A ?
   - `file` : Le nom du fichier à inclure.

4.12 Traitement de modèles hiérarchiques

Dans la modélisation SPICE, il y a souvent des situations où vous souhaitez créer une représentation schématique de quelques composants particuliers tels que .SUBCKT et embarquer ensuite ce modèle de composant dans un schéma de plus haut niveau. Un exemple normal peut être comme ce qui suit : vous faite une simulation en micro-onde et vous voulez utiliser un modèle de condensateur qui inclut des inductances et des résistances parasites, en plus de la capacitance. Les fabricants de condensateurs fournissent souvent un schéma imprimé montrant une topologie de circuits incorporant des parasites et spécifiant des valeurs pour les parasites. Vous pourriez dessiner le modèle de condensateur en utilisant gschem, en faire la netlist pour pour créer un .SUBCKT et utiliser alors le .SUBCKT comme modèle de condensateur dans un schéma de plus haut niveau.

Comme cette tâche est très habituelle dans les simulations SPICE, `gnet-spice-sdb` le supporte maintenant (en démarrant avec rev 20030331). Pour

créer un .SUBCKT bas niveau et utiliser un schéma de plus haut niveau , faites ce qui suit :

1. Dessinez un schéma du composant bas niveau (i.e. le condensateur + parasites) en utilisant `gschem`.
2. Sur le schéma bas niveau, placez un bloc `spice-subcircuit-LL` (`spice-subcircuit-LL-1.sym`). Cela permet d'alerter le netlister que le schéma est un .SUBCKT Bas Niveau. Attachez les attributs suivants au symbole :
   – `model-name` = cond_avec_parasites
   (Bien entendu, « cond_avec_parasites » est l'exemple que nous avons utilisé ici. Utilisez votre propre nom de modèle dans votre schéma.) Lors du netlisting, ce symbole de schéma fera insérer « .SUBCKT cond_avec_parasites » lors de la netlist, dans la première ligne du fichier netlist.
3. Sur le schéma bas niveau, attachez un symbole `spice-subcircuit-IO` (`spice-subcircuit-IO-1.sym`) pour chaque liaison IO (i.e. les connexions au plus haut niveau). Le nombre de refdes de symboles IO dans le même ordre que vous souhaiteriez que les liaisons IO soient listées dans la ligne .SUBCKT dans le fichier de sortie. (i.e. P1 = premier, P2 = second, etc.)
4. Lorsque vous avez fini avec le schéma bas niveau, faits un netlist de la manière habituelle. Par exemple, si votre schéma est appelé `cap_with_parasitics.sch`, faites un netlist en indiquant :
   `gnetlist -g spice-sdb -o cond_avec_parasites.cir cond_avec_parasites.sch`
   Cela videra le contenu de la netlist SPICE dans un fichier appelé « `cond_avec_parasites.cir` ». Inspectez visuellement le fichier .cir pour vous assurer que le netlisting fonctionne correctement.
5. Créez ensuite un symbole pour le schéma haut niveau qui pointera vers le .SUBCKT. Notez que le symbole doit avoir un `refdes` démarrant avec la lettre « X ». Pour s'assurer que cela se produit, faites ce qui suit :
   (a) Utilisez `gschem` pour dessiner le symbole. Je dessine habituellement une boîte autour d'un symbole de modèle pour le distinguer d'un composant normal. Écrivez aussi les annotations que vous souhaitez.
   (b) Dans le symbole, assurez-vous que les broches sont dans le même ordre que celui que vous avez placé dans le .SUBCKT. C'est réalisé en modifiant le symbole avec un éditeur de texte et en modifiant l'attribut `pinseq`. Le netlister sortira les broches dans l'ordre déterminé par l'attribut `pinseq`.
   (c) En utilisant l'éditeur de texte, donnez au symbole un attribut `device` comme « capacitor-model ». **N**'assignez **pas** au symbole un des types de composants natifs listés dans l'annexe A ! L'objectif est de créer un symbole dont `refdes` débute avec « X » et si le `device` est un type reconnu, cela ne se produira pas.
   (d) En utilisant un éditeur de texte, donnez au symbole d'attribut `refdefs` un « X ? ».
6. Créez un schéma haut niveau. Placez votre composant nouvellement créé dans le schéma autant de fois que nécessaire et câblez le schéma de la manière habituelle.
7. Pour pointer votre symbole dans le .SUBCKT bas niveau, double cliquez sur le symbole et initialisez les attributs suivants :

- `file` = cond_avec_parasites.cir
- `model-name` = cond_avec_parasites

de même que tout autre attribut nécessaire (i.e. `refdes`).

