UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

IMPLEMENTACIÓN DE UN MODELO PARA HFETS EN SPICE3 DE BERKELEY

FRANCISCO JAVIER DEL PINO SUÁREZ

Las Palmas de Gran Canaria, Octubre de 1997

Título del Proyecto Fin de Carrera:

IMPLEMENTACIÓN DE UN MODELO PARA HFETS EN SPICE3 DE BERKELEY

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Proyecto Fin de Carrera

Implementación de un Modelo para HFETs en SPICE3 de Berkeley

Autor: D. FRANCISCO JAVIER DEL PINO SUÁREZ
Tutor: DR. D. ANTONIO HERNÁNDEZ BALLESTER

Las Palmas de Gran Canaria, Octubre de 1997.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

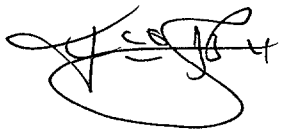
Proyecto Fin de Carrera

Implementación de un Modelo para HFETs en SPICE3 de Berkeley

Autor: D. FRANCISCO JAVIER DEL PINO SUÁREZ
Tutor: DR. D. ANTONIO HERNÁNDEZ BALLESTER

El Tutor

El Proyectando



Las Palmas de Gran Canaria, Octubre de 1997.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Proyecto Fin de Carrera

Implementación de un Modelo para HFETs en SPICE3 de Berkeley

Autor: D. FRANCISCO JAVIER DEL PINO SUÁREZ
Tutor: DR. D. ANTONIO HERNÁNDEZ BALLESTER

TRIBUNAL

Presidente: D. ROBERTO SARMIENTO RODRÍGUEZ
Vocal: D. ANTONIO NÚÑEZ ORDÓÑEZ
Secretario: D. VALENTÍN ~~DE~~ ARMAS SOSA
PEDRO PEREZ CARBALLO

Calificación: MATRÍCULA DE HONOR (10)

Las Palmas de Gran Canaria, a 7 *de* 11 *de 1997.*

AGRADECIMIENTOS

Creí que nunca iba a llegar el momento de escribir los agradecimientos. Ha pasado un año largo desde que comencé a trabajar en este proyecto. Elaborar una lista de todos aquellos que de una forma u otra han influenciado en mi trabajo casi correspondería a enumerar a gran parte de mis amigos y conocidos. Por ello, debo limitarme al corto número de influencias principales que seguramente nunca quede completo. Y no tengo otra forma de mostrar mi gratitud más que dar simplemente las gracias, pues como dice una famosa cantante actual,

"*Gracias. Es la mejor palabra que he encontrado para expresar lo que siento. Gracias a todas las personas que han puesto su ilusión, su cariño y su tiempo en este proyecto*".

Pero como no sólo se vive de ilusiones, sino que la mayoría de las veces hay que poner todo de nuestra parte para que salgan las cosas, quiero dar las gracias de forma especial a Toni, pues de hecho ha sido una de las personas que más ha puesto de su parte para que este proyecto llegase con éxito a su fin. También quiero dar las gracias a Javi, Benito y Juan Ignacio, compañeros del laboratorio que han estado ahí, junto a mí, durante todo este tiempo. Sólo espero que el "equipo" siga funcionando tan bien como ahora (y no me refiero precisamente a "Newton"). Gracias a Valentín por su *full-adder*, a Nelson y a Tomás por su ayuda al compilar por primera vez SPICE, a los compañeros del laboratorio de radar por explicarme que es una antena tipo *array* en fase, a Eli por prestarme su impresora (y por todo lo demás que es mucho...), y en general a todos los amigos que me han apoyado durante todo este tiempo.

Gracias finalmente a mi familia. Este agradecimiento no puede ser en grado alguno igual que los anteriores, pues aunque probablemente hayan contribuido sin darme cuenta con elementos intelectuales a mi trabajo, en forma diferente han hecho también algo muy importante. Han permitido que siguiera adelante e incluso han fomentado que lo hiciera. Para ellos gracias.

En fin, este es un día en el que la mayoría de los que como yo acaban su proyecto, están contentos y felices y sienten una sensación de alivio y satisfacción por el trabajo cumplido. Sólo espero que el trabajo que presento sea de interés y de utilidad para todos aquellos que lean estas páginas.

ÍNDICE

Índice

PETICIONARIO

PETICIONARIO

El peticionario de este Proyecto fin de carrera es el laboratorio de Tecnología de Circuitos adscrito al Departamento de Electrónica y Telemática y Automática de la Universidad de las Palmas de Gran Canaria.

OBJETO DEL TRABAJO

OBJETO DEL TRABAJO

El objeto fundamental de este proyecto es desarrollar la capacidad de incorporar modelos de funcionamiento de los dispositivos electrónicos en SPICE. Para ello se implementará el modelo de Angelov modificado por el FhG del HEMT en la versión 3e2 del programa de Berkeley. El resultado final será la redacción de un documento que, a modo de guía, permita reproducir el trabajo para cualquier otro dispositivo y modelo. El propósito de esto no es más que abrir una vía de desarrollo de nuestros propios modelos de dispositivos en el futuro.

SPICE es un programa de simulación de circuitos de propósito general para análisis no lineal en dc y en transitorio, y para análisis lineal en ac. Los circuitos pueden contener resistores, capacitores, inductores, inductores mutuos, fuentes de corriente y tensión independientes, cuatro tipos de fuentes dependientes, líneas de transmisión con y sin perdidas (dos implementaciones separadas), conmutadores, líneas RC uniformemente distribuidas, y los cinco dispositivos semiconductores más comunes: diodos, BJTs, JFETs, MESFETs, y MOSFETs.

La versión SPICE3 está escrita en C y se basa de forma directa en la versión SPICE2G.6 que está implementada en FORTRAN. Aunque SPICE3 incluye características nuevas, continúa soportando las características y modelos que más se usaban en las versiones anteriores.

La implementación del modelo IAF para los transistores HEMTs se hará en base al modelo de los MESFETs, pues es el que, desde el punto de vista de la estructura de las funciones matemáticas, más se parecen a dicho modelo.

RESUMEN

RESUMEN

La presente memoria de Proyecto Fin de Carrera se estructura en 7 capítulos. En el primero hacemos una introducción a SPICE y, en particular, a sus dos versiones más importantes SPICE2 y SPICE3 desarrolladas por la Universidad de California en Berkeley. Se detallan tanto los dispositivos que puede simular como los distintos análisis que se pueden realizar.

En el capítulo 2 se presentan las heteroestructuras y los transistores de efecto de campo de heteroestructura (HFETs). El estudio de este tipo de transistores se organiza señalando cual es la estructura básica así como el principio de funcionamiento de los mismos. También se presentan algunas consideraciones tecnológicas sobre la fabricación de dichos transistores.

Siguiendo con el estudio de los transistores HFET, en el tercer capítulo se revisa el modelo de Angelov [ANGEL92] el cual, a pesar de ser un modelo semiempírico, es hoy por hoy el modelo para simulación más aceptado y sobre el que se basan la mayoría de modelos que hay en la actualidad para los HFET. Un ejemplo de esto es el modelo del FhGIAF que es el que nosotros hemos implementado, y que, por tanto, es también objeto de estudio en este capítulo. Se eligió este modelo porque podemos comparar nuestros resultados con los de IAFSPICE, otra versión de SPICE que lo incorpora.

Una vez estudiados los transistores HFET pasamos a estudiar, en el capítulo 4, los algoritmos más importantes que conforman SPICE así como los principales problemas que pueden hacer que una simulación no llegue a buen término.

En el capítulo 5 se presenta la estructura global de SPICE3 así como la de los diferentes paquetes que lo componen. El objetivo fundamental de este capítulo es entender lo suficiente como para saber qué paquete de entre los que componen SPICE3 ejecuta una tarea determinada o cuando se encuentra un error asociado a una operación particular.

En el capítulo 6 se hace una descripción detallada de cómo se introduce un modelo de dispositivo en SPICE3. Esta tarea no puede ser llevada a cabo sin conocer, por un lado, el dispositivo y su modelo y, por el otro, el simulador y su forma de implementar los diferentes dispositivos. Por tanto, los estudios realizados en los capítulos anteriores resultan imprescindibles. Como ha sido dicho, el modelo implementado es el del HFET del FhGIAF que se presentó en el capítulo 3.

Finalmente, en el capítulo 7 llevamos a cabo la validación de los resultados obtenidos comparándolos con los resultados que da IAFSPICE. La

memoria finaliza con algunas conclusiones y trabajos futuros que se pueden llevar a cabo en base a este proyecto.

Capítulo 1

1. Introducción

El análisis de circuitos electrónicos es un problema común a la práctica totalidad de la "comunidad electrónica" a pesar de la enorme diversidad de tareas en que se ocupa.

Este problema está formalmente resuelto: basta tomar las leyes de Kirchoff y las ecuaciones constitutivas de los elementos del circuito y resolver la matemática asociada. Las ecuaciones constitutivas son las expresiones algebraicas con las que se describe el funcionamiento de cada uno de los componentes o dispositivos utilizados y son el objeto de la disciplina "modelado de dispositivos electrónicos".

El problema de análisis más sencillo consiste en encontrar el punto de operación en dc de un circuito lineal. Para un circuito pequeño compuesto por elementos lineales, descritos por ecuaciones constitutivas lineales, la solución exacta en dc se puede calcular a mano. Sin embargo, para circuitos lineales mayores el análisis en dc y más aún los análisis en el dominio de la frecuencia o en el dominio del tiempo se convierten en tareas mucho más complejas. El análisis de circuitos que contienen elementos descritos por relaciones no lineales entre las corrientes y las tensiones, añaden un nivel más de complejidad al requerir la resolución de ecuaciones de rama no lineales junto con las ecuaciones derivadas de las leyes de Kirchoff. La conclusión es clara, sólo se podrán resolver a mano circuitos pequeños obteniéndose soluciones aproximadas.

El problema se complica todavía más cuando se quiere predecir el funcionamiento de un circuito eléctrico con el tiempo o con la frecuencia. Las ecuaciones no lineales se convierten en ecuaciones integro-diferenciales, las

cuales sólo pueden ser resueltas bajo la aproximación de pequeña señal u otras restricciones.

Antes de la aparición de los simuladores eléctricos con los que los computadores resuelven las ecuaciones, los diseñadores que usaban elementos discretos tenían que utilizar placas tipo *protoboard* para analizar el funcionamiento de los circuitos eléctricos. Incluso en la actualidad se utiliza este método para construir circuitos analógicos. Sin embargo, este sistema es inadecuado para la fabricación de circuitos integrados. Esto se debe a que los transistores integrados dentro de un mismo chip funcionan de forma diferente a los transistores discretos situados en una placa de prueba. Además existe otro problema y es que los elementos integrados difieren de sus equivalentes discretos. La fabricación de un CI es un proceso caro tanto en coste como en tiempo lo cual hace que el diseño electrónico deba ser lo más preciso posible. Se hace por tanto necesario el empleo de un simulador circuital definido como aquel programa que permite dar solución a las ecuaciones que describen un circuito.

1.1 ¿Qué es SPICE?

SPICE (*Simulation Program with Integrated Circuits Emphasis*) es un programa de simulación de circuitos de propósito general para análisis no lineal en dc y en transitorio, y para análisis lineal en ac. Este programa resuelve las ecuaciones de red para las tensiones en los nodos del circuito. SPICE puede simular con igual precisión circuitos para varias aplicaciones que pueden ir desde fuentes de alimentación conmutables a células RAM. Los circuitos pueden contener resistencias, capacidades, inductores, inductores mutuos, fuentes de corriente y tensión independientes, fuentes de corriente y tensión dependientes, líneas de transmisión, y los dispositivos semiconductores más comunes como son los diodos, los transistores bipolares tanto de homounión como de heterounión (BJTs y HBTs), transistores de efecto de campo (JFETs), transistores MOS (MOSFETs), transistores FET metal-semiconductor (MESFET), y, en general, cualquier componente circuital cuya operación se describa mediante un conjunto de relaciones algebraicas.

El análisis en dc evalúa el punto de polarización del circuito con todas las capacidades en circuito abierto y todas las bobinas cortocircuitadas. SPICE utiliza métodos de iteración para resolver las ecuaciones de red no lineales; las no

linealidades se deben principalmente a las características corriente tensión (I-V) como las de los dispositivos semiconductores.

El análisis en ac calcula los valores complejos de las tensiones de los nodos de un circuito lineal en función de la frecuencia de una señal senoidal aplicada a la entrada del circuito. Para los circuitos no lineales, tales como los circuitos con transistores, este tipo de análisis requiere la suposición de pequeña señal; esto es, las amplitudes de las fuentes de excitación se supone que son pequeñas comparadas con la tensión térmica (V_{th} = KT/q = 25.8 mV a 27°C). Sólo bajo esta hipótesis se puede reemplazar un circuito no lineal por su equivalente lineal alrededor del punto de operación.

Con el análisis en régimen transitorio se evalúan las formas de onda de la tensión en cada nodo del circuito en función del tiempo. Se trata de un análisis en gran señal: las amplitudes de las señales de entrada no tienen restricciones. Esto implica que se están teniendo en cuenta las características no lineales de los dispositivos semiconductores.

Además de los análisis mencionados, que son los fundamentales, SPICE permite llevar a cabo otros tipos de análisis tales como análisis en función de la temperatura de operación, análisis de polos y ceros, análisis de sensibilidad, o análisis de ruido.

1.2 Descripción de SPICE2 y SPICE3

La mayor parte de los paquetes comerciales de SPICE se basan en SPICE2, versión g6, de la Universidad de California en Berkeley (USA). Actualmente, la Universidad de Berkeley concentra todos sus esfuerzos en SPICE3, concretamente en su versión f4. Aunque pocos productos comerciales se basan en SPICE3, la mayoría de ellos soportan muchas funciones de SPICE3 que no estaban disponibles en SPICE2.

SPICE2 es un programa de ejecución por lotes (*batch*). Si durante su ejecución se quiere cambiar un elemento del circuito o si se ha omitido una petición de salida, se deberán repetir los siguientes pasos: editar el fichero de entrada, ejecutar la simulación, y ver el fichero de salida.

Por contra, SPICE3 es interactivo de forma que el usuario entra al *shell* de SPICE (*spice shell*) cuando invoca el programa:

```
Spice 1 ->
```

Los comandos de SPICE3 deberán ser introducidos detrás del *prompt*. De esta forma, el usuario necesita conocer una serie de comandos adicionales que no estaban disponibles en SPICE2 para poder comunicarse con el programa. La figura 1.1 es la transcripción de una sesión interactiva de SPICE3 del circuito hemt.ckt. Como se puede observar, cada comando va seguido de la presentación en pantalla por parte del programa de los resultados. El primer comando, source, define el fichero de entrada; el segundo, listing, presenta en pantalla el listado del fichero de entrada para que pueda ser verificado; el tercero, op, ejecuta una simulación del punto de operación en dc del circuito; y el cuarto, print all, presenta en pantalla un listado de las tensiones en los nodos y las corrientes que circulan a través de las fuentes de tensión, es decir, prácticamente la misma información que daría SPICE2 si se incluyese el comando *.op* en el fichero de entrada.

Para circuitos grandes es útil hacer un display de todas las variables de salida disponibles, pero sólo imprimir (print) o representar (plot) las que sean más significativas.

Mientras un determinado circuito está activo, los resultados de los diferentes análisis pueden ser vistos gráficamente en la pantalla o impresos en un fichero. SPICE3 salva las tensiones de todos los nodos de todas las simulaciones llevadas a cabo en la misma sesión. Los ficheros de entrada se pueden editar y ejecutar cuantas veces se quiera dentro del *shell* de SPICE3. Para acabar la sesión basta con ejecutar el comando quit. Los resultados de las diferentes simulaciones se salvan en ficheros temporales, los cuales se muestran al usuario antes de salir del programa de forma que éste tiene una última oportunidad para guardar los resultados que desee. El resto de comandos y funciones disponibles en SPICE3 se puede encontrar en [JQNPS91] o haciendo uso del comando help el cual activa un sistema de ventanas de ayuda.

SPICE3 también puede ser ejecutado por lotes (modo *batch*) desde el *prompt* del UNIX con el comando:

```
% spice3 -b hemt.ckt > hemt.out
```

```
einstein% spice3
Program: Spice, version: 3e2
Date built: Wed Sep 24 16:11:16 WET DST 1997

Type "help" for more information, "quit" to leave.

Spice 1 -> source hemt.ckt

Circuit: CARACTERISTICA DE SALIDA DE UN HEMT

Spice 2 -> listing
        CARACTERISTICA DE SALIDA DE UN HEMT

   1 : caracteristica de salida de un hemt
   2 : vds 3 0 0
   3 : vgs 2 0 0
   4 : z1 1 2 0  efet03 w=25
   6 : vids 3 1
   7 : .model efet03 nmf level=2
         gexp=0.075 afact=4.0 vc=0.65 vsb=5.0 lambda=0.05 rd=600.0
         rs=600.0 rg=0.2 mfact=1.0 gamma=4.0 is=1.0e-14 dxl=10.0
         vs=0.4 cdvc=0.23e-3 cdvsb=0.0 delta=0.0 np=1.5 beta=2.8
         alpha=5.0 fc=1.5 vst=1.0 cgs0=2.5e-16 cgs1=9.0e-16 vat=-0.3
         vbt=0.3 qc1=0 rgs=800.0 rgd=800.0 td=3.0e-12
  37 :
  10 : .end
Spice 3 -> op
Warning: vids: has no value, DC 0 assumed
Spice 4 -> print all
v(1) = 0.000000e+00
v(2) = 0.000000e+00
v(3) = 0.000000e+00
vds#branch = -1.11662e-15
vgs#branch = 2.233247e-15
vids#branch = 1.116623e-15
z1#drain = -2.67990e-14
z1#gate = 1.116623e-14
z1#source = -2.67990e-14
Spice 5 -> quit
Warning: the following plot hasn't been saved:
op2     CARACTERISTICA DE SALIDA DE UN HEMT, operating point

Are you sure you want to quit (yes)?
Spice-3e2 done
einstein%
```

fig. 1.1 Transcripción de una sesión interactiva de SPICE3.

Después de la ejecución de este comando, el fichero de salida hemt.out contendrá información similar pero en diferente formato que la que produce SPICE2. Cuando se utiliza la opción -r SPICE3 produce un fichero *rawfile* el cual contiene

información útil para su representación gráfica. Si no se especifica ningún tipo de nombre de fichero, los datos se almacenarán en un fichero llamado rawspice.raw. El postprocesador para los ficheros *rawfiles* que acompaña a SPICE3 se llama NUTMEG.

Los ordenadores compatibles tipo PC se han convertido en la plataforma más común para ejecutar SPICE. Aunque el SPICE2 de la UC Berkeley no está disponible para PCs, en la actualidad existe un número considerable de ofertas comerciales, tales como PSpice de MicroSim, IsSpice de Intusoft, y HSPICE de Meta Software, que corren bajo DOS o Windows. La UC Berkeley distribuye SPICE3 tanto para PCs como para estaciones de trabajo bajo UNIX.

Aunque la secuencia de operaciones es básicamente la misma, el comando que hay que invocar para hacer una simulación puede diferir de un paquete a otro. Algunos paquetes ofertan *shells* para DOS, los cuales facilitan la secuencia de simulación: creación o modificación del fichero de entrada, ejecución de SPICE, y visualización de resultados. En la actualidad, la mayoría de los paquetes disponibles en el mercado permiten la captura de esquemáticos, de forma que la especificación del circuito se hace de forma gráfica en vez de a través del lenguaje tipo *netlist* típico de SPICE. Otra ventaja de estos paquetes es que la mayor parte de las operaciones se realizan a través de menús, con lo que se facilita la tarea al diseñador.

El comando típico para ejecutar Pspice desde DOS es:

```
C > PSPICE BJT.CKT BJT.OUT
```

donde BJT.CKT y BJT.OUT son los ficheros de entrada y salida respectivamente. El fichero de salida de PSpice tiene el mismo formato y contiene la misma información que el de SPICE2. SPICE3 y NUTMEG se pueden ejecutar de forma similar en un PC. En primer lugar se ejecuta SPICE3 en modo *batch*, y luego se ejecuta NUTMEG para ver los resultados. (Los dos comandos se pueden salvar en un fichero .BAT, SPICE3.BAT, el cual ejecuta BSPICE, el nombre del SPICE3 para PC en modo *batch*, y luego ejecuta NUTMEG con el fichero RAWSPICE.RAW creado por BSPICE).