8. Maintenant, netlistez votre schéma haut niveau de la manière habituelle. Le contenu de chaque fichier .SUBCKT est transféré dans la netlist principale. Inspectez votre netlist visuellement en utilisant un éditeur de texte pour s'assurer que tout est correct. C'est une bonne idée de prêter une attention particulière à ce qui suit :
   - Vérifiez que l'ordre de connexion des liaisons connectant la netlist de haut niveau au .SUBCKT bas niveau est correct.
   - Assurez-vous que le model-name haut niveau et le model-name bas niveau (dans la ligne de déclaration .SUBCKT) sont identiques.

Une fois que la netlist est créée, vous pouvez simuler votre dessin en utilisant le simulateur SPICE que vous souhaitez. Certains simulateurs fonctionnant sur GNU/Linux sont couverts ci-dessous.

Chapitre 5

Génération de netlist SPICE

5.1 En utilisant `gnetlist`

Une fois que le schéma est capturé, une netlist SPICE peut être générée en lançant le programme à ligne de commande de gEDA, `gnetlist`, sur le fichier de schéma. `Gnetlist` est architecturé en deux sections : un processeur avant écrit en C qui lit un fichier .sch et en crée une représentation interne, générique de votre dessin et un netlister arrière, écrit en SCHEME. En utilisant cette architecture, `gnetlist` est hautement personnalisable ; des utilitaires arrières SCHEME sont utilisés pour écrire des formats de netlists différents. La beauté de ce scheme (pun intended) est que les utilisateurs gEDA peuvent facilement écrire leurs propres netlisters pour adapter leurs propres applications. Le fichier Scheme interne qui implémente le netlisting SPICE avancé est appelé `gnet-spice-sdb.scm`, et il est situé dans le répertoire `${PREFIX}/geda/share/gEDA/scheme`.

`Gnetlist` avec `spice-sdb` est invoqué depuis la ligne de commande de la manière suivante : « `gnetlist [OPTIONS] -g spice-sdb filename1 ... filenameN` » . Les options de ligne de commande suivantes sont disponibles avec spice-sdb :

`-i` Mode scheme interactif

`-I` Placez `.INCLUDE <filename>` dans le fichier de sortie au lieu du contenu du fichier de modèles

`-q` Mode tranquille

`-l filename` Charge le fichier scheme avant le chargement de l'utilitaire arrière

`-m filename` Charge le fichier scheme après le chargement de l'utilitaire arrière mais encore avant la procédure d'exécution

`-g proc` Procédure scheme pour exécution (i.e. spice-sdb)

`-o filename` Noms de fichiers de netlist de sortie

`-c string` Exécute la chaîne comme un script scheme

`-v` Mode verbeux on

`-s` Trie la netlist de sortie (pour Gnucap)

5.2 Création de la netlist en utilisant `gnetlist` et `spice-sdb`

La création d'une netlist depuis un schéma est facile. Pour générer une netlist SPICE, faites simplement ce qui suit :

- Sauvegardez votre schéma dans <`filename.sch`>
- Créez votre netlist SPICE en effectuant "`gnetlist -g spice-sdb <filename.sch>`". La sortie est une netlist contenue dans le fichier `output.net`. Alternativement, si vous souhaitez donner à votre fichier de sortie un autre nom, initialisez le nom de sortie en utilisant l'option `-o`. Par exemple :
  `gnetlist -g spice-sdb -o amplifier.cir amplifier.sch`
  prend le schéma de dessin appelé « `amplifier.sch` » et sort une netlist SPICE appelée « `amplifier.cir` »".
- Inspectez votre netlist SPICE en utilisant un éditeur de texte. Vérifiez qu'il n'y a pas d'attribut manquant ou d'autres problèmes de netlist.

5.3 Les problèmes de netlisting habituels

La liste suivante tente de cataloguer les problèmes habituels avec la netlist et les solutions associées.

- `ERROR_INVALID_PIN :`
  Ceci peut se produire si l'attribut de symbole `pinseq` ne peut pas débuter à 1 ou il y a un trou dans la numérotation. Ceci doit être fixé en éditant le symbole lui-même dans un éditeur de texte.
- `ERROR : In procedure caddr :`
  `ERROR : Wrong type argument in position 1 : #f`
  Cette erreur est assez habituelle. Elle se produit habituellement lorsque vous oubliez d'ajouter un attribut obligatoire. Pour rectifier le problème, tentez de lancer gnetlist dans le mode bavard (« `gnetlist -v -g spice-sdb <filename.sch>` »). Le netlister stoppera le processus et extraira la partie avec l'attribut manquant. En ayant donc identifié la partie problématique, vous pouvez ré-ouvrir le schéma dans gnetlist et réparer les attributs.