Capítulo 2

2. Estructura y principio de funcionamiento del HFET

El desarrollo de las técnicas MBE (*molecular beam epitaxial*) y MOCVD (*metal-organic chemical vapor deposition*) desde finales de los años 70, ha revolucionado el diseño de nuevos dispositivos electrónicos y optoelectrónicos. En nuestros días se pueden crecer capas muy finas de material semiconductor manteniendo un fino control sobre el dopaje y las conexiones. Este desarrollo tecnológico ha propiciado la consolidación de una disciplina científica: la "ingeniería de la anchura de la banda prohibida" (*band-gap engineering*), que estudia las heterouniones. Una heterounión es una estructura constituida por la unión de dos semiconductores distintos y, por ende, con diferente anchura de la banda prohibida.

En las estructuras convencionales (homouniones: el mismo semiconductor a ambos lados de la unión) los electrones y los huecos experimentan fuerzas proporcionales a los gradientes de concentración de impurezas (difusión) y al campo eléctrico aplicado (arrastre). Las fuerzas electrostáticas serán iguales y opuestas en electrones y huecos debido a que la anchura de la banda prohibida es constante. Las heterouniones ofrecen un grado adicional de libertad en el diseño de dispositivos, ya que las fuerzas que actúan sobre los electrones y los huecos pueden ser controladas de forma independiente mediante el ajuste de los anchos de banda prohibida (es decir, de la composición) y del dopaje de los semiconductores.

El desarrollo de uniones de buena calidad entre semiconductores binarios (III-V) y sus aleaciones (III-III-V o III-V-V) es posible si las constantes de red de los materiales son la misma, o de valor muy próximo; pero esta imposición limita la elección de las energías de la banda prohibida (E_G), banda de conducción (E_C), y

banda de valencia (E_V), y de sus posibles combinaciones, a los valores que se obtienen de la figura 2.1 [LONBU90]. Esta figura tiene como ejes el ancho de la banda prohibida y la constante de red, y en ella se muestran los semiconductores III-V más usados. Las combinaciones que tienen la misma constante de red, tales como GaAs/AlGaAs o InGaAs/InAlAs, están alineadas verticalmente en la figura.

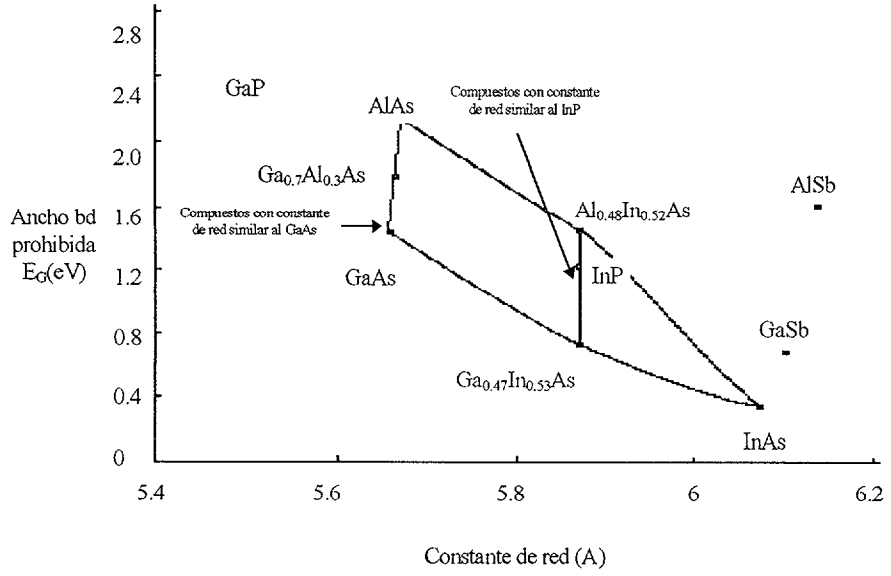


fig. 2.1 Ancho de banda prohibida en función de la constante de red para los semiconductores III-V más usados [LONBU90].

2.1 Perfil de una Heteroestructura

En la figura 2.2 se muestran los diagramas de bandas de energía de una heteroestructura formada por AlGaAs dopado con impurezas donadoras (n^+) y GaAs sin impurezas. El que el GaAs no esté dopado implica que su nivel de Fermi (E_F) se halla situado aproximadamente a la mitad de la banda prohibida. El nivel de Fermi en el AlGaAs se encuentra cerca de la banda de conducción debido a las impurezas n^+. Esto se puede observar en la figura 2.2.a. En dicha figura, los semiconductores no están en contacto y se han dibujado de tal forma que el *offset* de la banda de conducción ΔE_C y el *offset* de la banda de valencia ΔE_V están separados según las siguientes relaciones experimentales:

$$\Delta E_V = 0.55 x_{Al} \text{ y } \Delta E_V = 0.75 x_{Al},$$

ambas funciones lineales de la fracción molar del Al (x_{Al}) [AFKHM84]. Obsérvese que los *offsets* no dependen del dopaje, sino sólo de la composición.

En la figura 2.2.b los semiconductores se han puesto en contacto. Como no hay ninguna tensión aplicada a la heterounión, los niveles de Fermi estarán alineados. En esta figura no se han tenido en cuenta los *offsets*. En las figuras 2.2.c y d se puede observar cómo la inclusión de los *offsets* en los diagramas de bandas produce que éstos se curven, apareciendo, en el caso presentado, una discontinuidad en la banda de conducción del sistema en equilibrio. A esta discontinuidad o "pico" se alude más adelante.

El *offset* de la banda de valencia ΔE_V se encuentra justo entre las bandas de valencia de ambos semiconductores, de forma que, la pendiente en el GaAs y en el AlGaAs es la misma a ambos lados de la interfase. La ecuación de Poisson exige que la pendiente de las bandas de energía sean proporcionales al campo eléctrico

$$\nabla^2\Phi = {}^{\rho}\!/_{\varepsilon_0} \Rightarrow \nabla\nabla\Phi = {}^{\rho}\!/_{\varepsilon_0} \Rightarrow -\nabla\Phi = E \Rightarrow pte \propto E$$

Como la permitividad es la misma y el campo eléctrico debe ser continuo en la unión de ambos semiconductores, se deberá cumplir que la pendiente de las bandas de energía sea la misma a ambos lados. En el AlGaAs, si la concentración de impurezas donadoras es uniforme, la curvatura es parabólica [SMSZE81].

Las bandas de conducción deben seguir a sus respectivas bandas de valencia debido a que la anchura de la banda prohibida es constante para un mismo semiconductor. En la figura 2.2.d se observa cómo se unen las bandas de conducción, siendo el *offset* ΔE_C la altura de la discontinuidad. El pico que se observa en la banda de conducción del AlGaAs es una región que está vacía de electrones. Es decir, esta zona será una zona de deplexión en la que habrá una carga fija positiva asociada a las impurezas donadoras ionizadas. La carga fija negativa correspondiente que se requiere para que se dé la condición de neutralidad en la interfase estará en el valle de la banda de conducción, el cual actúa como pozo de potencial para los electrones de conducción.

En el siguiente apartado se presenta una descripción de los transistores de efecto de campo de heteroestructura.

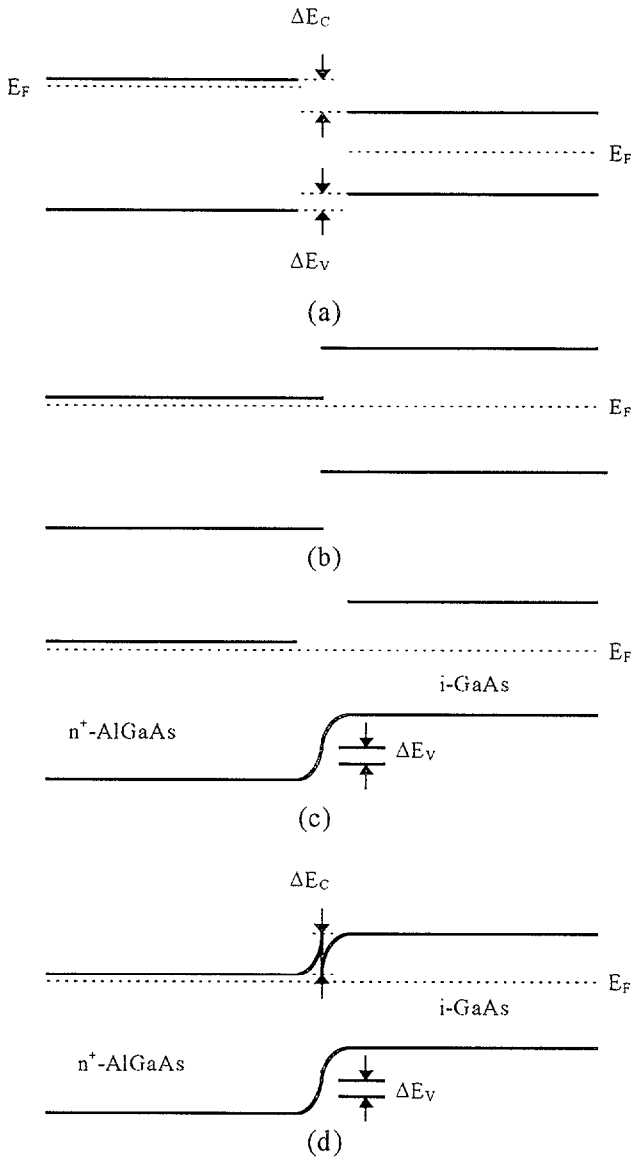


fig. 2.2 Construcción de un diagrama de bandas de energía de una heterounión: (a) antes del contacto, (b) después del contacto los niveles de Fermi quedan alineados, no se consideran los *offsets*, (c) incluye el *offset* de la BV, (d) incluye el *offset* de la BC.

2.2 Transistores de Efecto de Campo de Heteroestructura

La aparición de los Transistores de Efecto de Campo de Heteroestructura (HFET) ha supuesto casi una revolución en el campo de las microondas de ultra-alta velocidad y de los circuitos electrónicos digitales. Los HFETs muestran prestaciones extremadamente elevadas, encontrándose HFETs discretos de microondas con figuras de ruido de 1.3 dB a 60 Ghz y ganancias de 9.5 dB [SMSZE90]. En relación a los circuitos digitales, se han conseguido circuitos con retardos de propagación de 10 ps por puerta y memorias SRAM de tiempo de acceso de 0.5 ns [SMSZE90].

La mayor parte del trabajo realizado sobre los HFETs se basa en dispositivos de canal tipo n con heteroestructuras AlGaAs/GaAs. Como se vio en el apartado anterior, en este tipo de uniones, se hace crecer un material dopado de ancho de banda prohibida grande (AlGaAs) sobre otro no dopado de ancho de banda prohibida pequeño (GaAs). A este tipo de estructuras se las denomina "*modulation doped heterostructure*". La estructura física de la heterounión es tal que el transporte de carga dentro del dispositivo electrónico se puede optimizar ajustando las anchuras de las capas y los niveles de dopaje.

Los HFETs tienen varias denominaciones, como son SDHTs (*Selectively Doped Heteroestructure Transistors*), HEMTs (*High Electron Mobility Transistors*), TEGFETs (*Two-dimensional Electron Gas FETs*), y MODFETs (*Modulation Doped FETs*). Nosotros utilizaremos indistintamente las denominaciones HFET y HEMT. Si bien la estructura más utilizada es la n^+-AlGaAs/GaAs, también nos podemos encontrar con otro tipo de estructuras como pueden ser: AlGaAs-InGaAs-GaAs (dopado homogéneamente y simple y doble δ-dopado), AlInAs-GaInAs-InP, etc..

En la sección 2.2.1 presentamos la estructura básica de los HFETs y su principio de funcionamiento. En la sección 2.2.2, haremos una descripción de las diferentes variantes a la estructura HFET convencional con las que nos podemos encontrar. Por último, acabaremos este capítulo haciendo un repaso de las principales aplicaciones que tienen los HFETs. Este será el objetivo de la sección 2.2.3.

2.2.1 Estructura de Bandas y Comportamiento del Dispositivo

En esta sección presentamos las características físicas de los HFETs. En primer lugar haremos una descripción de la estructura convencional de estos transistores, y seguidamente pasamos a explicar su principio de funcionamiento. Para terminar se comentarán algunas consideraciones tecnológicas de la fabricación de los HFETs.

2.2.1.1 Estructura convencional de los HFETs de n^+-AlGaAs/GaAs

En la figura 2.3 se muestra la sección transversal de un HFET convencional. Los contactos de fuente y de drenador son óhmicos, como los de los transistores MOS de Si o los MESFETs de GaAs. La puerta se implementa mediante

una barrera Schottky. La estructura de capas epitaxiales del dispositivo se muestra en la figura 2.4. El dispositivo se hace crecer a partir de un substrato de GaAs. La concentración intrínseca de portadores en el substrato de GaAs intrínseco es baja (n_i = 2.3 $10^6 cm^{-3}$ a temperatura ambiente) como resultado de la amplia banda prohibida del GaAs. Es por ello por lo que el material intrínseco tiene una resistividad del orden de 10^8 Ωcm por lo que se le suele llamar material semi-aislante. Sobre el substrato semi-aislante se hace crecer una capa de GaAs sin dopar o ligeramente dopada tipo p denominada *buffer* del orden de 1μm de profundidad. Sobre ésta se hace crecer otra capa sin dopar de 10nm de AlGaAs denominada capa espaciadora, y a continuación otra de n^+-AlGaAs fuertemente dopada del orden de 50 a 100 nm llamada capa donadora. La capa de AlGaAs no dopada separa los electrones de las impurezas del canal. Por último, se hace crecer una capa de n^+-GaAs denominada *cap*. Las anchuras y dopajes de las diferentes capas influyen directamente sobre las propiedades del HFET. Las longitudes de puerta varían desde 1.0 a 0.1 μm de acuerdo con las restricciones de velocidad, aplicación y *yield* que se requieran.

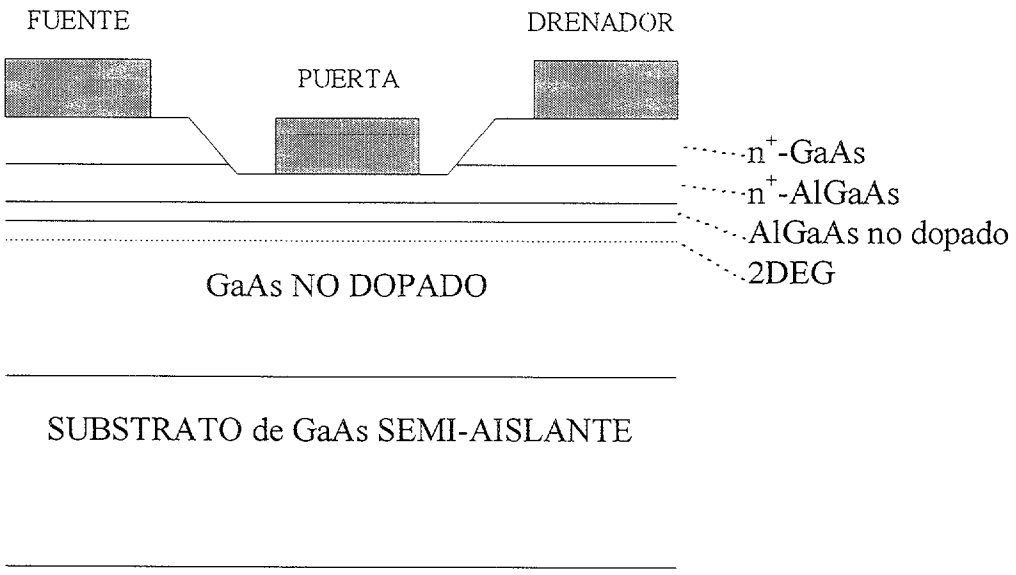


fig. 2.3 Sección transversal de un HFET convencional.

2.2.1.2 Principio de funcionamiento

Como vimos en el apartado 2.1, al formarse la heterounión se crea un valle en la banda de conducción. Los electrones se acumulan en este valle o pozo de potencial formando una zona de carga superficial análoga al canal de inversión que se forma en la estructura metal-oxido semiconductor SiO_2/Si de los MOS. La anchura de este canal es

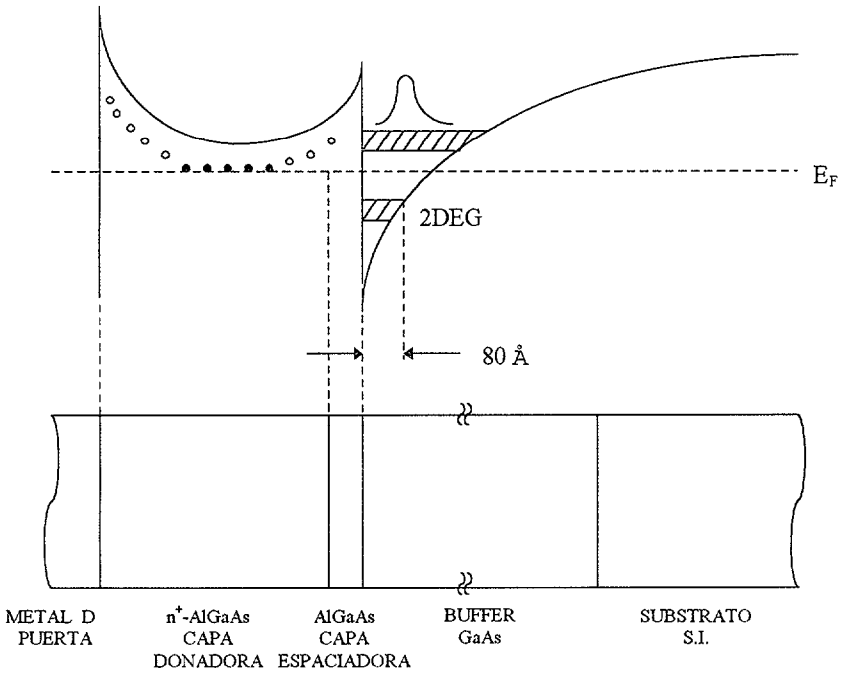


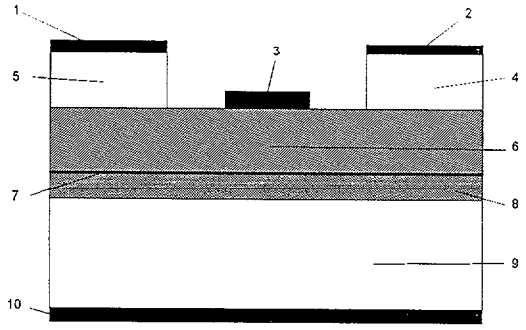
fig. 2.4 Estructura de capas epitaxiales y diagrama de bandas de energía para un HFET con polarización positiva.

muy pequeña con lo que los electrones quedan atrapados en un sistema bidimensional en la heterointerfase. Este es el motivo por el cual el canal de los HFETs se denomina
gas de electrones bidimensional (2DEG: *two-dimensional electron gas*). La separación física de los electrones de las impurezas donadoras reduce la dispersión por impurezas y por tanto aumenta la movilidad así como la velocidad efectiva de los electrones bajo la influencia de un campo eléctrico. Esta superficie o lámina de electrones de elevada movilidad se puede utilizar como canal activo de un FET y se puede modular por el efecto de un campo desde un electrodo de puerta. Esta es en definitiva la base del principio de funcionamiento de los HFETs.

2.2.1.3 Consideraciones tecnológicas

En la figura 2.5 se muestra la estructura típica de un HFET en más detalle.

El primer paso en la fabricación del dispositivo es su aislamiento eléctrico por medio de ataque químico "MESA" bajo el canal no dopado o el substrato semi-aislante. Éstas regiones se aprecian en la figura a ambos lados de la fuente y el drenador.



1. Electrodo de fuente.
2. Electrodo de drenador.
3. Contacto metálico de la puerta Schottky.
4. Drenador. GaAs tipo n. $N_D=2\cdot10^{20}cm^{-3}$.
5. Fuente. GaAs tipo n. $N_D=2\cdot10^{20}cm^{-3}$.
6. AlGaAs tipo n. $N_D=8\cdot10^{17}cm^{-3}$.
7. Dopaje delta tipo n. $N_D=1.3\cdot10^{19}cm^{-3}$.
8. AlGaAs intrínseco.
9. GaAs intrínseco.
10. Electrodo del substrato.

fig. 2.5 Estructura típica detallada de un HFET

Entonces se definen las áreas de fuente y drenador con fotorresistencia positiva y se evapora AuGe/Ni/Au como metal de contacto en fuente y drenador. El contacto metálico forma aleación durante 1 minuto a la temperatura de 44°C.

Como casi todas las dimensiones están por debajo de la micra, el tamaño y forma de los contactos óhmicos es crucial en el funcionamiento de estos dispositivos. Estas resistencias deben minimizarse todo lo posible, por ello, durante este proceso se difunde Ge para que haga contacto con el gas bidimensional.