Finalement, rappelez-vous qu'il est important d'inspecter manuellement votre netlist SPICE avant de l'utiliser dans une simulation. Veuillez garder en mémoire que le netlister est encore en qualité « bêta » et quelques problèmes peuvent encore exister dans la génération de netlist.

Chapitre 6

Simulation SPICE

Il existe plusieurs options pour effectuer des simulations SPICE sous GNU/Linux ; je vais y détailler :

- **LTSpice**, qui est un simulateur SPICE freeware originellement édité par Linear Technologies comme un outil de sélection/conception de composant lancé sous Windows. Comme c'est un outil SPICE très rapide et puissant, il est devenu un simulateur SPICE populaire parmi les hobbyists et les ingénieurs de conception qui préfèrent utiliser les outils libres. Originellement écrit pour Windows, LTSpice a été hacké pour fonctionner sous GNU/Linux en utilisant wine sur http://www.winehq.com/ ; je vous recommande son utilisation si vous avez besoin d'un simulateur SPICE robuste et de qualité professionnelle.
- **Ngspice,** qui est le simulateur SPICE « officiel » de la suite gEDA. Ngspice est une résurrection du code SPICE 3 pour GNU/Linux. Il fournit un engin de simulation, une interface à ligne de commande et la capacité de tracer les résultats de simulation graphiquement sous le X Windows System. Ngspice est natif GNU/Linux et open-source. C'est le SPICE de choix pour ceux qui veulent faire des simulation SPICE facilement sur GNU/Linux ou si l'on veut hacker et améliorer SPICE.
- **Tclspice**, est un fork du développement de ngspice. Tclspice est un super jeu de ngspice qui (en théorie) exporte le jeu de commandes SPICE vers un API TCL, vous permettant d'embarquer des analyses SPICE dans un programme TCL. C'est utile pour automatiser une optimisation de dessin, parmi d'autres choses. Tclspice est le simulateur à utiliser si vous êtes intéressés dans la conception avancée par scripts.

Il y a aussi un simulateur GPL appelé « `gnucap` », qui est basé sur (ou est le descendant de) Al's Circuit Simulator (`ACS`). Je ne l'ai pas beaucoup utilisé ; les informations sur http://www.gnu.org/software/gnucap/ sont donc à écrire.

6.1 LTSpice

LTSpice a été écrit par Mike Englehardt, ainsi que d'autres à Linear Technologies et il a été proposé par LinearTech comme une aide à la conception pour les ingénieurs souhaitant simuler les performances des contrôleurs d'alimentations à découpages de LinearTech. Le paquet incorpore une interface de capture de

schéma, un moteur SPICE rapide et puissant pour tracer les résultats d'analyse de plusieurs types de SPICE. Personnellement, je pense que l'interface de capture de schéma est difficile à utiliser et est très bruyant ; `gschem` a désactivé le son pour la facilité d'utilisation et les fonctionnalités. Néanmoins, le moteur de SPICE et le matériel d'analyse de LTSpice sont très bon.

LTSpice a été originellement développé pour fonctionner sous Windows mais Mike l'a hacké de telle manière qu'il puisse aussi fonctionner sous GNU/Linux avec wine. (Seul le menu d'aide est cassé – le reste du paquet fonctionne bien.) Une autre bonne fonctionnalité de LTSpice qui est bien supportée – Mike lit la liste de diffusion `sci.electronics.cad` régulièrement et est généralement heureux d'aider les personnes qui ont des problèmes avec son utilisation. C'est pourquoi, malgré son héritage Windoze, je recommande LTSpice comme un simulateur et un utilitaire puissant, de qualité professionnel pour gEDA.

6.1.1 Installation et configuration de LTSpice

Pour installer et configurer LTSpice, faites ce qui suit :

1. Téléchargez et installez wine. J'ai réussi en utilisant Wine-20030219. Les version ultérieures fonctionnent aussi probablement.
2. Téléchargez LTSpice. Il est disponible sous `http://www.linear.com/software` sous le nom de SwitcherCAD-III.
3. Lancez l'installateur de LTSpice sous wine.