A continuación se define la puerta generalmente mediante borrado químico o ataque de iones reactivos, con la ayuda de alguna capa que detenga el ataque. La profundidad del hueco se selecciona dependiendo de si se quiere que el dispositivo opere en modo de deplexión o enriquecimiento. En modo de deplexión, el grosor de la capa dopada debe ser tal que la barrera Schottky la vacíe de portadores, pero no vacíe el canal. En modo de enriquecimiento, la capa del AlGaAs dopado es más fina, de modo que el canal está también vacío.

El estrés mecánico causado por las interconexiones metálicas es un serio problema en dispositivos VLSI. La causa es el distinto coeficiente de expansión térmica de los metales y materiales dieléctricos de los del substrato (GaAs, InP o Si). Cuando el estrés es muy elevado, puede llevar a la rotura de la capa o fallo en la interfase, con lo que se inutitiliza el dispositivo.

Para reparar los daños por la exposición y activar las especies implantadas se requiere un proceso de recocido a temperatura elevada. Un método es colocar la muestra sobre una plataforma con poca masa térmica y calentarla con infrarrojos

desde una lámpara de cuarzo a la temperatura de 850°C. Después de 15 segundos se apaga la lámpara y se deja enfriar por debajo de 400°C.

2.2.2 Familias de los dispositivos HFETs

Las tres aplicaciones más importantes de los HFETs surgen con la necesidad de obtener dispositivos FET con elevada capacidad de suministrar corriente para circuitos integrados digitales, transistores FET de potencia con corrientes y tensiones de ruptura elevadas y transistores FET de bajo ruido para microondas y ondas milimétricas. Para los circuitos digitales y para los FETs de potencia, las características más importantes exigibles a los dispositivos son una transconductancia extrínseca elevada y un amplio rango de puntos de operación posibles. Para el caso de los dispositivos de microondas, la transconductancia pico es importante, y en todos los casos, es deseable el tener una capacidad puerta-canal mínima para que la modulación en la carga del canal sea efectiva. El confinamiento de los portadores en un canal bien definido elimina dos efectos indeseados. Uno es la transferencia de portadores desde el canal hasta las capas de confinamiento de anchos de banda mayores, lo cual traería con sigo la formación de un MESFET paralelo en el AlGaAs. El segundo es la inyección de portadores en el substrato, la cual traería con sigo un incremento en la conductancia de salida del transistor, especialmente en dispositivos con anchura del canal pequeña. Por último, la necesidad de tener un buen comportamiento en dispositivos con anchura del canal pequeña, donde se requiere la mayor velocidad posible, se hace crítica para una tecnología HFET.

Siguiendo estas sencillas pautas de trabajo, podemos encontrar dos variantes básicas de HFETs. Una consiste en un dispositivo cuya estructura se optimiza para conseguir el mejor confinamiento del canal posible. Este tipo de HFET requiere un ajuste de las anchuras y composiciones de las capas. La segunda variante consiste en un HFET cuyo dopaje es distribuido para permitir que la carga pueble el dispositivo. El dopaje se puede hacer a través de una capa dopada en volumen (*bulk-doped layer*), una capa dopada en δ (δ- *doped layer*), una super red (*superlattice*), dopando el propio canal de conducción, o creando un canal de inversión, sin dopaje. En la tabla 2-1 se muestra los distintos tipos de HFETs que podemos encontrar. Los diagramas de bandas de los diferentes tipos de HFETs se muestran en la figura 2.6. En [SMSZE90] se puede encontrar una descripción detallada de cada uno de los dispositivos aquí presentados.

Tabla 2-1 Familia de los dispositivos HFETs

Capa Donadora	♦ Dopaje Selectivo	• Capa dopada en volumen (SDHT, MODFET, TEGFET, HEMT, etc) • Capa dopada en δ • Super red
	♦ Puerta Aislada	• MISFET • SISFET
Confinamiento del Canal	♦ Canal de pozo cuántico (SQW, etc.) ♦ Estructura invertida (I-HEMT, I^2-HEMT, etc.)	
Dopaje del Canal	♦ No Dopado ♦ Dopado (DMT, etc.)	

2.2.3 Aplicaciones de los HFETs

Los dispositivos y circuitos HFETs son adecuados para su uso en aplicaciones de alta velocidad, bajo ruido, y baja potencia. Otras aplicaciones para las que pueden ser útiles estos dispositivos son, por un lado las comunicaciones, y por el otro el procesamiento de datos. En cuanto a las aplicaciones en comunicaciones podemos encontrar enlaces radar, emisión directa de televisión por satélite, telefonía móvil, convertidores de televisión por cable, etc.. En cuanto a las aplicaciones de procesamiento de datos podemos encontrar supercomputadores, conmutadores para redes ATM [MAEAS96], etc., aplicaciones todas ellas en las que las características de velocidad elevada y consumo de potencia bajo son críticas.

También existe un elevado número de aplicaciones militares posibles para los HFETs, en particular las antenas radar del tipo array en fase (*phased-array antenna radar*) [SKOLN88]. Estas consisten básicamente en un gran número de transistores individuales conectados en un circuito integrado que operan a frecuencias de microondas. El éxito de este tipo de antena está en la miniaturización, lo cual requiere un compromiso en la adaptación de los transistores con los otros elementos en el circuito de microondas. Los transistores normalmente no pueden operar en sus condiciones óptimas, y por lo tanto, para conseguir unas prestaciones aceptables, se necesitan dispositivos de muy buena calidad de forma que, incluso con sus prestaciones degradadas, estas todavía se adecuen a su función.

Por último, una aplicación que puede ser fundamental es la de la televisión de alta definición, en donde la complejidad de los circuitos electrónicos requieren potencias de disipación de los transistores individuales mínimas.

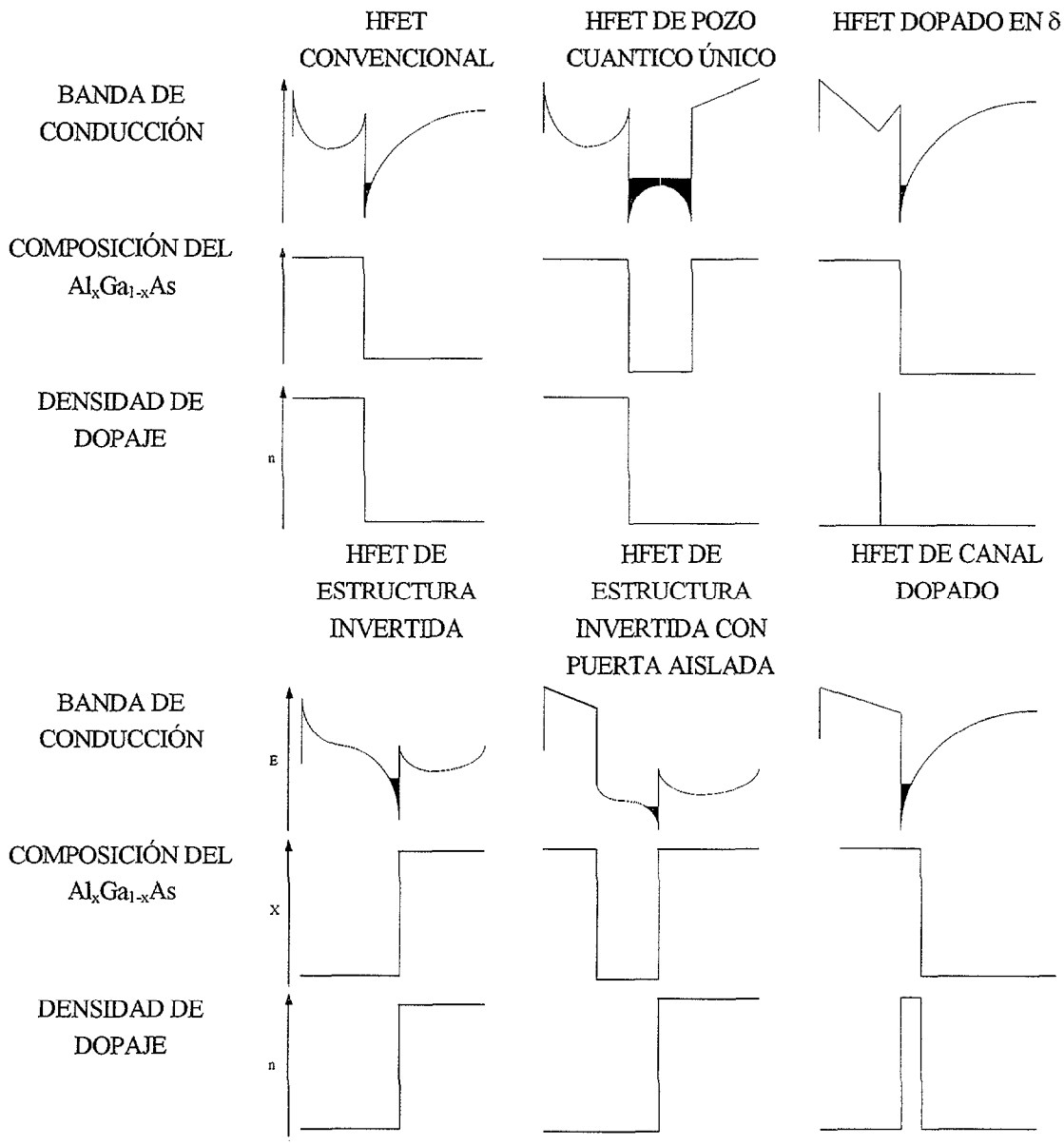


fig. 2.6 Diagramas de Bandas, Composición, y Dopaje de los diferentes HFETs.

Capítulo 3

3. Modelos del HFET

En la actualidad, los HFETs han demostrado ser los dispositivos óptimos para las aplicaciones de bajo ruido y ondas milimétricas [SCNRR94]. El uso de dichos dispositivos en aplicaciones no lineales, en la actualidad menos desarrollado, es un hecho de importancia, debido al incremento de la demanda de la integración MMIC de aplicaciones de ondas milimétricas no lineales. Para esas aplicaciones, los MESFETs no son aplicables debido a su limitado rango de frecuencias. Sin embargo, mientras que la caracterización del comportamiento no lineal de dichos componentes es ya bien conocido, la caracterización de los HFETs está todavía en su infancia.

En cuanto a las aplicaciones digitales, las tecnologías basadas en HFETs están todavía en proceso de maduración. No obstante, sus prestaciones en velocidad y consumo, así como las crecientes posibilidades de integración, las hace aparecer como una evolución probable de las futuras tecnologías digitales de GaAs [MAEAS96]. Las familias lógicas utilizadas y las metodologías de diseño utilizadas son normalmente desarrollo directo de las usadas con MESFETs sobre GaAs. Así, la familia lógica utilizada normalmente es la *Direct Coupled FET Logic* (DCFL) por su simplicidad y prestaciones. Para atacar cargas elevadas se escoge normalmente *Source-follower Direct Coupled FET Logic* (SDCFL) y *Super Buffer FET Logic* (SBFL).

Las herramientas de diseño para aplicaciones en gran señal de MESFETs y HFETs, requieren idealmente un único modelo CAD que pueda proporcionar descripciones precisas del comportamiento no lineal dc y ac sobre un amplio rango de puntos de polarización.

Al principio se hicieron intentos para utilizar el modelo convencional de los MESFETs para los HFETs, y así aparecieron los modelos de Curtice [CURTI80], Curtice-Ettemberg [CURET85] y Materka [MATKA85]. Esos primeros esfuerzos fueron sólo parcialmente exitosos debido a las características peculiares de los HFETs. De hecho, aunque los MESFETs y los HFETs admiten el mismo circuito equivalente para pequeña señal y ambos exiben un comportamiento en frecuencia similar hay diferencias en su funcionamiento. Quizá la discrepancia más importante es la compresión de la transconductancia, g_m, para bajas alimentaciones de puerta observadas en los HFETs y que no se detecta en los MESFETs.

Por lo tanto, empezaremos este capítulo viendo con detenimiento el modelo que más aceptación ha tenido para la simulación de los HFETs (ap. 3.1). Dicho modelo es el de Angelov [ANGEL92] a partir del cual se han derivado la mayoría de los modelos existentes para los HFETs. Un ejemplo de esto es el modelo del FhGIAF que es el modelo que hemos implementado nosotros y por ello su presentación será el objetivo del siguiente apartado (ap. 3.2).

3.1 Modelo de Angelov

3.1.1 Introducción

En la actualidad existen varios modelos empíricos para la simulación de MESFETs y HFETs en circuitos no lineales [CURTI80], [CURET85], [MATKA85], [STANE87], [TAWRM81], [MAANE90]. Estos modelos se usan para predecir la ganancia, la distorsión por intermodulación, la generación de armónicos, etc., en función de la alimentación, para circuitos tales como amplificadores, mezcladores, y multiplicadores. En estos modelos no se tiene en cuenta el hecho de que no sólo se tiene que modelar de forma correcta la característica corriente-tensión $I_{ds}[V_{gs},V_{ds}]$ sino que también hay que tener en cuenta sus derivadas, especialmente si se quiere que el modelo sea utilizado para predecir la distorsión por intermodulación. En [MAANE90], la dependencia $I_{ds}[V_{gs}]$ se modela como una serie armónica, y los coeficientes se ajustan tanto a las curvas $I_{ds}[V_{gs},V_{ds}]$ medidas, como a sus derivadas. Debido a que los modelos anteriores se crearon principalmente para modelar los MESFETs, existe actualmente un aumento de la demanda de modelos para dispositivos FETs en general, que modelen tanto a los MESFETs como a los HFETs. En particular, se

debe modelar de forma correcta el pico característico que presenta la transconductancia de los HFETs en función de la tensión de puerta. En principio se podrían utilizar los modelos propuestos en [CURET85] y [MAANE90]. Sin embargo, tales modelos necesitan normalmente muchos términos y la extracción de parámetros requiere técnicas especiales.

El modelo propuesto por Angelov [ANGEL92] consiste en un modelo nuevo y simple, en donde la extracción de parámetros se puede llevar a cabo mediante simple inspección de las características dc experimentales $I_{ds}[V_{gs}, V_{ds}]$ y $g_m[V_{gs}]$. Este modelo ha sido utilizado con buenos resultados en el modelado de la I_{ds} y sus derivadas en las siguientes estructuras materiales: AlGaAs-GaAs, AlGaAs-InGaAs-GaAs (dopado homogéneamente y simple y doble δ-dopado), AlInAs-GaInAs-InP y MESFET de GaAs.

3.1.2 El Modelo

Al igual que en los modelos anteriores, en el modelo de Angelov la corriente de drenador se expresa de la siguiente forma:

$$I_{ds}\left[V_{gs}, V_{ds}\right] = I_{dA}\left[V_{gs}\right] \cdot I_{dB}\left[V_{ds}\right] \tag{3.1}$$

donde el primer factor sólo depende de la tensión de puerta, y el segundo de la tensión de drenador. El término $I_{dB}[V_{ds}]$ es el mismo que el utilizado en los modelos [CURTI80] y [STANE87].

$$I_{dB}\left[V_{ds}\right] = (1 + \lambda \cdot V_{ds}) \cdot tanh(\alpha \cdot V_{ds}) \tag{3.1.1}$$

Como vemos, la corriente de drenador tendrá una variación con V_{ds} con forma de tangente hiperbólica viéndose afectada su amplitud por la modulación del canal. Sin embargo, el modelo de Angelov propone una nueva función para $I_{dA}[V_{gs}]$, cuya primera derivada tiene la misma forma en "campana" que la función de transconductancia medida $g_m[V_{gs}]$. La tangente hiperbólica describe bien la dependencia con la tensión de puerta y sus derivadas. La función $I_{dA}[V_{gs}]$ propuesta es la siguiente:

$$I_{dA}\left[V_{gs}\right] = I_{pk} \cdot (1 + tanh(\Psi)) \tag{3.1.2}$$

siendo la expresión total para la I_{ds} la siguiente:

$$I_{ds}\left[V_{gs}, V_{ds}\right] = I_{pk} \cdot (1 + tanh(\Psi)) \cdot (1 + \lambda \cdot V_{ds}) \cdot tanh(\alpha \cdot V_{ds}) \tag{3.2}$$

Donde I_{pk} es la corriente de drenador para la cual tenemos una transconductancia máxima, sin tener en cuenta la contribución de la conductancia de salida (es decir, con la contribución de la conductancia de salida restada). λ es el parámetro de modulación del canal y α es el parámetro de la tensión de saturación. Los parámetros α y λ son los mismos que los de los modelos de Statz y Curtice. ψ es en general una función potencial en serie centrada en V_{pk} y con V_{gs} como variable, es decir:

$$\Psi = P_1 \cdot (V_{gs} - V_{pk}) + P_2 \cdot (V_{gs} - V_{pk})^2 + P_3 \cdot (V_{gs} - V_{pk})^3 + \ldots \tag{3.3}$$

V_{pk} es la tensión de puerta para la transconductancia máxima g_{mpk}. La Función $I_{ds}[V_{gs}, V_{ds}]$ seleccionada posee derivadas bien definidas. Una ventaja del modelo es su simplicidad. Los diferentes parámetros pueden ser obtenidos como primera aproximación mediante la inspección de la curva $I_{ds}[V_{gs}, V_{ds}]$ medidas en las siguientes condiciones de canal saturado: se suponen nulos todos los términos superiores de ψ, λ se obtiene de la pendiente de la característica I_{ds}-V_{ds}, I_{pk} y V_{pk} se determinan como los valores de la I_{ds} y la V_{gs} en la transconductancia pico g_{mpk}. La transconductancia intrínseca máxima g_{mpk} se calcula de la transconductancia máxima medida g_{mpkm} teniendo en cuenta el efecto de realimentación debido a la resistencia del surtidor, R_s, la cual se puede obtener de las medidas en dc [HOBLC86]:

$$g_{mpk} = \frac{g_{mpkm}}{(1 - R_s \cdot g_{mpkm})} \tag{3.4}$$

P_1 se obtiene ahora como:

$$P_1 = \frac{g_{mpk}}{I_{pk} \cdot (1 + \lambda \cdot V_d)} \tag{3.5}$$

En algunos HFETs, V_{pk} es ligeramente dependiente con la tensión de drenador V_{ds} en la región de saturación. Este efecto puede ser tenido en cuenta mediante:

$$V_{pk} = V_{pko} + \gamma \cdot V_{ds} \tag{3.6}$$

En la región no saturada y para una V_{ds} negativa, V_{pk} variará de forma considerable con V_{ds}. Si queremos que el modelo prediga el funcionamiento del transistor correctamente, debemos encontrar la dependencia de V_{pk} con V_{ds} (experimentalmente o modelada).

Para el modelado de las dependencias de las capacidades C_{gs} y C_{gd} con la tensión de puerta y de drenador se utiliza el mismo tipo de funciones:

$$C[V_{gs}, V_{ds}] = C_A[tanh(V_{gs})] \cdot C_B[tanh(V_{ds})] \tag{3.7}$$

como se sugiere en [ROKAM90] y [AKFKM84]. Debido a la similitud que existe entre $I_{ds}[V_{gs}, V_{ds}]$ y $C_{gs}[V_{gs}, V_{ds}]$ las funciones se pueden expresar como:

$$C_{gs} = C_{gso} \cdot [1 + tanh(\Psi_1)] \cdot [1 + tanh(\Psi_2)] \tag{3.8}$$

$$C_{gd} = C_{gdo} \cdot [1 + tanh(\Psi_3)] \cdot [1 - tanh(\Psi_4)] \tag{3.9}$$

donde

$$\Psi_1 = P_{0gsg} + P_{1gsg} V_{gs} + P_{2gsg} V_{gs}^2 + P_{3gsg} V_{gs}^3 + \ldots \tag{3.10}$$

$$\Psi_2 = P_{0gsd} + P_{1gsd} V_{ds} + P_{2gsd} V_{ds}^2 + P_{3gsd} V_{ds}^3 + \ldots \tag{3.11}$$

$$\Psi_3 = P_{0gdg} + P_{1gdg} V_{gs} + P_{2gdg} V_{gs}^2 + P_{3gdg} V_{gs}^3 + \ldots \tag{3.12}$$

$$\Psi_4 = P_{0gdd} + (P_{1gdd} + P_{1cc} V_{gs}) V_{ds} + P_{2gdd} V_{ds}^2 + P_{3gdd} V_{ds}^3 + \ldots \tag{3.13}$$

El término $P_{1cc}\ V_{gs}\ V_{ds}$ refleja el acoplamiento cruzado de V_{gs} y V_{gd} sobre C_{gd}. Cuando se considera suficiente una precisión del orden del 5% de C_{gs} y C_{gd}, las ecuaciones (3.8)-(3.13) se pueden simplificar a:

$$C_{gs} = C_{gso} \cdot [1 + tanh(P_{1gsg} V_{gs})] \cdot [1 + tanh(P_{1gsd} V_{ds})] \tag{3.14}$$

$$C_{gd} = C_{gdo} \cdot [1 + tanh(P_{1gdg} V_{gs})] \cdot [1 - tanh(P_{1gdd} V_{ds} + P_{1cc} V_{gs} V_{ds})] \tag{3.15}$$

La ecuación (3.15) se puede simplificar aún más si se desprecia el acoplamiento cruzado para tensiones de drenador grandes ($V_{ds} > 1V$):

$$C_{gd} = C_{gdo} \cdot \left[1 + tanh(P_{1gdg} V_{gs})\right] \cdot \left[1 - tanh(P_{1gdd} V_{ds})\right] \tag{3.16}$$

Las ecuaciones (3.14)-(3.16) son válidas para HFETs δ-dopados con una capa espaciadora de AlGaAs no dopado, ya que estos poseen una característica $C_{gs}[V_{gs}]$ saturada para V_{gs} aumentando debido a la ausencia de formación de canal parásito MESFET en la capa AlGaAs que se encuentra en los HFETs con una capa de AlGaAs dopado.

3.2 Modelo del FhGIAF

El modelo del FhGIAF [EUROC94] fue diseñado en principio para modelar los transistores HFET fabricados en sus laboratorios. Se trata de HFETs de enriquecimiento y deplexión con longitudes de puerta de 0.3 y 0.5 μm. Este modelo se deriva del modelo de Angelov que hemos descrito en el apartado anterior y está implementado en IAFSPICE, el cual es una variación de SPICE2G6 que incluye el modelo del HFET desarrollada por el FhGIAF. La corriente de drenador tiene la misma forma que la ecuación (3.1), en este caso las ecuaciones que describen las componentes I_{dA} e I_{dB} son diferentes. El término I_{dB} ha sido modificado para dar cuenta de la variación de la modulación del canal. De esta forma tenemos que I_{dB} ya no depende solamente de V_{ds} sino que también depende de V_{gs}.

$$I_{dB}[V_{ds}] = \left(1 + \frac{\lambda \cdot V_{ds}}{1 + \Delta\lambda \cdot (V_{gs} - V_{to})^2}\right) \cdot tanh(\alpha \cdot V_{ds}) \tag{3.17}$$

$\Delta\lambda$ es la variación del parámetro de modulación del canal. La tensión umbral, V_{to}, se expresa a través de una relación empírica entre dos parámetros del modelo, pero conserva el mismo significado que en otros tipos de transistores definiendo las distintas regiones de trabajo del dispositivo.

$$V_{to} = V_C - \frac{2}{\beta} \tag{3.18}$$

Los parámetros de modelo V_C y β se describirán más adelante. De la ecuación (3.17) se desprende que al aumentar la tensión de puerta, V_{gs}, la variación de la corriente de drenador con V_{ds} se ve afectada menos por la modulación del canal.

Sin embargo, el cambio más significativo propuesto por el FhGIAF respecto al modelo de Angelov lo constituye el realizado con el término I_{dA}. La ecuación propuesta consiste en sumarle al término I_{dA} dado por Angelov otro término similar. Así tenemos:

$$I_{dA}\left[V_{gs}\right] = I_{dA1}\left[V_{gs}\right] + I_{dA2}\left[V_{gs}\right] = I_{dVC}(1+tanh(\Psi_1)) + I_{dVSB}(1+tanh(\Psi_2)) \quad (3.19)$$

I_{dVC} y ψ_1 actúan de forma similar a I_{pk} y ψ del modelo de Angelov, es decir, modelan el primer pico de la transconductancia (ver figura 3.1). En este caso se toman solamente los términos de orden 1 y 3 de la función potencial ψ_1. Por tanto I_{dA1} queda:

$$I_{dA1}\left[V_{gs}\right] = I_{dVC}(1+tanh(\beta\cdot(V_{gs}-V_C)+\gamma\cdot(V_{gs}-V_C)^3)) \quad (3.20)$$

donde β y γ son los parámetros P_1 y P_3 del modelo de Angelov, I_{dVC} equivale a la I_{pk} y V_C es la V_{pk}. El término I_{dA2} corresponde al punto de transconductancia mínima, es decir, modela el segundo pico de la transconductancia (ver figura 3.1). Así, I_{dVSB} es la corriente de drenador para la cual tenemos una transconductancia mínima, y ψ es en general una función potencial en serie centrada en V_{SB} (V_{gs} para transconductancia mínima) y con V_{gs} como variable. De esta forma y tomando sólo el primer término de la serie tenemos:

$$I_{dA2}\left[V_{gs}\right] = I_{dVSB}(1+tanh(\delta\cdot(V_{gs}-V_{SB}))) \quad (3.21)$$

δ es el parámetro P_1 de la sucesión. En la mayoría de los casos y sobre todo en los HFETs de enriquecimiento, el pico negativo de la transconductancia está muy alejado del positivo, o es muy poco pronunciado. En estos casos se puede despreciar su efecto sobre la I_d con lo que los parámetros I_{dVSB} y δ toman valores nulos.

La ecuación que modela la corriente de drenador queda finalmente como:

$$I_{ds} = f_1(V_{gs}) \cdot f_2(V_{ds}, V_{gs}) \cdot f_3(V_{ds}) \tag{3.22}$$

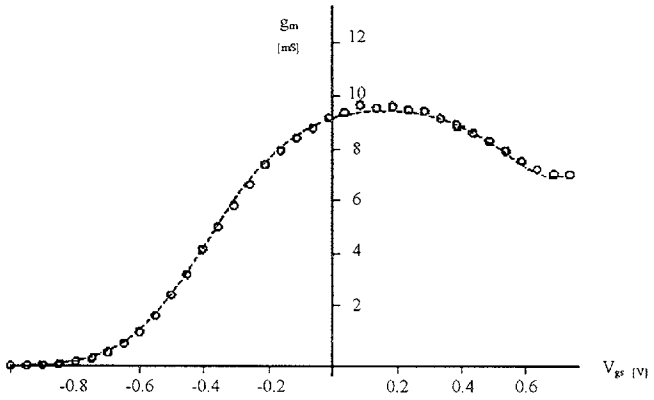


fig. 3.1 Transconductancia de un HFET de deplexión de 0.5 μm x 25 μm, con V_{ds}=1.5 V, V_c= 0.13 V, y β=2.15. Línea continua: medida; línea de puntos: IAFSPICE.

donde:

$$\begin{aligned} f_1[V_{gs}] &= I_{dVC}(1 + tanh(\beta \cdot (V_{gs} - V_C) + \gamma \cdot (V_{gs} - V_C)^3)) \\ &+ I_{dVSB}(1 + tanh(\delta \cdot (V_{gs} - V_{SB}))) \end{aligned} \tag{3.23}$$

$$f_2[V_{ds}, V_{gs}] = \left(1 + \frac{\lambda \cdot V_{ds}}{1 + \Delta\lambda \cdot (V_{gs} - V_{to})^2}\right) \tag{3.24}$$

$$f_3[V_{ds}] = tanh(\alpha \cdot V_{ds}) \tag{3.25}$$

El modelado de las dependencias de las capacidades C_{gs} y C_{gd} con la tensión de puerta y de drenador se hace utilizando el mismo esquema propuesto por Angelov. De esta manera, la capacidad C_{gs} se define para las diferentes regiones de trabajo del transistor como:

$$C_{gs} = C_{gso} + f_{c1}(V_{gs}) \cdot f_{c2}(V_{ds}) \tag{3.26}$$

donde C_{gso} es la capacidad para alimentación nula y f_{c1} y f_{c2} son las funciones descritas a continuación para tres regiones de trabajo: por debajo del *pinch-off* (V_{gs}<V_{at}), el canal en conducción (V_{gs}≥V_{bt}) y una región intermedia entre ambas (V_{at}≤V_{gs}<V_{bt}).

Para $V_{gs}<V_{at}$

$$\begin{cases} f_{c1} = 0 \\ f_{c2} = 0 \end{cases} \tag{3.27}$$

Para $V_{at}\leq V_{gs}<V_{bt}$

$$\begin{cases} f_{c1} = C_{gs1} \cdot \dfrac{V_{gs} - V_{at}}{V_{bt} - V_{at}} \\ f_{c2} = 1 + V_{st} \cdot tanh(\alpha \cdot V_s \cdot V_{ds}) \end{cases} \tag{3.28}$$

Para $V_{gs}\geq V_{bt}$

$$\begin{cases} f_{c1} = C_{gs1} \cdot \left(1 + C_f (V_{gs} - V_{bt})^m\right) \\ f_{c2} = 1 + V_{st} \cdot tanh(\alpha \cdot V_s \cdot V_{ds}) \end{cases} \tag{3.29}$$

Para el caso de la C_{gd}, sólo hay que cambiar V_{gs} por V_{gd} e invertir el signo de V_{ds}.

3.2.1 Adaptación del Modelo FhGIAF a SPICE

Una vez definidos los elementos básicos del modelo, el siguiente paso es su adaptación para poder incluirlo en SPICE. Esta adaptación consiste en definir los circuitos equivalentes del transistor para su modelado en dc, ac, y régimen transitorio. Además se deberán definir los elementos parásitos que se han de incluir en dichos circuitos equivalentes para que el transistor quede modelado de forma completa.

3.2.1.1 Modelo en Gran Señal

El modelado en gran señal comprende dos aspectos, el comportamiento estático o dc, y el comportamiento dinámico o transitorio.

3.2.1.2 Modelo Equivalente en dc

El modelo dc para los HFETs es el que se muestra en la figura 3.2. El valor de la corriente de drenador, representado como una fuente de corriente, es el dado por la ecuación (3.22). En las figuras 3.3 y 3.4 se muestran la transconductancia y las curvas I_{ds}-V_{gs} e I_{ds}-V_{ds} para un transistor de deplexión de 0.5 μm. Como se puede observar, para tensiones de puerta elevadas, aparece un mínimo en la transconductancia lo cual justifica la inclusión de I_{dA2} en la

corriente de drenador. El modelo FhGIAF presenta una mayor exactitud para los transistores de deplexión que para los de enriquecimiento.

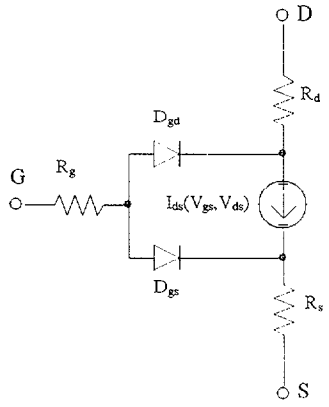


fig. 3.2 Circuito equivalente dc.

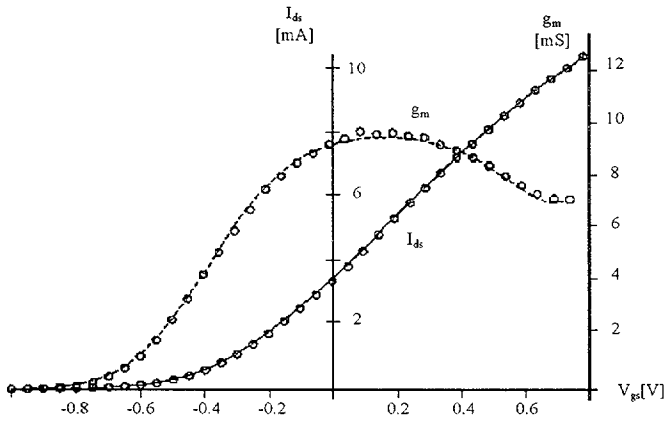


fig. 3.3 Transconductancia y Corriente de Drenador de un HFET de deplexión de 0.5 μm x 25 μm, con V_{ds}=1.5 V, V_c= 0.13 V, y β=2.15. Línea continua: medida; línea de puntos: IAFSPICE.

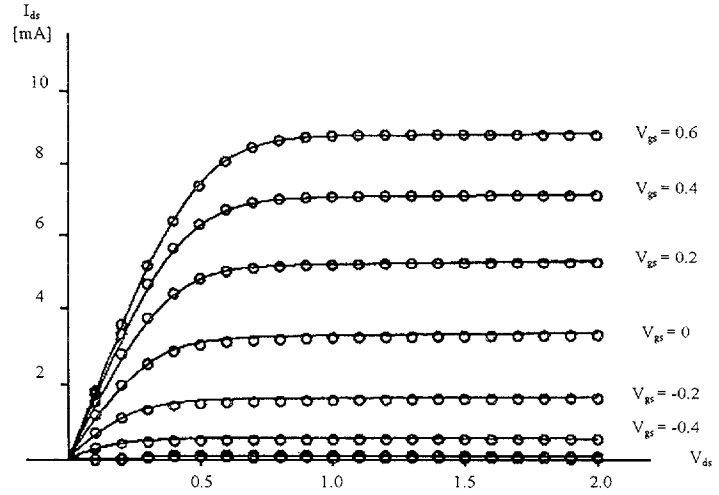


fig. 3.4 Curva I-V de un HFET de deplexión de 0.5 μm x 25 μm, con V_c= 0.13 V, y β=2.15. Línea continua: medida; línea de puntos: IAFSPICE.

La corriente de puerta se modela a través de dos diodos conectados entre la puerta y la fuente y entre la puerta y el drenador. Estos diodos se modelan para tensiones de puerta positivas a través de la ecuación típica de los diodos, es decir:

$$I_g = I_{so} \cdot \left(e^{\frac{q \cdot V_g}{n \cdot K \cdot T}} - 1 \right) \tag{3.30}$$

donde I_{so} es la corriente de saturación del diodo que es igual tanto para I_{gs} como para I_{gd} al ser el transistor simétrico respecto a la puerta. Sin embargo, para tensiones de puerta negativas, especialmente con longitudes de puerta por debajo de la micra, aparecen corrientes de fuga debidas tanto a corrientes residuales desde el substrato, como al efecto túnel cerca de la superficie. En este caso, la corriente de puerta se modela como:

$$I_g = -I_{so} \cdot A \cdot \left(e^{\frac{q \cdot \sqrt{|B V_g|}}{n \cdot K \cdot T}} - 1 \right) \tag{3.31}$$

Los parámetros empíricos A y B dan cuenta de los fenómenos que hemos mencionado. Como se aprecia en la figura 3.5, el uso de este modelo en las dos regiones para los diodos permite obtener una buena aproximación para las corrientes de puerta para $V_{ds} = 0$. Para $V_{ds} > 0$, las variaciones en las corrientes de puerta pueden ser despreciadas.

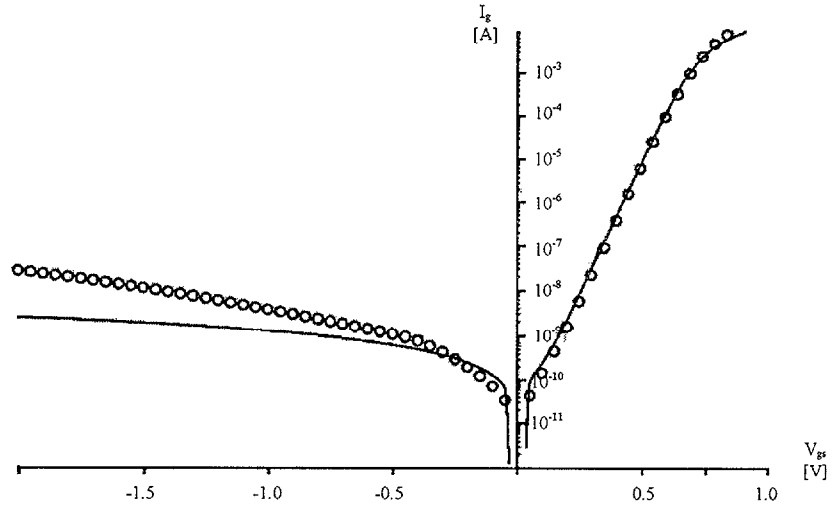


fig. 3.5 Corriente de Puerta frente a V_{gs} de un HFET de deplexión de 0.5 μm x 25 μm, con V_{ds}=0, V_c= 0.13 V, y β=2.15. Línea continua: medida; línea de puntos: IAFSPICE.

Por último, en el circuito equivalente en dc encontramos las resistencias parásitas que conectan el dispositivo intrínseco con los terminales externos. Nótese la inclusión de la resistencia de puerta, R_g, la cual no aparece en los MESFETs ni en los MOSFETs debido al material de puerta altamente conductivo que poseen. En el caso de los HFETs, no se puede despreciar su resistencia aunque sea mucho menor que las resistencias de drenador y de surtidor.

3.2.1.3 Modelo en Régimen Transitorio

El comportamiento dinámico está regido por los cambios que se producen en la carga almacenada en las distintas regiones del dispositivo al variar la tensión. El caso cuasiestático puede modelarse considerando por separado el comportamiento intrínseco y el extrínseco. El circuito equivalente del HFET para régimen dinámico es el que se muestra en la figura 3.6.

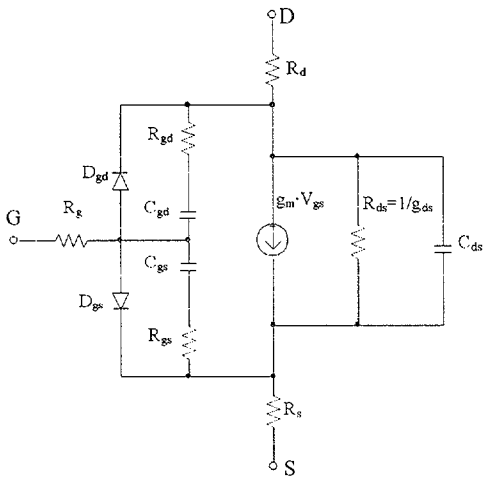


fig. 3.6 Circuito equivalente para régimen dinámico.

El comportamiento extrínseco se modela mediante condensadores no lineales. En este caso, esos condensadores son los descritos en el apartado 3.2 como C_{gs} y C_{gd}. En serie con estas capacidades nos encontramos con las resistencias R_{gs} (también llamada R_i) y R_{gd} las cuales, aunque aparecen en el modelo, no están implementadas en IAFSPICE. Las conductancias de puerta a drenador, g_{gd}, y de puerta a surtidor, g_{gs}, se definen como las derivadas de las corrientes de puerta respectivas. Así, para el caso de la conductancia de puerta a surtidor tenemos:

para $V_{gs} \geq 0$

$$g_{gs} = \frac{\partial I_{gs}}{\partial V_{gs}} = I_{so} \cdot \frac{q \cdot e^{\frac{q \cdot V_{gs}}{n \cdot K \cdot T}}}{n \cdot K \cdot T} \tag{3.32}$$

para $V_{gs} < 0$

$$g_{gs} = \frac{\partial I_{gs}}{\partial V_{gs}} = I_{so} \cdot A \cdot \frac{q \cdot B \cdot e^{\frac{q \cdot \sqrt{|B \cdot V_{gs}|}}{n \cdot K \cdot T}}}{2 \cdot n \cdot K \cdot T \cdot \sqrt{|B \cdot V_{gs}|}} \tag{3.33}$$

Para el caso de la conductancia de puerta a drenador las ecuaciones son las mismas cambiando V_{gs} por V_{gd}.

En el circuito equivalente para régimen dinámico aparece también la capacidad de drenador a surtidor, C_{ds}, conectada en paralelo a la resistencia R_{ds}. El valor de C_{ds} es normalmente pequeño comparado con los valores de C_{gs} y C_{gd}. Además, su valor se considera que no varía con la tensión. La resistencia de drenador a surtidor, R_{ds}, se define como la inversa de la conductancia g_{ds}. Tanto la transconductancia, g_m, como la conductancia de drenador a surtidor, g_{ds}, vienen definidas como las derivadas parciales de la corriente de drenador en un punto de polarización Q respecto a las tensiones V_{gs} y V_{ds} respectivamente, es decir:

$$I_{ds}[V_{gs}, V_{ds}] = I_Q + \left.\frac{\partial I_Q}{\partial V_{gs}}\right|_Q + \left.\frac{\partial I_Q}{\partial V_{ds}}\right|_Q = I_Q + g_m + g_{ds} \tag{3.34}$$

Aplicando estas definiciones obtenemos las siguientes expresiones para g_m y para g_{ds}:

$$g_m = f_3 \cdot \left\{ f_2 \cdot \left[I_{dVC}\left(\frac{\beta + \gamma \cdot 3 \cdot (V_{gs} - V_C)^2}{\cosh^2(\beta \cdot (V_{gs} - V_C) + \gamma \cdot (V_{gs} - V_C)^3)} \right) + I_{dVSB}\left(\frac{\delta}{\cosh^2(\delta \cdot (V_{gs} - V_{SB}))} \right) \right] \right.$$
$$\left. - f_1 \cdot \left[\frac{V_{ds} \cdot \lambda \cdot \Delta\lambda \cdot 2 \cdot (V_{gs} - V_{to})}{(1 + \Delta\lambda \cdot (V_{gs} - V_{to})^2)^2} \right] \right\} \tag{3.35}$$

$$g_{ds} = f_1 \left[\left(\frac{\lambda}{1 + \Delta\lambda \cdot (V_{gs} - V_{to})^2} \right) \cdot tanh(\alpha \cdot V_{ds}) + \left(1 + \frac{\lambda \cdot V_{ds}}{1 + \Delta\lambda \cdot (V_{gs} - V_{to})^2} \right) \cdot \frac{\alpha}{\cosh^2(\alpha \cdot V_{ds})} \right] \tag{3.36}$$

Para $V_{ds} < 0$, debemos cambiar el signo de dicha tensión en la ec. (3.22) y volver a calcular las expresiones de g_m y g_{ds} tal y como lo acabamos de hacer.