6.1.2 Lancer LTSpice avec les dessins gEDA

LTSpice peut lire un fichier contenant une netlist SPICE de gEDA. J'ai réussi à faire des simulations LTSpice de la manière suivante :

1. Assurez-vous d'abord que vous êtes logués comme utilisateur normal – Wine n'aime pas fonctionner lorsqu'il est invoqué par root.
2. Créez un fichier dans votre répertoire de projet appelé « `Simulation.cmd` ». Placez dans ce fichier vos commandes d'analyse pour spice (i.e. .OP, .AC, .DC, etc.)
3. Placez un bloc d'include SPICE dans votre schéma. Pour l'attribut de fichier, faites « `Simulation.cmd` ».
4. Netlistez votre dessin.
5. Créez un lien depuis votre netlist « `output.net` » et une netlist dans le répertoire dans lequel est SwCADIII. Mettez le suffix .cir à la netlist. Par exemple :
   ln -s ${DESIGN_HOME}/output.net ${WINE_HOME}/.wine/fake_windows/Program Files/LTC/SwCADIII/MyDesign.cir
6. Lancez LTSpice : cd dans le répertoire où est SwCADIII et faites « `wine scad3.exe` »
7. Depuis l'interface graphique SwCADIII , faites : *File*▷ *Open*▷ *(files of type netlist [.cir])* et sélectionnez votre fichier.
8. Lancez le simulateur en cliquant sur le bouton run ou faites : *Simulate*▷ *Run.*

9. Sélectionnez les variables à tracer et cliquez alors sur OK. SwCADIII effectue le reste du travail.

Naturellement, il est très important de jouer avec LTSpice pour comprendre comment l'utiliser effectivement mais la description ci-dessus devrait suffire pour vous permettre de démarrer.

6.2 Ngspice

Ngspice a été démarré à l'Université de Rome « La Sapienza » par Paolo Nenzi comme une tentative de créer une version GPL du SPICE standard de Berkeley SPICE version 3 en réécrivant complètement le paquet SPICE. Les plans étaient aussi de créer des algorithmes de calculs meilleurs et plus robustes pour l'outil de simulation. Plus d'informations peuvent être trouvés sur le site web de ngspice : http://ngspice.sourceforge.net/. À la lumière de ces plans ambitieux, ce que fit Paolo est légèrement différent : il a pris le code SPICE 3 qui traînait depuis des années sur internet, l'a retravaillé et a modifié le système de construction de telle manière qu'il puisse compiler en utilisant la procédure normale de GNU make. Cela a été une réalisation majeure pour laquelle Paolo a réalisé de nombreuses prières. Malheureusement, lorsque l'on regarde la page web, le développement de `ngspice` semble avoir cessé à la fin 2001. En fait, le développement a fortement ralenti après 2001 mais Paolo a recommencé récemment à retravailler sur `ngspice`. Il a réalisé la dernière version, `ngspice-rework-15`, en Février 2004. Cette version est seulement disponible sur la page de téléchargement de Sourceforge ; Paolo n'a pas mis à jour le reste du site web du projet.

6.2.1 Installation and configuration de ngspice

Je trouve qu'il est généralement mieux de télécharger, configurer et compiler les sources de `ngspice` au lieu de tenter d'installer un paquet binaire. C'est l'approche que je développe ici.

6.2.2 Télécharger le code source

Obtenez la dernière version depuis :
`sourceforge` http://sourceforge.net/projetcs/ngspice. Assurez-vous que c'est la dernière version pour de meilleures performances et plus de fonctionnalités. Installez le code source là où vous en avez l'habitude. J'aime garder mes sources gEDA dans une répertoire séparé, par exemple
`/usr/local/geda/sources/ngspice`. Vous pouvez adopter un système similaire.

6.2.3 Extraire le code source

Le code source que vous avez téléchargé est distribué comme un « tarball », une archive compressée. Vous devez extraire les fichiers archivés en faisant :

user@host :˜$ cd <répertoireoùvousvoulezextrairelessources>
user@host :˜sources$ tar -xvzf </path/to/package.tar.gz>
user@host :˜sources$ cd <répertoireextrait>

À ce moment, vous êtes dans le répertoire haut de ngspice. Lisez les fichiers habituels, comme `README` et `INSTALL`, pour connaître le simulateur et le processus d'installation. Lire le fichier `NOTES` est aussi une bonne idée ; il contient des informations intéressantes si vous voulez développer ou déboguer des fonctionnalités présentes dans ngspice.