El comportamiento intrínseco se modela teniendo en cuenta la carga almacenada en la estructura del HFET, es decir, la carga almacenada en las capacidades C_{gs} y C_{gd}. Por tanto, la carga almacenada en las capacidades C_{gs} y C_{gd} quedarán definidas como su integral respecto a V_{gs} y V_{gd} respectivamente. Así, para las tres regiones para las que están definidas las capacidades de puerta tenemos las siguientes expresiones:

Para $V_{gs}<V_{at}$
$$Q_{gs} = C_{gs0} \cdot V_{gs} + Q_{c1} \tag{3.37}$$

Para $V_{at} \leq V_{gs} < V_{bt}$
$$Q_{gs} = C_{gs0} \cdot V_{gs} + C_{gs1} \frac{(V_{gs} - V_{at})^2}{2 \cdot (V_{bt} - V_{at})} \cdot f_{c2} + Q_{c2} \tag{3.38}$$

Para $V_{gs} \geq V_{bt}$
$$Q_{gs} = C_{gs0} \cdot V_{gs} + C_{gs1} \cdot \left(V_{gs} + C_f \cdot \frac{(V_{gs} - V_{bt})^{m+1}}{m+1} \right) \cdot f_{c2} + Q_{c3} \tag{3.39}$$

Para Q_{gd} las expresiones son las mismas cambiando V_{gs} por V_{gd} e invirtiendo el signo de V_{ds}. Los valores de Q_{c1}, Q_{c2} y Q_{c3} deben ser definidos para respetar las condiciones de continuidad de la carga y de sus derivadas en las interfases entre las tres regiones [M89/42]. De no ser así, podríamos tener problemas de convergencia a la hora de simular con SPICE. Por tanto, para $V_{gs} = V_{at}$ se deberá cumplir

$$Q_1 = Q_2,\ C_1 = C_2,\ \frac{\partial C_1}{\partial V_{gs}} = \frac{\partial C_2}{\partial V_{gs}} \tag{3.40}$$

donde los subíndices 1 y 2 indican las regiones anterior y posterior de la interfase. Las condiciones a cumplir en la otra interfase, $V_{gs} = V_{bt}$, son:

$$Q_2 = Q_3,\ C_2 = C_3,\ \frac{\partial C_2}{\partial V_{gs}} = \frac{\partial C_3}{\partial V_{gs}} \tag{3.41}$$

donde los subíndices 2 y 3 indican aquí igualmente las regiones anterior y posterior de la interfase. Aplicando estas condiciones obtenemos los valores de Q_{c1}, Q_{c2} y Q_{c3}:

$$Q_{c2} = Q_{c1} \tag{3.42}$$

$$Q_{c3} = Q_{c2} - C_{gs1} \cdot f_{c2} \frac{(V_{bt} + V_{at})}{2} \tag{3.43}$$

Como vemos, las cargas Q_{c2} y Q_{c3} dependen de Q_{c1}. Desde un punto de vista físico, esta carga da cuenta de la carga residual que tiene el dispositivo que se supone nula.

3.2.1.4 Modelo en Pequeña Señal

Como sabemos, el análisis en ac de SPICE se hace bajo la suposición de pequeña señal. Esto se debe a que sólo bajo estas condiciones, los circuitos no lineales, tales como los constituidos por transistores, se pueden reemplazar por su equivalente lineal alrededor del punto de operación dc.

El modelo en pequeña señal se divide en una parte intrínseca y en otra extrínseca. En la figura 3.7.a se muestra el circuito equivalente en pequeña señal completo. El modelo intrínseco corresponde al encerrado por la línea discontinua. El elemento principal lo constituye la fuente de corriente controlada por tensión de valor $I_{ds} = \hat{g}_m \cdot V_{gs}$. La expresión para la transconductancia modificada $\hat{g}_m$ es la siguiente:

$$\hat{g}_m = g_m \cdot e^{-j\Omega\tau} = g_m \cdot (\cos\Omega\tau - j\,\mathrm{sen}\,\Omega\tau) \text{ , con } \Omega = 2 \cdot \pi \cdot frec \qquad (3.44)$$

donde g_m es la transconductancia y τ es un exceso de fase similar al parámetro *ptf* de los transistores bipolares.

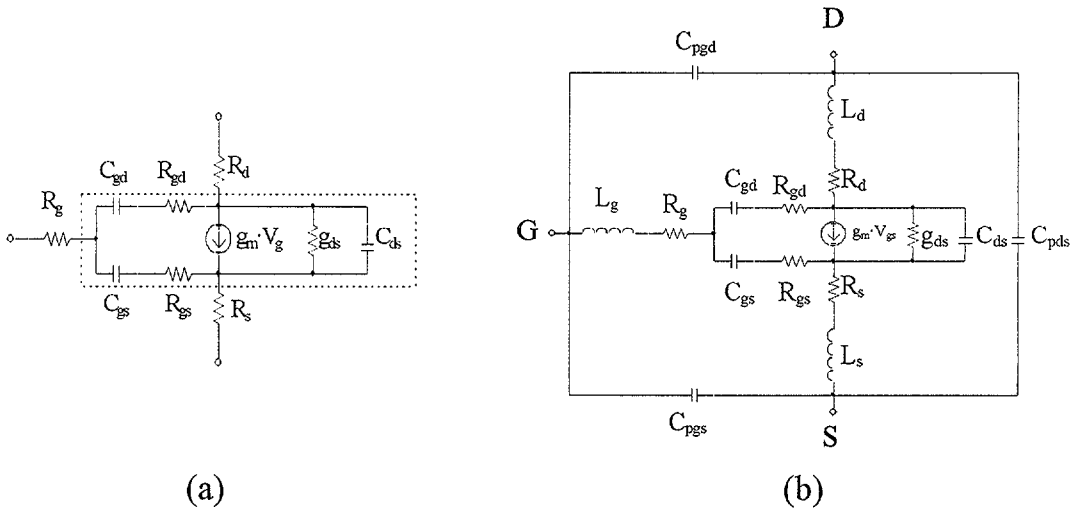


fig. 3.7 Circuito equivalente en pequeña señal: (a) completo, (b) simplificado.

La parte extrínseca del modelo corresponde a las resistencias de los terminales. Para poder hacer pruebas con transistores reales y poder comparar los resultados con los del modelo, debemos añadir el efecto de los PADs. Esto se hace incluyendo tres bobinas y tres condensadores tal y como se muestra en la figura 3.7.b. Estos elementos no forman parte del modelo del transistor y por tanto se deben incluir como elementos externos dentro del fichero de entrada de SPICE.

3.2.1.5 Lista de Parámetros

Como conclusión, en este capítulo se han presentado los elementos adecuados para simular las características más relevantes de los HFETs. En la tabla 3-1 se recoge una lista de los parámetros del modelo implementado. En ella aparecen los nombres de los parámetros dados tanto en el modelo, como en SPICE e internamente en el código fuente, junto con las unidades respectivas.

Tabla 3-1 Parámetros SPICE de los transistores HFETs.

Nombre en el Modelo	Nombre en SPICE	Nombre Interno	Significado	Unid.
I_{GSS}	IS	HEMTgateSatCurrent	Corriente de saturación de los diodos de puerta	A/μm
n	NP	HEMTnp	Coeficiente de emisión o de no idealidad	-
A	AFACT	HEMTafact	Parámetro de ajuste A	-
B	GEXP	HEMTgexp	Parámetro de ajuste B	V^{-1}
CD_{VC}	CDVC	HEMTcdvc	Corriente de drenador para transconductancia máxima	A/μm
β	BETA	HEMTbeta	Parámetro de ajuste β	V^{-1}
V_C	VC	HEMTvc	Tensión de puerta para transconductancia máxima	V
γ	GAMMA	HEMTgamma	Parámetro de ajuste γ	V^{-3}
CD_{VSB}	CDVSB	HEMTcdvsb	Corriente de drenador para transconductancia mínima	A/μm
δ	DELTA	HEMTdelta	Parámetro de ajuste δ	V^{-1}
V_{SB}	VSB	HEMTvsb	Tensión de puerta para transconductancia máxima	V
λ	LAMBDA	HEMTlModulation	Parámetro de modulación del canal	V^{-1}
Δ_λ	DXL	HEMTdxl	Variación del parámetro de modulación del canal	V^{-2}
α	ALPHA	HEMTalpha	Parámetro de tensión de saturación	V^{-1}
V_{at}	VAT	HEMTvat	Tensión de pinch-off	V
V_{bt}	VBT	HEMTvbt	Tensión de conducción	V
C_{gs0}	CGS0	HEMTcgs0	Capacidad para alimentación nula	F/μm
C_{gs1}	CGS1	HEMTcgs1	Factor de capacidad para alimentación nula	F/μm
V_{st}	VST	HEMTvst	Parámetro de ajuste V_{st}	-
V_s	VS	HEMTvs	Parámetro de ajuste V_s	-
C_f	FC	HEMTdepletionCapCoeff	Parámetro de ajuste C_f	V^{-m}
m	MFACT	HEMTmfact	Parámetro de ajuste m	-
R_d	RD	HEMTdrainResist	Resistencia de drenador	Ω μm
R_s	RS	HEMTsourceResist	Resistencia de surtidor	Ω μm
R_g	RG	HEMTrg	Resistencia de puerta	Ω/μm
R_{gs}	RGS	HEMTrgs	Resistencia pueta-surtidor	Ω μm
R_{gd}	RGD	HEMTrgd	Resistencia puerta-drenador	Ω μm
Q_{c1}	QC1	HEMTqc1	Carga residual	-
τ	TD	HEMTtd	Exceso de fase	rad
C_{ds}	CDS	HEMTcds	Capacidad drenador-surtidor	F/μm

Capítulo 4

4. Algoritmos de SPICE

En la actualidad existe un considerable número de algoritmos para la solución de ecuaciones de circuitos eléctricos que han demostrado su valía para este propósito. Estos algoritmos no sólo están implementados en los simuladores SPICE, sino en la mayor parte de los simuladores circuitales que existen en la actualidad.

El proceso de resolución implementado en SPICE en el dominio del tiempo se muestra en la figura 4.1. Generalmente el programa comienza resolviéndose para un punto de operación dc estable. Esta resolución se inicia con una suposición del punto de operación, la cual viene seguida por una serie de iteraciones con objeto de determinar la solución de las ecuaciones no lineales dc. Este proceso iterativo corresponde al bucle interno de la figura 4.1. Cuando el proceso iterativo converge, el resultado obtenido puede corresponder a la solución en pequeña señal (SSBS, *small signal bias solution*) o a la solución transitoria inicial (ITS, *initial transient solution*). Esta es la solución para el instante de tiempo cero. El proceso iterativo se repite para cada punto en el tiempo en los cuales las ecuaciones circuitales han de ser resueltas en el análisis transitorio. En las secciones 4.1 y 4.2 se presentan de forma respectiva los algoritmos utilizados en SPICE para las soluciones dc de circuitos lineales y no lineales.

La solución en el dominio del tiempo, representada por el bucle exterior de la figura 4.1, necesita la participación de un algoritmo de integración numérica para transformar el conjunto de ecuaciones diferenciales ordinarias en un conjunto de ecuaciones no lineales. El análisis en el dominio del tiempo se

reemplaza, por tanto, por una secuencia de soluciones cuasi estáticas. Estas técnicas se describirán en la sección 4.3.

Un simulador circuital queda definido por la siguiente secuencia de algoritmos específicos; en primer lugar, un método de integración numérica implícito que transforme las ecuaciones diferenciales no lineales en ecuaciones algebraicas no lineales, en segundo lugar, un algoritmo iterativo de Newton-Raphson modificado para la linealización de éstas, y finalmente un algoritmo de eliminación Gaussiana y técnicas de matrices dispersas (*sparse*) que resuelvan las ecuaciones lineales. Los simuladores que utilizan estas técnicas, tales como SPICE2 y SPICE3 y sus derivados, se conocen como simuladores circuitales de tercera generación. La solución adoptada en estos simuladores se conoce también como método directo. Estos algoritmos se describen en detalle en los trabajos de McMalla [MCCAL88] y Nagel [NAGEL75] y en otros artículos más generales como los de McCalla y Pederson [MCCPE71] y los de Hachtel y Sangiovanni-Vicentelli [HACSA81].

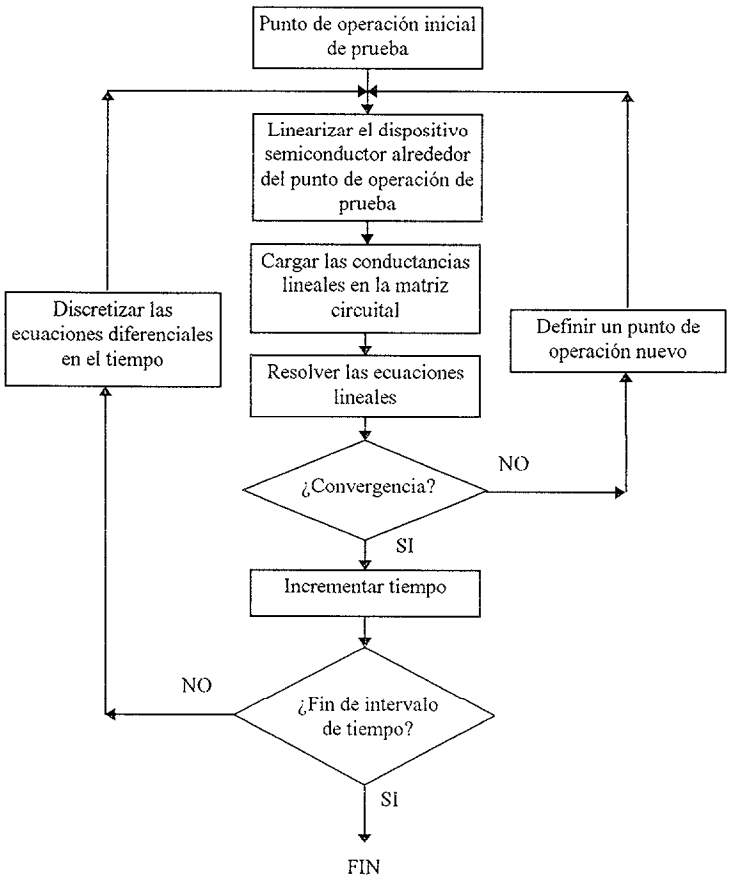


fig. 4.1 Algoritmo de resolución de SPICE.

4.1 Solución en dc de Circuitos Lineales

Esta sección describe cómo se formulan las ecuaciones circuitales así como los algoritmos de resolución para los sistemas lineales. Los algoritmos de resolución de las ecuaciones lineales que se describen aquí se utilizan para la resolución en dc de circuitos lineales y para la resolución en ac, la cual implica siempre circuitos lineales. Las ecuaciones dc se formulan con números reales, y las ecuaciones ac utilizan números complejos.

La formulación matricial para la conectividad y las ecuaciones nodales en SPICE se presenta en la sección 4.1.1 junto con el algoritmo de resolución. Una versión extendida del análisis nodal ([DORF89]; [NILSS90]; [PAUL89]), denominado análisis nodal modificado (MNA, *modified nodal analysis*) es la que utiliza SPICE para representar los circuitos. La eliminación Gaussiana y la factorización asociada en matrices triangulares inferiores y superiores, la factorización LU, se implementa en SPICE para la solución de un sistema lineal de ecuaciones simultáneas. En esta sección se describe también una característica importante de la matriz de admitancia de un circuito, su dispersión (*sparsity*), así como su impacto en el análisis.

Ciertas características de la formulación MNA y la eliminación Gaussiana junto con las limitaciones de los ordenadores para representar números reales puede producir una pérdida de precisión y una solución errónea. Estos inconvenientes se detallan en la sección 4.1.2. En este punto se detalla también un elemento importante: la reordenación de las ecuaciones para aumentar la precisión y la dispersión (*sparsity*).

4.1.1 Formulación de las Ecuaciones Circuitales: Ecuaciones Nodales Modificadas

Supongamos el circuito puente-T de la figura 4.2. Su solución en dc se deriva de las ecuaciones nodales escritas para cada nodo. En la evaluación de las tensiones de los nodos, el valor de la fuente de tensión puesta a tierra, V_{BIAS}, se asigna, por inspección , a V_1, es decir la tensión del nodo 1. De esta manera, sólo se deben resolver dos ecuaciones nodales en dos puntos desconocidos:

$$\begin{array}{lrrrl} nodo2: & -G_1 \cdot V_1 & +(G_1+G_2+G_3)\cdot V_2 & -G_3 \cdot V_3 & = 0 \\ nodo3: & -G_4 \cdot V_1 & -G_3 \cdot V_2 & +(G_3+G_4)\cdot V_3 & = 0 \end{array} \qquad (4.1)$$

La ec. (4.1) se puede expresar como una ecuación de matrices:

$$G \cdot V = I \tag{4.2}$$

donde G es la matriz de conductancia del circuito, V es el vector de tensiones de nodos desconocidas, e I es el vector de corriente RHS (RHS, *right hand side*). La representación en forma de matrices y vectores se ajusta bien para la programación y, por tanto, es la metodología que se utiliza en SPICE. La matriz de conductancia, G, se calcula fácilmente sumando a cada término diagonal todas las conductancias que inciden en un nodo y restando a los correspondientes términos fuera de diagonal las conductancias que conectan dos nodos.

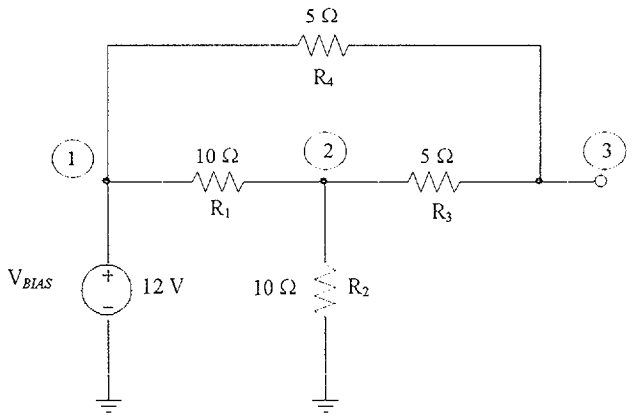


fig. 4.2 Circuito puente-T.

Un problema en la formulación de la matriz de conductancias, G, es que no se puede incluir fácilmente una fuente de tensión en el conjunto de las ecuaciones nodales: la conductancia de una fuente de tensión ideal es infinita, y su corriente es desconocida. Una fuente de tensión conectada entre dos nodos de un circuito complica aún más esta labor e introduce la necesidad de una formulación consistente apropiada para la programación. Un ejemplo de este problema lo constituye el circuito puente-T modificado, el cual se muestra en la figura 4.2, y contiene una fuente de tensión adicional, V_A, en serie con R_4.

Este problema hizo que los desarrolladores de SPICE ampliaran el conjunto de ecuaciones nodales para incluir las ecuaciones de las fuentes de

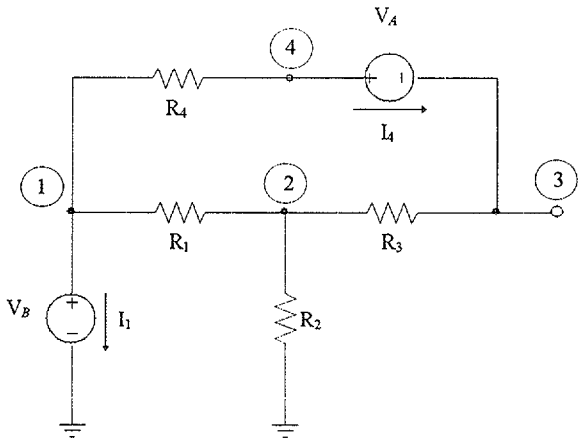


fig. 4.3 Circuito puente-T modificado.

tensión representadas por corrientes en el vector desconocido y por tensiones en el vector RHS. Esta aproximación, denominada análisis nodal modificado, es por tanto una extensión del análisis nodal en el que las ecuaciones de las tensiones de cada nodo se amplían con las ecuaciones de corriente para los elementos de tensión definida [NAGRO71].