6.2.4 Configuration et compilation de ngspice.

Ngspice utilise la séquence typique « `configure && make && make install` » utilisée par les autres logiciels GNU. Il existe beaucoup d'options pour configure, disponibles pour ngspice. Une liste complète avec la documentation qui s'y rattache est à faire ; la meilleure manière de toutes les voir est de regarder dans `configure.ac`. Plusieurs options lors de la configuration sont pour le déboguage du simulateur ou permettent les analyses expérimentales. Pour les nouveaux, trois options de configuration valent la peine d'être mentionnées :

- `--enable-xspice` : Ce drapeau compile le support pour les extensions XSpice. Ces extensions vous permettent de définir les composants dont le comportement est donné par des « modèles de code » arbitraires. Incontestablement ; le modèle de code le plus important est `spice2poly`, qui est un modèle traduisant le style de construction POLY de SPICE2 dans un modèle XSpice utilisable par SPICE 3.
- `--with-readline` : Ce drapeau compile le support GNU readline dans `ngspice`, ce qui signifie que vous pouvez utiliser les commandes clavier dans le style d'emacs, de même que les touches fléchées pour se déplacer dans l'interface à ligne de commande (CLI). Sans cette fonctionnalité, l'interface de ligne de commande peut être hostile, ce qui signifie que si vous faites une erreur dans la saisie d'une longue commande, vous n'avez pas d'autre choix que de la ressaisir en entier. Paolo désapprouve l'utilisation de cette fonctionnalité car elle mélange du code GPL (readline) avec du code BSD (`ngspice`) mais il a laissé l'option ouverte aux autres pour les laisser décider ce qu'ils veulent faire.
- `--prefix` : Ce drapeau pointe sur le répertoire de base où vous souhaitez que soit installé votre binaire.

Avant de lancer configure, vous devriez contrôler les options que vous souhaitez inclure, une brève description est donnée dans l'annexe. Une fois que vous êtes prêts, saisissez :

user@host :˜sources/<tld>$./configure–enable-xspice –with-readline –prefix=/usr/local/geda <autres options de configuration>

Bien sûr, « `--prefix=` » doit pointer vers le répertoire où vous avez placé gEDA. Après la commande, votre simulateur est configuré et prêt à être compilé. La compilation est directe :

user@host :˜sources/<tld>$ make && make install

Comme toujours, vous aurez probablement besoin d'être root pour installer les paquets dans un répertoire public, auquel cas, vous devriez faire :

user@host :˜sources/<tld>$ make

...

...

...

user@host :˜sources/<tld>$ su -c make install

6.2.5 Tester l'installation

A ce moment, vous devriez pouvoir utiliser ngspice. Vous pouvez tester votre installation en essayant un des circuits de test placé dans le répertoire des tests. Je vous recommande de lancer le test TransImpedanceAmp, car il teste la fonctionnalité SPICE2 POLY.

6.2.6 Utilisation de ngspice

Lancer ngspice est très simple. Saisissez seulement la commande
user@host :~$ ngspice filename.net
au prompt de commande unix et ngspice chargera la netlist SPICE appelée `filename.net` dans son espace de travail et vous propose un prompt de commande ngspice. Vous pouvez lancer le simulateur en saisissant « `run` ». Vos résultats seront stockés dans des vecteurs SPICE pour une impression ou un tracé ultérieur. Le jeu de commande disponible est documenté sur :
http://newton.ex.ac.uk/teaching/CDHW/Electronics2/userguide/sec5.html#5
Pour utiliser les codemodel SPICE2 POLY, vous devez le charger dans `ngspice` **avant** de charger votre netlist. (Si vous le chargez après la netlist, les POLY de la netlist ne sont pas traduits et ne seront donc pas simulés correctement.) Pour charger le codemodel, indiquez simplement « `codemodel /usr/local/geda/lib/spice/spice2poly.cm` » (ou à l'endroit où vous mettez vos codemodels) au prompt ngspice. Notez que vous devez fournir le **chemin absolu** de la localisation de votre codemodel ; ngspice n'est pas suffisamment puissant pour chercher dans des zones par défaut. (Notez aussi que vous devriez spécifier l'endroit où est situé `spice2poly.cm` sur votre machine ; le chemin ci-dessus est celui du mien.)
Une meilleure manière de lire le codemodel `spice2poly` est de l'inclure dans le fichier d'initialisation de ngspice, « `spinit` ». Les fichiers d'initialisation sont dans le répertoire `/usr/local/geda/share/ng-spice-rework/scripts` (ou là où vous avez placé votre installation gEDA). D'autres personnalisations ngspice peuvent être aussi placées dans le fichier `spinit`.

6.3 Tclspice

Alors que la branche principale de développement de ngspice a hiberné en 2002, quelques amis à MultiGig Ltd. (http://www.multigig.com/) ont développé activement une branche de ngspice qu'ils ont appelé « tclspice ». Tclspice est une couche pour ngspice dans lequel beaucoup de commandes SPICE sont exportées comme API à TCL. L'objectif est de faciliter le scripting des analyses de SPICE. C'est un outil très puissant : Avec tclspice, vous pouvez écrire des scripts TCL qui fonctionnent en boucle, prennent les valeurs de composants, lancent une analyse et évaluent alors les performances du circuit avec les composants avant de boucler à nouveau. Cette possibilité peut évidement être utilisée pour effectuer une optimisation de circuit multi-circuits automatisée. Lorsqu'il sera complet, tclspice pourra être l'application « de la mort » pour l'EDA open-source.

6.3.1 Télécharger, installer et construire tclspice

La page d'accueil du projet Tclspice est sur : http://tclspice.sourceforge.net/. Les sources tclspice sont sur http://sourceforge.net/projects/tclspice. Le téléchargement et l'installation de tclspice suit les mêmes étapes que celle de ngspice détaillées ci-dessus. Comme tclspice est une sur-couche de ngspice, vous pouvez installer ngspice seul depuis les sources de tclspice si vous le désirez. Pour construire le paquet entier, il y a quelques étapes supplémentaires. Ici, je présente une série d'étapes qui permettront à la fois de construire ngspice (le programme pilotant de CLI seul) et l'API TCL pour la source tclspice.

Avant de construire tclspice, vous devez avoir les paquets suivants déjà installés :

- TclX (tclx8.3.5 fonctionne pour moi.)
- tclreadline (tclreadline-2.1.0 fonctionne pour moi.)
- BLT pour TCL (blt2.4z fonctionne pour moi.)
- TCL/Tk (8.4.3. fonctionne pour moi.)

Si vous n'avez pas déjà ces paquets sur votre GNU/Linux, vous avez besoin de les récupérer et les construire. Notez que construire TclX nécessite d'avoir les sources pour TCL et Tk, vous aurez donc besoin d'avoir ces sources si vous ne les avez pas déjà installées. Je l'ai fait fonctionner avec succès avec TCL/Tk 8.4.3, bien que les versions 8.3.X soient aussi supposées fonctionner. Aussi, si vous voulez lancer spice en tâche de fond vous avez besoin de recompiler TCL et Tk pour permettre le support des threads s'il n'ont pas encore été validés jusqu'ici (C'est le cas des paquets redhat).

En supposant que vous avez téléchargé et installé les paquets additionnels mentionnés ci-dessus, les étapes suivantes construiront à la fois ngspice et l'API TCL sur votre machine :

user@host :~sources/<tld>$ ./configure –enable-xspice –with-readline –prefix=/usr/local/geda

user@host :~sources/<tld>$ make && make install (this makes and installs regular old ngspice)

user@host :~sources/<tld>$ ./configure –enable-xspice –prefix=/usr/local/geda –enable-tcl –enable-experimental –disable-shared

user@host :~sources/<tld>$ make tcl && make install-tcl

Comme toujours, vous aurez probablement besoin d'être root de manière à installer les paquets dans un répertoire public, dans un tel cas, vous devriez faire :

user@host :~sources/<tld>$ su -c make install

user@host :~sources/<tld>$ su -c make install-tcl

pour installer vos paquets. Vous serez maintenant prêt à écrire des scripts TCL qui incorporent des commandes SPICE. Les informations sur l'installation de tclspice sont données ci-desous. Finalement, si vous êtes intéressés dans le développement de tclspice (ou même si vous ne l'êtes pas), il est une bonne idée de lire les fichiers de `NOTES` dans le répertoire de base des sources pour avoir quelques adresses intéressantes.

6.3.2 Utilisation de Tclspice

Tclspice est conçu pour exporter les commandes SPICE dans des programmes TCL. Pour utiliser tclspice, vous avez juste besoin d'indiquer « `package require spice` » au début de votre programme TCL. Après, pour invoquer une

commande SPICE, vous l'appelez simplement dans l'interface de spice. Par exemple, le programme TCL suivant lira une netlist SPICE, commandera une analyse de transitoire, lancera la simulation et tracera alors la tension observée sur la liaison Vout au cours du temps :

```
#! tclsh
package require spice
spice : :codemodel
/usr/local/src/tclspice-0.2.12/src/xspice/icm/spice2poly.cm
spice : :source netlistname.cir
spice : :tran 0.1ns 40ns
spice : :run
spice : :plot Vout
puts "All done now !"
```