El conjunto completo de ecuaciones para el circuito puente-T de la figura 4.2 se puede expresar ahora incluyendo las fuentes de tensión:

$$\begin{array}{rccccccc}
nodo1: & (G_1+G_4)V_1 & -G_1V_2 & & -G_4V_4 & +I_1 & = 0 \\
nodo2: & -G_1V_1 & +(G_1+G_2+G_3)V_2 & -G_3V_3 & & & = 0 \\
nodo3: & & -G_3V_2 & +G_3V_3 & & -I_4 & = 0 \\
nodo4: & -G_4V_1 & & & +G_4V_4 & +I_4 & = 0 \\
V_B: & V_1 & & & & & = V_B \\
V_A: & & & -V_3 & +V_4 & & = V_A
\end{array} \qquad (4.3)$$

Las ecuaciones anteriores se pueden expresar en forma de matrices como sigue:

$$\left[\begin{array}{cccc|cc}
G_1+G_4 & -G_1 & 0 & -G_4 & 1 & 0 \\
-G_1 & G_1+G_2+G_3 & -G_3 & 0 & 0 & 0 \\
0 & -G_3 & G_3 & 0 & 0 & -1 \\
-G_4 & 0 & 0 & +G_4 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0
\end{array}\right]
\cdot
\left[\begin{array}{c}
V_1 \\ V_2 \\ V_3 \\ V_4 \\ \hline I_1 \\ I_4
\end{array}\right]
=
\left[\begin{array}{c}
0 \\ 0 \\ 0 \\ 0 \\ \hline V_B \\ V_A
\end{array}\right] \qquad (4.4)$$

Las ecuaciones MNA anteriores se pueden reescribir de forma abreviada:

$$\begin{bmatrix} G & F \\ B & R \end{bmatrix} \cdot \begin{bmatrix} V \\ I \end{bmatrix} = \begin{bmatrix} C \\ E \end{bmatrix}$$

donde C y E son los vectores de las fuentes de corriente y tensión respectivamente.

Hasta ahora la discusión ha tratado solamente de las dificultades de incluir las fuentes de tensión en un conjunto de ecuaciones nodales. Otro elemento circuital que presenta el mismo problema es el inductor o bobina, ya que éste es un corto en dc, y, por tanto, la tensión que cae en él es cero y la conductancia es infinita. El inductor es también en SPICE un elemento definido en tensión y se incluye como una ecuación de corriente en la formulación MNA. El número total de ecuaciones, N, utilizadas para representar un circuito en SPICE es

$$N = n + n_v + n_l \tag{4.5}$$

donde n es el número de nodos del circuito sin incluir el de tierra, n_v es el número de fuentes de tensión independientes, y n_l es el número de bobinas. Nótese que todas las fuentes controladas, excepto las fuentes de corriente controladas por tensión (VCCS, *voltage controled current sources*), que son una transconductancia, introducen ecuaciones de corriente. La representación matricial MNA de los diferentes elementos se detalla en el libro de McCalla [MCCAL88].

El conjunto de ecuaciones MNA que debemos resolver, ec. (4.4), se puede expresar en forma de ecuación matricial:

$$A \cdot x = b \tag{4.6}$$

Donde A es la matriz de conductancias y b es el RHS. El vector solución, x, se puede calcular invirtiendo la matriz A. Esta solución consume mucho tiempo, por lo que se prefiere el método de la eliminación Gaussiana [FORMO67] para la resolución numérica.

El método de la eliminación Gaussiana utiliza el escalado de cada ecuación seguido de la resta de las ecuaciones restantes con objeto de eliminar los elementos desconocidos por otros hasta que la matriz A queda reducida a una matriz triangular superior. Por tanto, se puede obtener la solución calculando cada elemento del vector x en orden inverso (sustitución hacia atrás). A continuación se describen los pasos que debemos dar para llegar a la solución para un sistema de ecuaciones lineales de 3x3,

$$\begin{matrix} e_1^{(0)}: \\ e_2^{(0)}: \\ e_3^{(0)}: \end{matrix}\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & a_{23}^{(0)} \\ a_{31}^{(0)} & a_{32}^{(0)} & a_{33}^{(0)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ b_3^{(0)} \end{bmatrix} \tag{4.7}$$

Las ecuaciones están designadas por $e_1^{(0)}$, $e_2^{(0)}$, y $e_3^{(0)}$. Los superíndices tanto de los designadores de las ecuaciones como de los elementos de las matrices, representan el paso en el que nos encontramos en el proceso de eliminación. En primer lugar, x_1 se puede eliminar de $e_2^{(0)}$, y $e_3^{(0)}$ restando $e_1^{(0)}$ multiplicado por $a_{21}^{(0)}/a_{11}^{(0)}$ a $e_2^{(0)}$ y restando $e_1^{(0)}$ multiplicado por $a_{31}^{(0)}/a_{11}^{(0)}$ a $e_3^{(0)}$:

$$\begin{aligned} e_1^{(1)} &= e_1^{(0)} \\ e_2^{(1)} &= e_2^{(0)} - (a_{21}^{(0)} / a_{11}^{(0)}) e_1^{(0)} \\ e_3^{(1)} &= e_3^{(0)} - (a_{31}^{(0)} / a_{11}^{(0)}) e_1^{(0)} \end{aligned} \tag{4.8}$$

En segundo lugar, x_2 se puede eliminar de $e_3^{(1)}$ restando $e_2^{(1)}$ multiplicado por $a_{32}^{(1)}/a_{22}^{(1)}$ a $e_3^{(1)}$:

$$\begin{aligned} e_1^{(2)} &= e_1^{(1)} \\ e_2^{(2)} &= e_2^{(1)} \\ e_3^{(2)} &= e_3^{(1)} - (a_{32}^{(1)} / a_{22}^{(1)}) e_2^{(1)} \end{aligned} \tag{4.9}$$

lo cual da como resultado un sistema de ecuaciones triangular superior:

$$\begin{matrix} e_1^{(2)}: \\ e_2^{(2)}: \\ e_3^{(2)}: \end{matrix}\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & 0 & a_{33}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(1)} \\ b_3^{(2)} \end{bmatrix} \tag{4.10}$$

En tercer lugar, la sustitución hacia atrás produce la siguiente solución:

$$
\begin{aligned}
x_3 &= b_3^{(2)} / a_{33}^{(2)} \\
x_2 &= (b_2^{(1)} - a_{23}^{(1)} x_3) / a_{22}^{(1)} \\
x_1 &= (b_1^{(0)} - a_{13}^{(0)} x_3 - a_{12}^{(0)} x_2) / a_{11}^{(0)}
\end{aligned}
\tag{4.11}
$$

Una variante de la eliminación Gaussiana es la *factorización LU*. Este procedimiento transforma la matriz circuital A en una matriz triangular inferior, L, y otra superior, U. La ecuación (4.6) se puede reescribir de la siguiente manera

$$LUx = b \tag{4.12}$$

El primer paso del método consiste en factorizar A en L y U, que para un sistema de tercer orden quedan

$$
U = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix}
L = \begin{bmatrix} 1 & 0 & 0 \\ a_{21}/a_{11} & 1 & 0 \\ a_{31}/a_{11} & a_{32}/a_{22} & 1 \end{bmatrix}
\tag{4.13}
$$

donde U es el resultado de la eliminación Gaussiana y L almacena los factores de escala de cada paso de eliminación. La segunda fase es la sustitución hacia adelante, que da como resultado un nuevo RHS:

$$Ux = L^{-1}b = b' \tag{4.14}$$

El último paso es la sustitución hacia atrás, mediante la cual se calculan los elementos del vector desconocido, x:

$$x = U^{-1}b' = U^{-1}L^{-1}b \tag{4.15}$$

Nótese que invertir L y U es trivial ya que son triangulares. la ventaja de la factorización LU frente a la eliminación Gaussiana es que el circuito se puede resolver repetidamente para vectores de excitación diferentes, es decir, para diferentes RHS. Esta propiedad es muy útil en ciertos análisis de SPICE tales como análisis de sensividad, ruido, y distorsión.

Una observación importante sobre la matriz de conductancia G (ec. (4.4)) es que es dominantemente diagonal y muchos de los términos que están fuera de

la diagonal son cero. Una explicación simple de esto es que los términos que están fuera de la diagonal son generados por conductancias conectadas entre pares de nodos, y normalmente un nodo se conecta sólo a dos o tres nodos vecinos. En la matriz de conductancia de un circuito que tiene unas pocas decenas de nodos, sólo dos o tres de las decenas de términos que están fuera de la diagonal son distintos de cero; en otras palabras la matriz de conductancia es dispersa (*sparse*). La dispersión (*sparsity*) se mantiene en la submatriz de ecuaciones de corriente R (ec. (4.4)), donde muchos elementos de la diagonal son también cero. Este hecho se estudiará con más detalle en la siguiente sección que trata de la precisión. La matriz MNA para el circuito puente-T, ec. (4.4), tiene 20 elementos iguales a cero de un total de 36 elementos de la matriz. La dispersión de la matriz se puede definir como

$$dispersión(sparsity) = \frac{\text{nº de elementos iguales a cero}}{\text{nº tot al de elementos de la matriz}} \tag{4.16}$$

La dispersión en este ejemplo es del 55.6%. La dispersión es una característica muy útil, ya que gracias a ella se puede reducir la cantidad de datos a almacenar y a calcular. La siguiente sección explica la necesidad de una reordenación cuidadosa de las ecuaciones manteniendo al mismo tiempo la precisión y la dispersión.

4.1.2 Precisión

Los problemas de precisión en la resolución de un circuito lineal se pueden clasificar en topológicos y numéricos. Los elementos definidos en tensión, tales como las fuentes de tensión y las bobinas, generan elementos de la diagonal iguales a cero relacionados con la ecuación de corriente [HORUB75]. Este problema se puede solucionar en la fase de inicialización basándonos en una reordenación topológica, también conocida como *preordenación*. En SPICE la fila de la matriz MNA que corresponde a la ecuación de corriente de una fuente de tensión se intercambia con la ecuación de tensión de nodo que corresponde al terminal positivo de la misma fuente de tensión [COHEN81].

Un segundo problema, también de naturaleza topológica, es la puesta en corto de elementos definidos en tensión. Esto trae como consecuencia una cancelación del valor de un elemento de la diagonal durante el proceso de

factorización. Se ha propuesto un algoritmo de reordenación [HAYAT81] que encuentra una secuencia de ecuaciones libre de problemas topológicos. Este esquema de reordenación topológica adicional no es, sin embargo, necesario una vez que se utiliza la reordenación numérica conocida como *pivoting*.

Un circuito que no puede ser resuelto sólo mediante preordenación debido al problema anterior es el que se muestra en la figura 4.3. La matriz MNA de este circuito es

$$\begin{bmatrix} 1/5 & -1/5 & 0 & 0 & 0 & 0 \\ -1/5 & 1/5 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & -1/2 & -1 & 0 \\ 0 & 0 & -1/2 & 1/2 & 0 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ I_L \\ I_V \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 3 \end{bmatrix} \tag{4.17}$$

Después de la preordenación, la cual intercambia la fila I_L por el nodo 2 y la fila I_V por el nodo 4, y la reordenación para mantener la dispersión, descrita anteriormente, la matriz del circuito queda

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 & 0 & -1/2 \\ 0 & 0 & 1/5 & -1/5 & 0 & 0 \\ 0 & 0 & -1/5 & 1/5 & 1 & 0 \\ -1/2 & 0 & 0 & 0 & -1 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

Las ecuaciones matriciales 3 y 4 forman un bloque diagonal

$$\begin{bmatrix} G_1 & -G_1 \\ -G_1 & G_1 \end{bmatrix}$$

donde G_1=1/5; durante la factorización LU se crea un cero en la diagonal en la fila 4. Este ejemplo muestra la necesidad de un esquema de reordenación basado en las entradas de las matrices en cada paso de la descomposición LU.

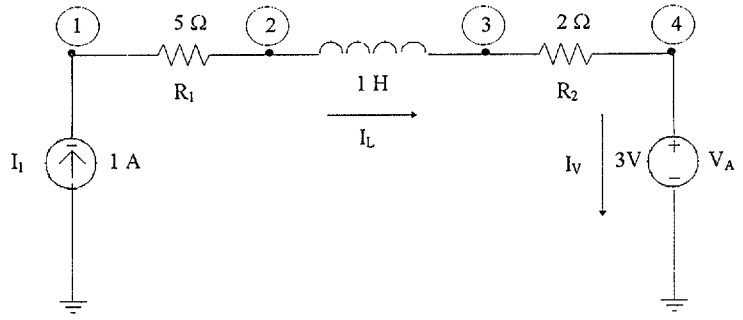


fig. 4.4 Circuito ejemplo de cancelación diagonal.

Sin embargo, no todos los problemas asociados con la solución de la matriz MNA son topológicos. Otro problema es que los ordenadores sólo tienen precisión finita. El límite del número de dígitos en la mantisa de un número en coma flotante, hasta 15 dígitos decimales para doble precisión, o 64 bits para números reales en el formato de coma flotante del IEEE, pueden producir la pérdida de significado de un término de la matriz relativo a otro durante la resolución de las ecuaciones lineales.

El sencillo circuito que se muestra en la figura 4.4 [FRERE76] demuestra este punto. Se asume que el ordenador puede representar sólo cuatro dígitos de precisión en coma flotante. Es fácil imaginar que habiendo un elemento de conmutación un circuito puede tener resistencias dentro de un rango desde 1 mΩ a 1 GΩ, es decir, valores que difieren 12 ordenes de magnitud. Las ecuaciones del circuito son

$$\begin{bmatrix} 1 & -1 \\ -1 & 1.0001 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.18}$$

Sin embargo, el ordenador hipotético de este ejemplo redondea el elemento a_{22} a

$$a_{22} = G_1 + G_2 = 1.000$$

debido a la limitación de los cuatro dígitos. Como en el ejemplo anterior, se crea un cero en la diagonal durante la eliminación Gaussiana, dando como resultado valores erróneos de infinito para V_1 y V_2. Este problema se debe a la insuficiente precisión del ordenador y no se puede solucionar mediante reordenación. La matriz de conductancia es singular.

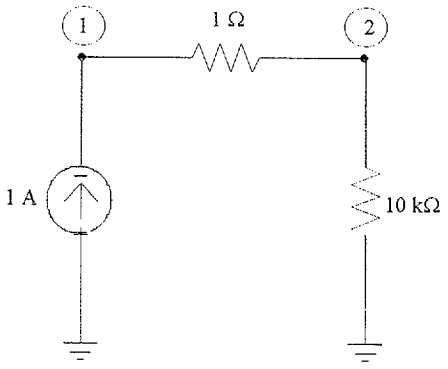


fig. 4.5 Circuito ejemplo de limitación del rango de coma flotante.

Otro problema de precisión, también debido al número limitado de dígitos, se puede deber al proceso de eliminación Gaussiana. Un ejemplo de este caso se muestra en la figura 4.5. Las ecuaciones nodales para este circuito son

$$\begin{aligned} G_1V_1 + g_2V_2 &= I \\ -g_1V_1 + G_2V_2 &= 0 \end{aligned} \tag{4.19}$$

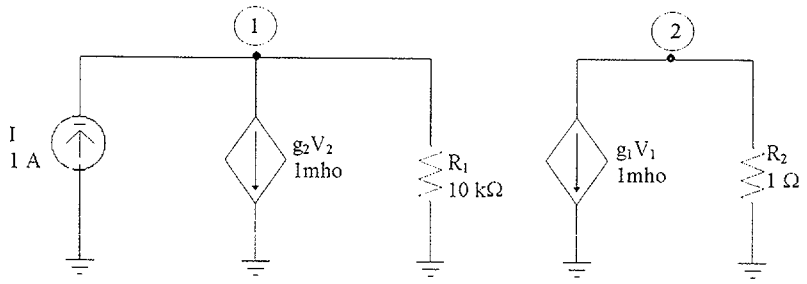


fig. 4.6 Circuito ejemplo de error de redondeo.

La sustitución de los valores de las conductancias, G_1 y G_2, y las transconductancias, g_1 y g_2, producen el siguiente sistema de ecuaciones:

$$\begin{bmatrix} 0.0001 & 1 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Después de un paso de eliminación el sistema se convierte en

$$\begin{bmatrix} 0.0001 & 1 \\ 0 & 10{,}000 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 10{,}000 \end{bmatrix}$$

con la siguiente solución en un ordenador con cuatro dígitos de precisión

$$V_2 = 1$$
$$V_1 = 0$$

que es obviamente incorrecta. Si las filas de la ecuación se intercambian entre si antes de la factorización para poner el elemento mayor en la diagonal, el sistema de ecuaciones es:

$$\begin{bmatrix} -1 & 1 \\ 0.0001 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

con la solución que se ajusta más a la realidad

$$V_2 = 1$$
$$V_1 = 1$$

La precisión de esta solución se ve afectada todavía por el limitado número de dígitos; sin embargo, la solución es correcta para el número de representación disponible. La solución exacta es

$$V_2 = 0.9999$$
$$V_1 = 0.9999$$

Este tipo de pérdida de precisión se puede observar también en el caso de una serie de etapas de ganancia elevada conectadas en un lazo cerrado con retroalimentación, tal como un oscilador en anillo; los elementos fuera de la diagonal son transconductancias, las cuales crecen según una función potencial de las ganancias durante la factorización y pueden hacer diverger eventualmente los valores de los términos de la diagonal. En la figura 4.7 se muestra un oscilador en anillo implementado con transistores bipolares. El reemplazo de los transistores con un modelo linealizado durante cada iteración, como se describe en la sección 4.2, produce el siguiente sistema de ecuaciones:

$$\begin{bmatrix} G+g_{\pi 2} & 0 & g_{m1} \\ g_{m2} & G+g_{\pi 3} & 0 \\ 0 & g_{m3} & G+g_{\pi 1} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} I_{eq1} \\ I_{eq2} \\ I_{eq3} \end{bmatrix} \tag{4.20}$$

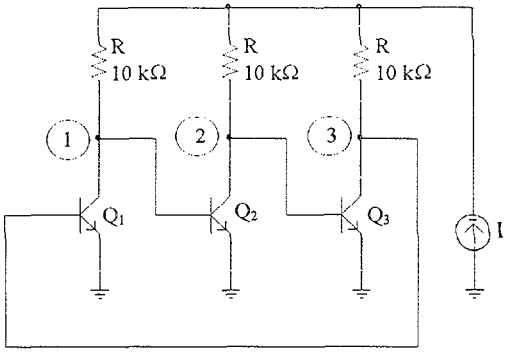


fig. 4.7 Oscilador en anillo con BJT. Ejemplo de error de redondeo.

Las soluciones de las tensiones en los nodos, V_1, V_2 y V_3 deberían ser idénticas. Sin embargo difieren si el sistema anterior se resuelve como está, debido a que la autoconductancia de cada nodo, o término diagonal, pierde su contribución durante el proceso de eliminación.

Otro problema numérico ocurre cuando la matriz entra en un cierto paso del proceso de eliminación siendo muy pequeño, de forma que la relación entre el valor mayor en la matriz que nos queda y el valor mayor en la matriz inicial es de un orden de magnitud mayor del que puede representar el ordenador.

Los ejemplos anteriores demuestran que no son suficientes los reordenamientos topológicos y que la ordenación de la matriz MNA debe basarse en los valores reales generados por el circuito tal cual. A veces también es necesaria la reordenación durante el proceso de iteración. La reordenación basada en la selección del elemento mayor de la diagonal, o *pivoting*, es esencial para una resolución precisa de un conjunto de ecuaciones si los valores originales pueden verse alterados de forma significativa por el proceso de factorización. El *pivoting* consiste en tomar un valor llamado *pivot* para comparar con los valores de la diagonal de la matriz resultante de forma que si no se encuentra ningún valor que sea mayor que ese *pivot* la matriz queda declarada como singular y SPICE aborta el análisis. Cuanto mayor sea el circuito más importante es el *pivoting*, debido a la acumulación de los errores de redondeo en el proceso de resolución. En la tesis de [COHEN81] se puede ver una presentación detallada de los elementos relacionados con la precisión numérica.

Como una parte de la reordenación numérica, se puede elegir una estrategia de *pivoting parcial*, la cual toma el elemento mayor en la columna o fila, o una estrategia de *pivoting total*, la cual selecciona el elemento mayor de la

matriz que va quedando. La segunda restricción que se menciona en la sección anterior es la preservación de la dispersión de la matriz. En SPICE se utiliza el algoritmo de Markowitz para seleccionar, entre un número aceptable de valores de *pivots* el que introduzca el menor número de términos de relleno. Los términos de relleno son aquellos que son cero al comienzo del proceso de factorización y se hacen distintos de cero durante la descomposición LU. En [VLADI94] se puede ver como opera este algoritmo.