Notez que comme tclspice ne lit pas le fichier d'initialisation « `spinit` », vous aurez besoin de placer toutes les commandes d'initialisation directement dans le programme TCL. Dans l'exemple ci-dessus, nous lisons le codemodel spice2poly directement dans la zone de travail. Plusieurs autres commandes sont aussi disponibles ; le jeu complet des commandes est documenté sur :

http://tclspice.sourceforge.net/docs/tclspice_com.html

6.3.3 Problèmes de Tclspice

Un problème majeur avec tclspice (qui a été hérité de ngspice) est le manque de mémoire. C'est la raison pour laquelle le temps pendant lequel vous pouvez lancer une simulation est limité. Cela signifie que si vous voulez faire une optimisation en bouclant dans un circuit beaucoup, beaucoup de fois, vous pouvez manquer de mémoire avant que votre programme ait complété son optimisation. C'est un problème connu et des efforts sont en cours pour combler le manque de tclspice.

Entre temps, il existe quelques développements qui permettent d'être utilisés sur ces dessins de tailles modérées pour faciliter les longues optimisations. Une méthode que j'ai employée est que l'optimiseur écrive son état courant dans un fichier après chaque analyse de circuit et lire ses conditions de démarrage depuis le même fichier. L'optimiseur stocke aussi la liste courante de ses meilleurs composants dans un autre fichier et lit ce fichier au début de chaque lancer. J'ai alors un programme TCL appelé `TaskMgr.tcl` qui fonctionne en boucle ; à chaque itération de la boucle, il fork un processus fils pour lancer l'optimiseur. Entre temps, le processus parent attend 5 minutes (un temps déterminé heuristiquement) et lance alors un signal « KILL » au fils avant de boucler et de redémarrer l'optimiseur à nouveau. De cette manière, l'optimiseur ne fonctionne jamais suffisamment longtemps pour consommer toute la mémoire de ma machine. Le programme `TaskMgr.tcl` est montré ici :

```
#! tclsh
package require Tclx
while {1} {
set PID [fork]
if {$PID} {
# Parent
after 300000
puts "About to kill child PID = $PID . . . ."
```

```
kill $PID
wait $PID
} else {
# Child
source Optimize.tcl
# If we ever get through this, we can print out the following :
error "We are done now!!!!!!"
}
}
```

Notez que `TaskMgr.tcl` a besoin du paquet TclX que vous avez déjà installé pour lancer tclspice. Vous pouvez aussi vouloir changer le temps d'attente pour différentes valeurs en fonction de votre mémoire et de la vitesses de votre machine. Finalement le parent doit attendre $PID car cela implique que le corps du processus enfant soit enlevé de la liste des tâches du noyau Linux lorsqu'il meurt. Sinon, vous laisserez un tas de processus fantômes roder dans votre machine lors du fonctionnement de l'optimiseur – une longue optimisation peut changer votre système en « la nuit des morts vivants » !

Cette méthode d'attente d'un certain temps pour le processus enfant est préférable si une analyse unique prend peu de temps comparé au temps nécessaire pour prendre toute la mémoire de la machine. Si le temps d'analyse est comparable au temps pris pour prendre toute la mémoire de la machine, une meilleure approche est d'avoir un parent qui garde une trace de l'état d'analyse, qui lance une analyse unique et la termine après chaque itération. Le fait d'être préférable dépend de la taille et de la complexité de votre dessin ; vous pourrez faire des expérimentations avec votre analyse pour voir combien de temps il faut et combien de mémoire elle prend. J'ai trouvé qu'un dessin avec six amplificateurs (avec les modèles de fabricants correspondants) et dans les 50 passifs fera fonctionner un PIII 333MHz avec 128Mo de RAM pendant 10 secondes. C'est pourquoi votre dessin doit être très grand avant qu'une analyse simple ne prenne suffisamment de mémoire RAM.

Chapitre 7

Conclusion

TBD

Annexe A

Composants natifs et leurs attributs.

Les composants et les attributs associés sont présentés dans ce tableau utilisé avec spice-sdb. Les attributs en gras sont nécessaires, les attributs en typographie normale sont optionnels. Notez que l'attribut `device` est invisible et est normalement attaché au symbole lorsqu'il a été créé. Les autres attributs sont attachés au symbole pendant la capture de schéma en utilisant `gschem`.

Lors du traitement des actifs simples (diodes, transistors) avec les modèles SPICE inclus dans le fichier, vous avez seulement besoin d'initialiser le model-name et les attributs de fichiers ; vous n'avez pas besoin d'initialiser l'attribut model. Néanmoins, si votre simple actif possède une ligne unique de modèle SPICE, vous pouvez souhaiter l'entrer directement dans le schéma, puis initialiser les attributs model et model-name ; vous n'avez pas besoin d'initialiser l'attribut de fichier.

composant	refdes	value	modèle	fichier	model-name	type	commentaire
RESISTOR	R ?	(4)	(2)	-	Nom du modèle	-	(11)
CAPACITOR	C ?	(4)	(3)	-	Nom du modèle	-	(11)
POLARIZED CAPACITOR	C ?	(4)	(3)	-	Nom du modèle	-	(11)
INDUCTOR	L ?	(4)	(3)	-	Nom du modèle	-	(11)
SPICE-ccvs	H ?	(5)	-	-	-	-	
SPICE-cccs	F ?	(5)	-	-	-	-	
SPICE-vscs	E ?	(5)	-	-	-	-	
SPICE-vccs	G ?	(5)	-	-	-	-	
SPICE-nullor	E ?	(5)	-	-	-	-	
DIODE	D ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
PMOS_TRANSISTOR	M ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
NMOS_TRANSISTOR	M ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
PNP_TRANSISTOR	Q ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
NPN_TRANSISTOR	Q ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
PFET_TRANSISTOR	J ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
NFET_TRANSISTOR	J ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
MESFET_TRANSISTOR	B ?	Numéro de part	(14)	Nom de fichier .model	Nom du modèle	-	(12)
IC	U ?	Numéro de part		Nom de fichier .model	Nom du modèle	-	Pour CI avec fichier .model
IC	X ?	Numéro de part		Nom de fichier .subckt	Name of .subckt	-	Pour CI avec fichier .subckt
model	A ?	-	(14)	Nom de fichier.model	(9)	(10)	(12)
include	A ?	-	-	Nom de fichier .include	-	-	(13)
options	A ?	(8)	-	-	-	-	(13)
directive	A ?	(1)	-	-	-	-	(12).
VOLTAGE_SOURCE	V ?	(6)	-	-	-	-	Source de tension indépendante
CURRENT_SOURCE	I ?	(7)	-	-	-	-	Source de courant indépendante

- (1) Chaîne à ligne unique contenant des définitions SPICE pour l'inclusion dans une netlist
- (2) Une ligne de paramètres de modèles SPICE (i.e. TC, etc.)
- (3) Une ligne de paramètres de modèles SPICE (i.e. IC, POLY, etc.)
- (4) Valeur numérique du composant
- (5) Chaîne décrivant le comportement de la source
- (6) Chaîne unique contenant le comportement de la source de tension
- (7) Chaîne unique contenant le comportement de la source de courant
- (8) ligne d'options dans l'include
- (9) Le nom du modèle pointé par les autres composants
- (10) Type de modèle SPICE correspondant (les types valide sont donnés ci-dessous)
- (11) Les paramètres modèles sont placés entre parenthèses après la valeur de composant
- (12) Pour modéliser, in doit inclure soit un modèle, soit un fichier
- (13) Place la directive .include dans la netlist SPICE
- (14) Modèle SPICE à ligne unique

"Natif au netlister" signifie qu'il existe un blob de code scheme correspondant qui sait exactement comment traiter ces composants et est garanti (pour la plupart) pour générer le code spice correct. Les symboles ayant des attributs « device » qui ne sont pas sur la liste au-dessus sont traités en utilisant la fonction scheme « spice-sdb :write-default-component », qui cherche le refdes du composant pour prendre une décision sur la manière de traiter le composant. En général, cette fonction fera « ce qu'il faut » lors de la génération du code spice mais cela n'est pas garanti. En particulier, cette fonction ne peut pas faire de distinction entre les types transistors de type N et P et générera un type <unknown> pour la chaîne .MODEL dans la netlist. Cela fera sûrement capoter votre simulateur SPICE. Il est donc mieux de s'assurer que tous les composants utilisés ont l'attribut « device » correct.

Annexe B

« Types » de values valides

L'attribut « type » est un drapeau signalant au moteur spice le type de composant et le prépare à accepter les paramètres de modèle spécifiques à ce type de composant. Les valeurs suivantes sont des « types » SPICE valides :

Component	"type"	Comment
RESISTOR	RES	
CAPACITOR	CAP	
POLARIZED_CAPACITOR	CAP	
INDUCTOR	IND	
DIODE	D	
PMOS_TRANSISTOR	PMOS	
NMOS_TRANSISTOR	NMOS	
PNP_TRANSISTOR	PNP	
NPN_TRANSISTOR	NPN	
PFET_TRANSISTOR	PJF	
NFET_TRANSISTOR	NJF	
MESFET_TRANSISTOR	-	