4.2 Solución en dc de Circuitos No Lineales

En el capítulo 3 se mostró que los dispositivos semiconductores se describen mediante características I-V no lineales tales como funciones exponenciales o cuadráticas e incluso hiperbólicas. Las fuentes controladas también se pueden describir mediante ecuaciones constitutivas de ramas (BCEs) las cuales se limitan a polinomiales en SPICE2 pero pueden ser cualquier función arbitraria en SPICE3 y en otras versiones comerciales. Las ecuaciones nodales modificadas para un circuito con transistores son un conjunto de ecuaciones simultáneas no lineales. El bucle iterativo que se muestra en la figura 4.1 para la solución en dc de ecuaciones no lineales se implementa en SPICE utilizando el algoritmo de Newton-Raphson. Este algoritmo se describe en la sección 4.2.1.

El proceso de resolución iterativo continua hasta que convergen los valores de las tensiones y corrientes desconocidas, es decir, hasta que las soluciones de dos iteraciones consecutivas son la misma. El criterio para la convergencia se presenta en la sección 4.2.2.

4.2.1 Método de NEWTON-RAPHSON

El algoritmo de Newton-Raphson para la resolución de las ecuaciones de circuitos no lineales lo explicaremos utilizando un circuito simple consistente en un diodo, una conductancia, G, y una fuente de corriente, I_A, tal como se muestra en la figura 4.8.a. La representación gráfica de las ecuaciones constitutivas de ramas (BCEs) para los dos elementos,

$$I_D = I_S(e^{V/V_{th}} - 1) \qquad (4.21.a)$$

$$I_G = GV \qquad (4.21.b)$$

se muestra en la figura 4.4.b; la solución V_D se localiza en la intersección de las dos funciones I_D e I_G, debiéndose satisfacer la ley de corrientes de Kirchhoff.

$$I_A = I_D + I_G \tag{4.22}$$

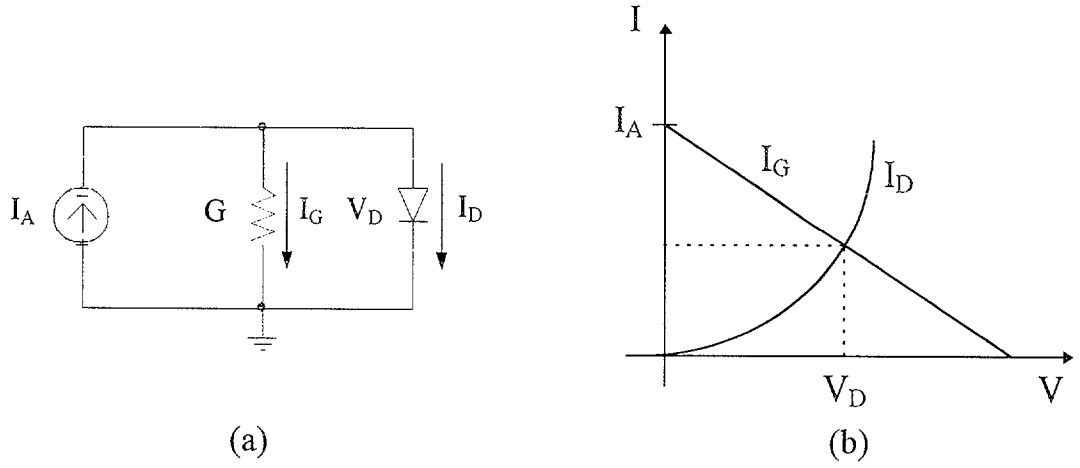


fig. 4.8 Circuito con diodo: (a) circuito, (b) solución gráfica.

Las ecuaciones no lineales de este tipo se resuelven generalmente mediante una aproximación iterativa, tal como el método de Newton, el cual se conoce también como el método de la tangente [ORTRE70]. La corriente I_D en la ecuación (4.21.a) se puede aproximar mediante una serie de Taylor alrededor de la solución de prueba, o punto de operación, V_{D0}:

$$I_D = I_{D0} + \left.\frac{dI_D}{dV_D}\right|_{V_{D0}} (V_D - V_{D0}) = I_{D0} + G_{D0}(V_D - V_{D0}) = I_{DN0} + G_{D0}V_D \tag{4.23}$$

donde sólo se consideran los términos de primer orden. La ecuación (4.23) representa la ecuación constitutiva de rama linealizada de el diodo al rededor del punto de operación de prueba, V_{D0}. Basándonos en esta ecuación, el diodo se puede representar mediante un modelo equivalente Norton con corriente I_{DN0} y conductancia G_{D0}. En la figura 4.9 se muestra el nuevo circuito y la representación gráfica de la ecuación constitutiva de rama linealizada del diodo. Una vez se ha introducido la ecuación (4.23) en la ecuación (4.22), la ecuación nodal para este circuito queda

$$(G + G_{D0})V = I_A - I_{DN0} \tag{4.24}$$

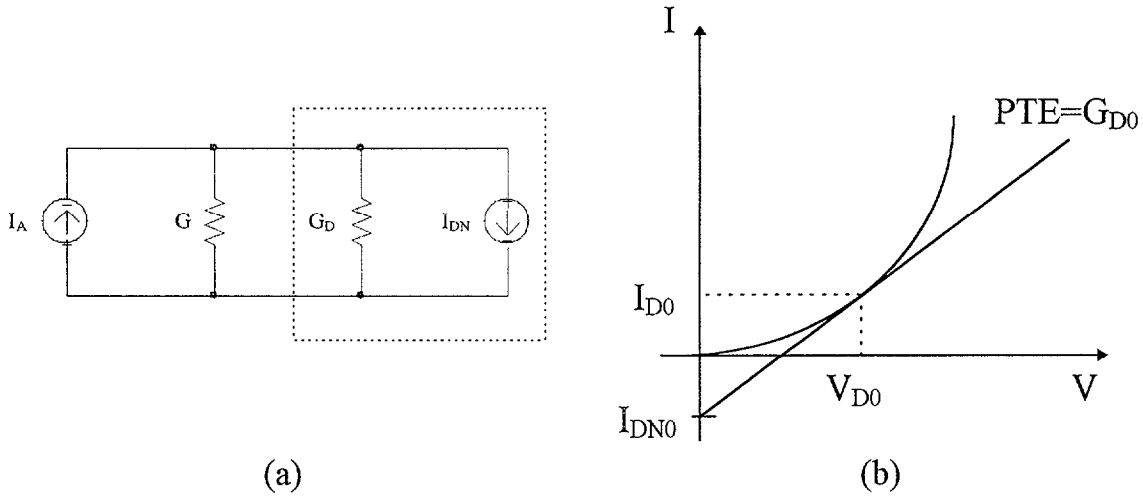


fig. 4.9 Circuito (a) y representación gráfica (b) de la ecuación constitutiva de rama linealizada del diodo.

La solución de esta ecuación es V_{D1}, la cual se convierte en el nuevo punto de operación de prueba para el diodo. La ecuación (4.24) se resuelve de forma iterativa con los nuevos valores de G_D e I_{DN} en cada iteración *(i+1)*, la cual se basa en la tensión en la iteración anterior *(i)*. El proceso iterativo se describe mediante la ecuación nodal

$$(G + G_D^{(i)})V^{(i+1)} = I_A - I_{DN}^{(i)} \tag{4.25}$$

la cual representa para este caso simple el método iterativo de Newton. En la figura 4.10 se muestra la representación gráfica del proceso iterativo para este circuito. Las intersecciones de la conductancia linealizada del diodo con la línea de carga G definen los valores de V_{Di} de las sucesivas iteraciones. Este proceso converge a la solución, $\hat{V}$.

El método de Newton general aplicado a una función no lineal $g(x)$ de una única variable es

$$x^{(i+1)} = x^{(i)} - g^{(i)} / g'^{(i)} \tag{4.26}$$

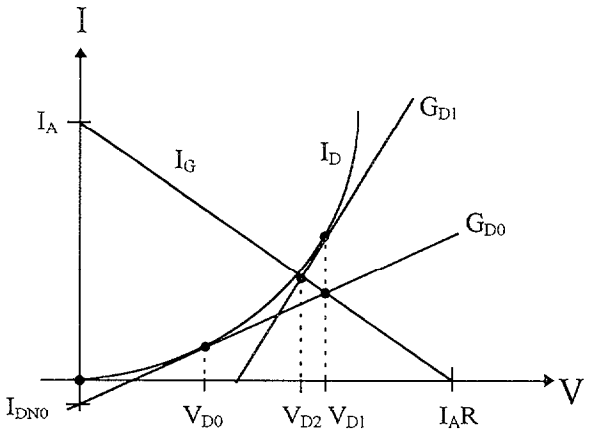


fig. 4.10 Método iterativo de Newton para el circuito del diodo.

donde para la ecuación del diodo

$$g(x) = I_D(V) = I_S(e^{V/V_{th}} - 1)$$

y en la solución $\hat{x}$, $I_D^{(i+1)} = I_D^{(i)}$.

De acuerdo con la ecuación (4.23), la aproximación de la linealización generalizada para un conjunto de ecuaciones no lineales $g(x) = 0$ consiste en calcular las derivadas parciales respecto a las tensiones de control. Esos valores forman la contribución de la conductancia de las ecuaciones constitutivas de ramas al análisis nodal modificado (MNA) del circuito, llamado Jacobiano, J. Para un circuito arbitrario, el conjunto de todas las ecuaciones no lineales se reemplaza en cada iteración por el siguiente conjunto de ecuaciones lineales:

$$J(x^{(i)}) \cdot x^{(i+1)} = J(x^{(i)}) \cdot x^{(i)} - g(x^{(i)}) \tag{4.27}$$

donde $J(x^{(i)})$ es el Jacobiano calculado con $x^{(i)}$, es decir, la solución a la iteración anterior. La matriz de conductancia A y el RHS b para un circuito no lineal tienen la siguiente representación matricial:

$$A = J(x^{(i)}) + G \tag{4.28}$$

$$b = J(x^{(i)}) \cdot x^{(i)} - g(x^{(i)}) + C \tag{4.29}$$

Las ecuaciones (4.28) y (4.29) incluyen la contribución de los elementos lineales y las fuentes de corriente independientes del circuito, G y C, respectivamente. En cada iteración el circuito se describe como un sistema lineal, ecs. (4.6), (4.28), y

(4.29), el cual se resuelve utilizando la factorización LU descrita en la sección 4.1.1.

Una cuestión importante es si las ecuaciones anteriores convergen hacia una solución y cuantas iteraciones son necesarias. Las características no lineales I-V de un dispositivo semiconductor son funciones exponenciales o cuadráticas. Tomando como ejemplo la característica del diodo de la figura 4.5, la conductancia equivalente puede variar desde 0 en la región inversa a infinito en la región directa. Debido al limitado rango de números en punto flotante de que dispone un ordenador y a la falta de límites que proporciona el algoritmo de Newton según la ecuación (4.26), el esquema iterativo puede no converger.

Se hace necesario un algoritmo que controle los cambios en las variables de estado de los elementos no lineales de iteración a iteración si se quiere que el circuito simulado converja a la solución correcta. Este tipo de algoritmos se conocen como *algoritmos de limitación* y son los que determinan las características de convergencia del simulador. Un algoritmo de limitación acepta las soluciones que no cambian y, cuando se dan cambios grandes en los valores de las funciones no lineales, los limita corrigiendo los nuevos valores de la iteración Newton [CALAH72] de la ecuación (4.26) a la siguiente:

$$x^{(i+1)} = x^{(i)} - \alpha g^{(i)} / g'^{(i)} \tag{4.30}$$

El parámetro α ($0 < \alpha \leq 1$) indica que sólo se acepta una fracción de la carga en cada iteración. La ecuación (4.30) define el algoritmo iterativo de Newton-Raphson. La elección de α se implementa mediante el algoritmo de limitación , el cual se ajusta a los diferentes valores de las características no lineales de cada iteración para cada dispositivo.

En la figura 4.11 se muestra una implementación práctica del método de Newton-Raphson modificado. El esquema propuesto por Colon se utiliza con éxito en SPICE para limitar las nuevas tensiones de unión de los diodos y BJTs. En cada iteración, la nueva solución de la tensión de unión, V_1, si es mayor que el valor anterior, V_0, se utiliza para derivar una nueva corriente, I'_1, la cual se basa en la última linealización de la característica del diodo. La tensión que corresponde a I'_1 en la característica no lineal del diodo es V'_1, siendo esta menor que la solución original, V_1; V'_1 se convierte en el nuevo punto de operación de

prueba para el diodo en la iteración actual. Si V_1 es menor que V_0, se acepta directamente como el nuevo punto de operación de prueba debido a que no hay peligro de que se produzca una solución que se salga de los límites. El peligro de la ausencia de ningún límite es que la solución de prueba puede generar valores de la función exponencial muy grandes, los cuales no se pueden corregir en las siguientes iteraciones, y el proceso no convergería.

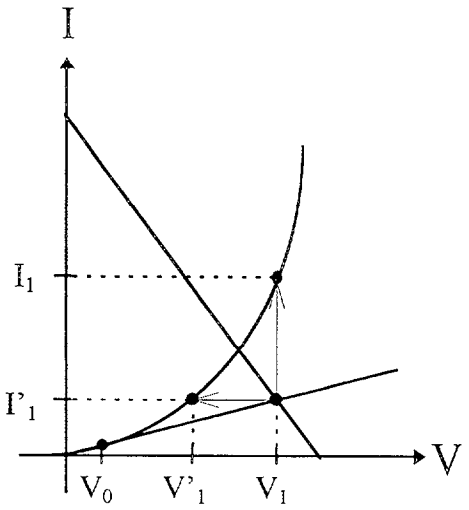


fig. 4.11 Método iterativo de Newton-Raphson con limitación de corriente.

El algoritmo de Newton-Raphson tiene propiedades de convergencia cuadrática si el valor supuesto inicialmente, V_0 en la figura 4.11, es cercano a la solución. Por lo tanto, en un simulador circuital es importante el proporcionar un valor inicial como un conjunto de tensiones de nodo o tensiones de terminal para los dispositivos semiconductores no lineales. Los simuladores circuitales suelen comenzar con todos los valores desconocidos puestos a cero. SPICE2 inicializa los dispositivos semiconductores en un punto de operación de tal forma que en la primera iteración puede haber conductancias no nulas en la matriz circuital MNA. Una protección adicional que se utiliza en SPICE es el uso de una conductancia paralela mínima, *GMIN*, que por defecto vale 10^{-12} mho, la cual se conecta en paralelo a todas las uniones pn tales como las de BE y BC en los BJTs y las de BD y BS en los MOSFETs, para prevenir que cuando se carguen las conductancias en la matriz circuital de admitancias éstas tengan valor cero cuando dichas uniones están polarizadas en inversa.

4.2.2 Convergencia

Esta sección trata de los elementos que definen cómo y cuando se consigue la convergencia. La ecuación (4.30) define el método de iteración de Newton-Raphson; el proceso iterativo se acaba cuando se alcanzan las dos condiciones siguientes:

1. Todas las tensiones y corrientes del vector desconocido están dentro de una tolerancia prescrita para dos iteraciones consecutivas.
2. Los valores de las funciones no lineales y los de las aproximaciones lineales están dentro de la tolerancia prescrita.

Pasamos a introducir las tolerancias utilizadas en SPICE para establecer la convergencia. Cada tolerancia, para las variables de corriente o tensión, está formada por un término relativo y otro absoluto. La tolerancia en tensión para un nodo n, ε_{Vn}, se define como

$$\varepsilon_{Vn} = RELTOL \cdot max(V_n^{(i+1)}, V_n^{(i)}) + VNTOL \tag{4.31}$$

Los valores de las tensiones nodales de dos iteraciones consecutivas deben satisfacer la siguiente desigualdad para que se cumpla la condición de convergencia:

$$\left| V_n^{(i+1)} - V_n^{(i)} \right| \leq \varepsilon_{Vn} \tag{4.32}$$

RELTOL y *VNTOL* son parámetros SPICE que representan respectivamente la tolerancia en tensión relativa y absoluta. Sus valores por defecto son:

$$RELTOL = 10^{-3}$$
$$VNTOL = 1\mu V$$

La tolerancia absoluta define el valor mínimo para el cual una variable dada es todavía precisa; con los valores por defecto de SPICE, las tensiones son precisas en 1 parte por 1000 bajo 1μV de resolución. Esto significa que un nodo con 100 V es preciso en 100 mV y una tensión de 10 μV es preciso en sólo 1 μV.

La convergencia no sólo se basa en las variables circuitales sino también en los valores de las funciones no lineales que definen las ecuaciones constitutivas de ramas (BCEs) de los elementos no lineales. En el caso de los dispositivos semiconductores, las funciones no lineales son corrientes, por ejemplo, la I_D de los diodos, la I_C y la I_B de los BJTs, y la I_{DS} de los FETs. En la discusión que sigue la función no lineal se refiere por simplicidad a una corriente.

SPICE define como *test* para la convergencia la diferencia entre la expresión no lineal, I_D, evaluada con la última solución de la tensión de unión $V_J^{(i)}$ y la aproximación lineal $\hat{I}_D$, evaluada utilizando la tensión resultante actual, $V_J^{(i+1)}$:

$$\begin{aligned} \hat{I}_D &= G_D V_J^{(i+1)} - I_{DN} \\ I_D &= I_S(e^{V_J^{(i)}/V_{th}} - 1) \end{aligned} \tag{4.33}$$

La tolerancia se define como

$$\varepsilon_I = RELTOL \cdot max(\hat{I}_D, I_D) + ABSTOL \tag{4.34}$$

y el proceso de iteración converge cuando

$$\left|\hat{I}_D - I_D\right| \le \varepsilon_I \tag{4.35}$$

$$\left|V_J^{(i+1)} - V_J^{(i)}\right| \le \varepsilon_{VJ} \tag{4.36}$$

ABSTOL es la tolerancia absoluta en corriente y su valor por defecto en SPICE2 y SPICE3 es 10^{-12} A. Este valor por defecto es bastante preciso para la corriente de base de los BJTs pero puede ser demasiado restrictivo para otros dispositivos, como los FETs, cuyo funcionamiento es controlado mediante la tensión de puerta por lo que I_{DS} es mayor que 1μA para la mayoría de las aplicaciones.

Durante el proceso de resolución se deben evaluar el equivalente lineal de cada elemento no lineal. Este es un proceso que consume mucho tiempo y no siempre es necesario. En los análisis en el dominio del tiempo, durante las iteraciones que se ejecutan en cada instante de tiempo, se comprueba cada elemento no lineal para ver si ha habido algún cambio en las tensiones de terminal o en la corriente de salida respecto al último instante de tiempo en el que se hayan calculado. Si las variables de control y las funciones resultantes del

dispositivo no han cambiado, no se calcula un nuevo modelo linealizado para dicho dispositivo; es decir se hace un *bypass*. En otras palabras, las conductancias linealizadas obtenidas en el instante de tiempo anterior se utilizan de nuevo en la matriz circuital.

La operación de *bypass* trae como resultado un ahorro de tiempo sin que esto afecte al resultado para la mayoría de los circuitos. Sin embargo, hay casos en donde un *bypass* puede causar una no convergencia en un instante temporal posterior. La verificación de si ha de hacerse o no un *bypass* a un dispositivo se basa en las ecuaciones (4.35) y (4.36). Esta verificación puede limitar los dispositivos a los que se les puede hacer un *bypass* reduciendo las tolerancias; sin embargo, esta reducción puede afectar de forma negativa al *test* de convergencia del circuito global.

4.3 Solución en el Dominio del Tiempo

La solución en el dominio del tiempo es el análisis más complejo que se lleva a cabo en un simulador circuital, ya que éste envuelve todos los algoritmos presentados a lo largo de este capítulo, tal y como se muestra de forma gráfica en el diagrama de flujo de la figura 4.1. Como se describió en la introducción de este capítulo, el análisis en régimen transitorio se divide en una secuencia de soluciones cuasiestáticas.

En esta sección se pretende esbozar las técnicas numéricas utilizadas en SPICE para transformar un conjunto de ecuaciones diferenciales, tales como las que representan las ecuaciones constitutivas de ramas de capacitores e inductores, en un conjunto de ecuaciones algebraicas. Esta no pretende ser una presentación detallada que puede encontrarse en [CHULI75] y [NAGEL75].

4.3.1 Integración Numérica

Los diferentes algoritmos de integración numérica y sus propiedades se introducen mejor con un ejemplo.

Considérese el circuito RC serie que se muestra en la figura 4.12, al cual en el instante $t = 0$ se le aplica a la entrada un escalón de tensión de magnitud V_i.

El problema consiste en encontrar las tensiones $v_R(t)$ y $v_C(t)$ que caen en la resistencia y en el condensador respectivamente a lo largo del tiempo.

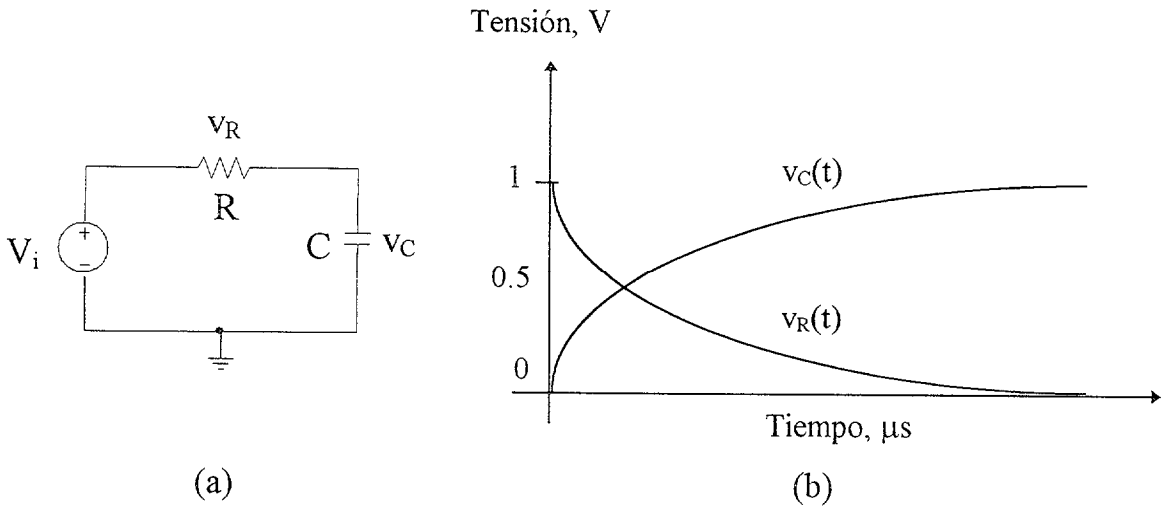


fig. 4.12 Circuito RC: (a) circuito, (b) solución de $v_C(t)$ y $v_R(t)$.

De las ecuaciones constitutivas de ramas de los resistores y capacitores se obtiene la siguiente ecuación:

$$v_R(t) = R \cdot i_R(t) = RC\frac{dv_C}{dt} \tag{4.37}$$

Por otro lado, aplicando las leyes de Kirchhoff obtenemos:

$$v_C(t) = V_i - v_R(t) \tag{4.38}$$

De las dos ecuaciones anteriores obtenemos las siguientes ecuaciones diferenciales para $v_R(t)$ y $v_C(t)$:

$$\begin{aligned} \dot{v}_R(t) &= -\frac{1}{RC}v_R = -\frac{1}{\tau}v_R = \lambda v_R = f(v_R) \\ \dot{v}_C(t) &= -\frac{1}{RC}(V_i - v_C) = \frac{1}{\tau}(V_i - v_C) = f(v_C) \end{aligned} \tag{4.39}$$

Las soluciones para $v_R(t)$ y $v_C(t)$ son

$$\begin{aligned} v_R(t) &= V_i e^{-t/\tau} \\ v_C(t) &= V_i(1 - e^{-t/\tau}) \end{aligned} \tag{4.40}$$

y se presentan en la figura 4.12.b.

La resolución SPICE de la ecuación anterior en el intervalo de tiempo predeterminado por el usuario, se realiza para un número discreto de puntos en el tiempo, en donde las ecuaciones diferenciales se reemplazan por ecuaciones algebraicas. Por simplicidad, sea x la solución y x_n los valores en los instantes t_n:

$$x_n = x(t_n)$$

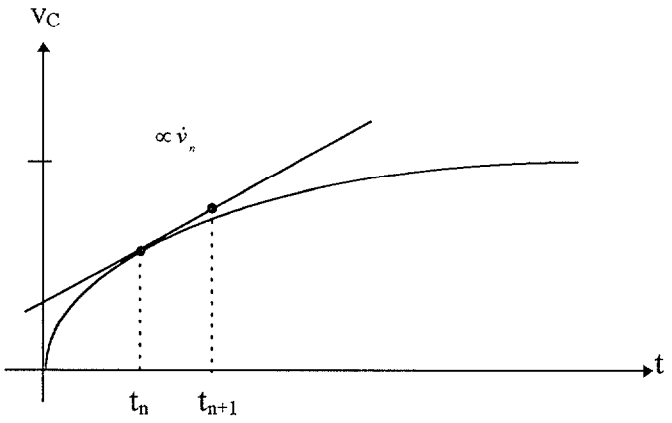
La solución en t_{n+1}, x_{n+1}, se puede expresar mediante una serie de Taylor alrededor de x_n:

$$x_{n+1} = x_n + h\dot{x}_n \tag{4.41}$$

donde h es el incremento del tiempo el cual se supone igual para todos los puntos en el tiempo. Esta ecuación es idéntica a la aproximación de las diferencias finitas de la derivada de x y representa la formula de integración hacia adelante de Euler (FE, *forward Euler*). La sustitución de las ecuaciones (4.39) en la (4.41) produce la siguiente solución recursiva para v_C en t_{n+1}:

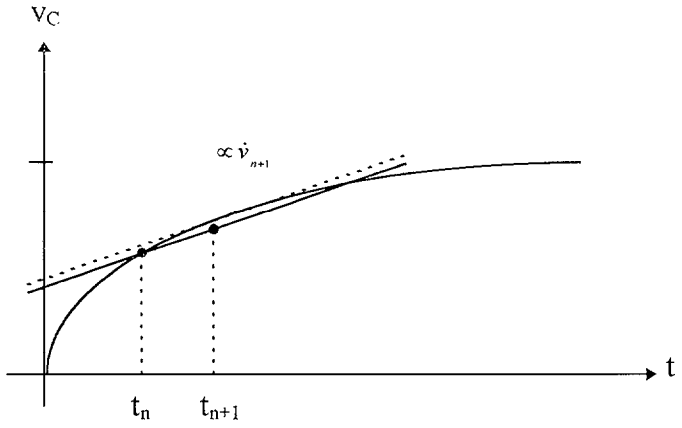
$$v_C(t_{n+1}) = v_C(t_n) + h\dot{v}_C(t_n) = v_C(t_n) + hf(v_{Cn}) \tag{4.42}$$

Esta ecuación se representa de forma gráfica en la figura 4.13.a. Se puede observar un error algo considerable para $v_C(t_{n+1})$ calculado con la ecuación (4.42).



(a)

fig. 4.13 Solución de $v_C(t)$ entre t_n y t_{n+1}: (a) solución FE.



(b)

fig. 4.14 Solución de $v_C(t)$ entre t_n y t_{n+1}: (b) solución BE.

Si en la ecuación (4.41) x_{n+1} se expresa en términos de la derivada en $t_{n+1}, \dot{x}_{n+1}$, se obtiene una solución diferente para v_C

$$x_{n+1} = x_n + h\dot{x}_{n+1} \tag{4.43}$$

Esta representa la formula de integración hacia atrás de Euler (BE, *back Euler*). Debido a que la ecuación (4.43) debe ser resuelta simultáneamente para x así como para su derivada, esta formula se conoce como un método implícito, mientras que la FE es un método explícito. La interpretación gráfica de esta solución se muestra en la figura 4.13.b. Se puede observar que v_{n+1} tal y como viene dado por la formula BE es menos sensible al tamaño del intervalo de tiempo h que si fuese dado por la formula FE.

Una medida importante de la precisión de un método de integración numérico es el error de truncación local (*LTE, local truncation error*), el cual se evalúa en cada punto en el tiempo. Para los métodos FE y BE, el *LTE* se puede aproximar por el primer término de la serie de Taylor:

$$\begin{aligned} x_{n+1} &= x_n + h\dot{x}_n + \frac{h^2}{2}\ddot{x}_n \\ LTE &= \left|\frac{h^2}{2}\ddot{x}_n\right| \end{aligned} \tag{4.44}$$

Los algoritmos para el control automático del intervalo de tiempo h tales como el que usa SPICE se basan en comprobar si el *LTE* de cada ecuación constitutiva de rama está dentro de los límites prescritos.

Los dos métodos introducidos hasta ahora se conocen como métodos de primer orden debido a que se desprecian los términos de ordenes superiores en las series. Basándonos en la definición anterior del *LTE*, puede ocurrir que utilizando términos de ordenes superiores de las series en la resolución de x_{n+1} obtengamos *LTE* menores. La integración trapezoidal es un método de segundo orden. Este se basa en la observación de que se puede obtener una solución más precisa $v_C(t_{n+1})$ si en las ecuaciones (4.41) y (4.43) se usan las pendientes en t_n y t_{n+1} para comparar con una o la otra:

$$x_{n+1} = x_n + \frac{1}{2}h(\dot{x}_n + \dot{x}_{n+1}) \tag{4.45}$$

La mayor precisión de este método es obvia si observamos la solución gráfica de $v_C(t_{n+1})$ que se muestra en la figura 4.14.

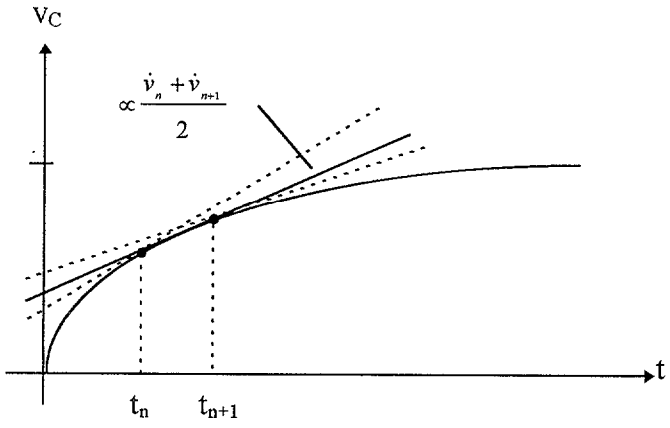


fig. 4.15 Solución trapezoidal de $v_C(t)$ entre t_n y t_{n+1}.

El *LTE* de la formula de la integración trapezoidal se puede conseguir sustituyendo $\dot{x}_{n+1}$ en la ecuación (4.45) por una serie de Taylor:

$$\dot{x}_{n+1} = \dot{x}_n + h\ddot{x}_n + \frac{h^2}{2}\dddot{x}_n \tag{4.46}$$

y luego restando la solución trapezoidal, ecuación (4.45), a la solución exacta dada por los tres primeros términos de la serie de Taylor:

$$x_{n+1} = x_n + h\dot{x}_n + \frac{h^2}{2}\ddot{x}_n + \frac{h^3}{6}\dddot{x}_n \tag{4.47}$$

El *LTE* resultante para el método de integración trapezoidal para x_{n+1} es

$$LTE = \left| \frac{h^3}{12}\dddot{x}_n \right| \tag{4.48}$$

El *LTE* para $\dot{x}_{n+1}$ se obtiene sustituyendo x_{n+1} en la ecuación (4.45) por la ecuación (4.47) y luego restando la ecuación resultante en $\dot{x}_{n+1}$ a la ecuación (4.46):

$$LTE = \left| \frac{h^2}{6}\dddot{x}_n \right| \tag{4.49}$$

Una propiedad importante de un método de integración es su *estabilidad* o característica de convergencia. Mientras que el *LTE* es una medida local de la precisión en cada instante de tiempo, la estabilidad es una medida global de como la solución calculada por un método dado se aproxima a la solución exacta a medida que el tiempo tiende a infinito. La estabilidad es también una función del circuito específico. Podemos realizar un análisis cuantitativo de la estabilidad de los métodos de integración explicados hasta ahora para el circuito RC de la figura 4.12. Podemos comparar la solución exacta para $v_R(t)$, ecuaciones (4.40), con las soluciones FE, BE, y TR calculadas después de n intervalos de tiempo:

$$(FE) \quad V_i(1 - h/\tau)^n \tag{4.50}$$

$$(BE) \quad \frac{V_i}{(1 + h/\tau)^n} \tag{4.51}$$

$$(TR) \quad V_i \frac{(1 - h/2\tau)^n}{(1 + h/2\tau)^n} \tag{4.52}$$

Se puede observar que la solución FE puede producir soluciones erróneas si el intervalo de tiempo h es mayor que 2τ. Por contra, la solución BE decrece hacia cero a medida que aumenta el intervalo de tiempo tal como lo hace la solución exacta $v_R(t)$ en las ecuaciones (4.40). El método TR ofrece un resultado interesante convergiendo hacia cero pero haciéndolo de forma oscilatoria si $h >$

2τ. Este comportamiento del método TR se puede observar en SPICE especialmente cuando la solución viene a través de discontinuidades.

Los circuitos electrónicos poseen constantes de tiempo que pueden diferir en varios ordenes de magnitud; las ecuaciones que representan a estos circuitos constituyen *sistemas "torpes" (stiff systems)*. Los métodos de integración utilizados para resolver tales sistemas deben ser *torpemente estables (stiffly stable)*; en otras palabras, estos métodos deben proporcionar la solución correcta sin la restricción de que el intervalo de tiempo sea menor que la constante de tiempo más pequeña del circuito. Los métodos implícitos introducidos hasta ahora, BE y TR, son torpemente estables, pero no lo son los métodos explícitos, tales como el método FE. Los métodos de integración TR y BE son los utilizados por defecto en la mayoría de versiones de SPICE.

Se han desarrollado otras fórmulas de integración las cuales caen en la categoría general de métodos de integración polinomiales y se definen por

$$x_{n+1} = \sum_{i=0}^{n} a_i x_{n-i} + \sum_{i=-1}^{n} b_i \dot{x}_{n-i} \tag{4.53}$$

Si b_{-1} es cero, el método es explícito, y si b_{-1} es distinto de cero, el método es implícito. El algoritmo es de multipaso (*multistep*) si $i > 1$, es decir, si se necesita más de un instante de tiempo del pasado para calcular x_{n+1}. Las fórmulas de integración de Gear [GEAR67] de orden 2 a 6 han demostrado tener buenas propiedades de estabilidad. Estas fórmulas de integración se pueden ver en [VLADI94]. Las fórmulas de integración de Gear de orden 2 a 6 están implementas en SPICE2, SPICE3 y en la mayoría de las versiones comerciales de SPICE como alternativas al método por defecto TR. PSPICE usa sólo los algoritmos de Gear.

La respuesta en el dominio del tiempo de un circuito puede diferir dependiendo del método de iteración utilizado. Aunque en la gran mayoría de los casos tanto el método de Gear como el TR proporcionan la misma solución, los dos tienen diferentes características. Cuando el intervalo de tiempo es mayor que un cierto límite, el método TR converge a una solución de forma oscilatoria (ec. (4.52)). La formula de Gear de orden 2 tiene un comportamiento contrario proporcionando una respuesta amortiguada. Esta diferencia entre los dos métodos

se puede observar en la figura 4.16 que corresponde a la respuesta temporal del circuito LC de la figura 4.15 utilizando ambos métodos.

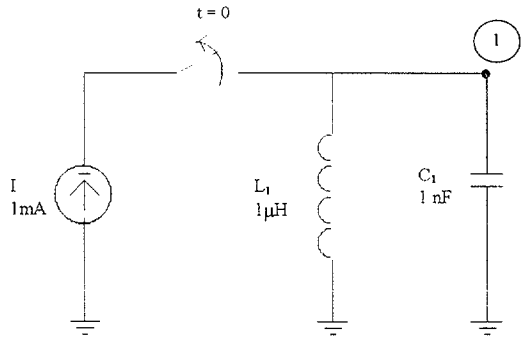


fig. 4.16 Circuito LC.

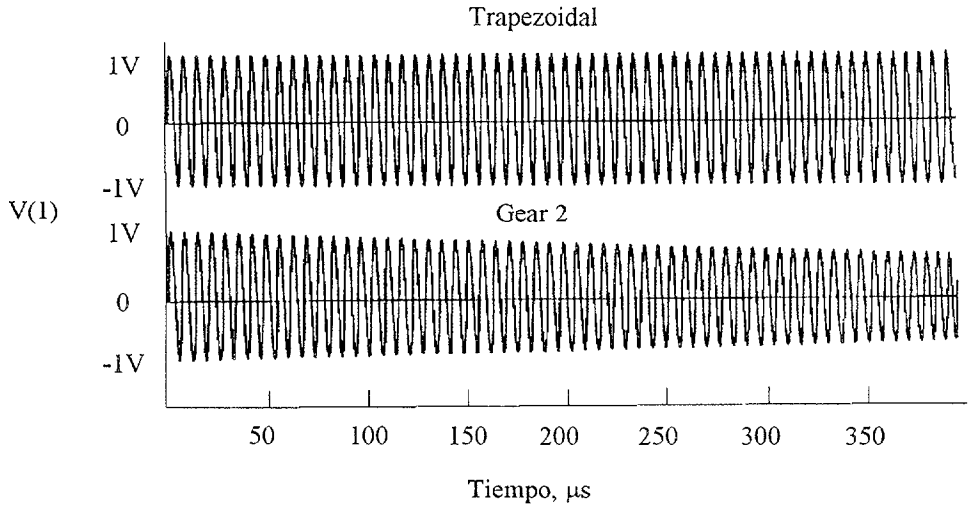


fig. 4.17 Respuesta del circuito LC con los algoritmos Trapezoidal y de Gear 2.

La estabilidad de los métodos de integración se presenta con más detalle en los trabajos de [MCCAL88] y de [NAGEL75]. Asimismo, la forma en que SPICE implementa los métodos de integración descritos puede encontrarse en [VLADI94]. Del análisis presentado en esta sección se extraen varias conclusiones; en primer lugar, el *LTE* de un método numérico disminuye para los métodos de ordenes mayores; por contra, la estabilidad se deteriora a medida que aumenta el orden del método. En segundo lugar, las ecuaciones de los circuitos electrónicos se deben resolver mediante métodos de integración torpemente estables en los que el intervalo de tiempo se determina por el *LTE* y no por restricciones de estabilidad.

Capítulo 5

5. Estructura de SPICE3

Un problema común cuando se intenta entender y modificar un programa como SPICE es su gran tamaño y la complejidad de la organización del programa. SPICE3 fue diseñado utilizando una filosofía *toolbox*. Cada paquete de rutinas es relativamente independiente de cualquier otro, de forma que permite a aquellos que realizan el mantenimiento y desarrollo del programa el elegir los algoritmos que mejor se ajusten a la tarea de un amplio rango de opciones disponibles. Para el uso general, se puede ensamblar una versión que posea todas las características del programa. Sin embargo en aquellos sistemas en los que el espacio es reducido, se puede ensamblar una versión más reducida simplemente eliminando las rutinas que no se van a utilizar (incluyendo paquetes enteros de análisis y dispositivos).

La estructura global de SPICE3 se muestra en la figura 5.1. En esta figura se pueden observar dos elementos importantes: el núcleo de SPICE3 y el núcleo de NUTMEG. El núcleo de SPICE3 que es el que realiza las simulaciones y se comunica con el resto del programa a través de una estructura de datos de entrada y otra de salida. NUTMEG es un post-procesador para SPICE. Su función es la de tomar el fichero de salida de SPICE y dibujar los resultados en un terminal gráfico o en una pantalla o display de una estación de trabajo. Según se desprende de la figura 5.1, la entrada de datos hacia el programa se puede hacer de dos maneras diferentes. En primer lugar se puede hacer a través del analizador compatible con SPICE2. Este toma un fichero tipo SPICE2 y lo convierte en la estructura de entrada entendible por el simulador. La finalidad de este analizador no es otra que la de hacer compatible SPICE3 con SPICE2. Sin embargo, existe una segunda manera de hacer la entrada de datos que es ya específica de SPICE3. Consiste en introducir los datos y los comandos de forma interactiva a través del

interfaz de NUTMEG. De esta manera se puede interactuar con el simulador para, por ejemplo, realizar nuevas simulaciones con un mismo circuito. Al igual que para la entrada de datos, para la salida de resultados tenemos dos vías diferentes. La primera consiste en obtener un fichero con formato de salida tipo SPICE2 el cual se podrá representar con cualquier programa de representación de datos (por ejemplo *gnuplot*). Sin embargo SPICE3 proporciona otra manera de obtener los resultados y es a través de NUTMEG. Una vez efectuada una simulación podemos imprimir o representar en pantalla los datos de salida de la simulación sin más que ejecutar uno o varios comandos del interfaz gráfico del NUTMEG.

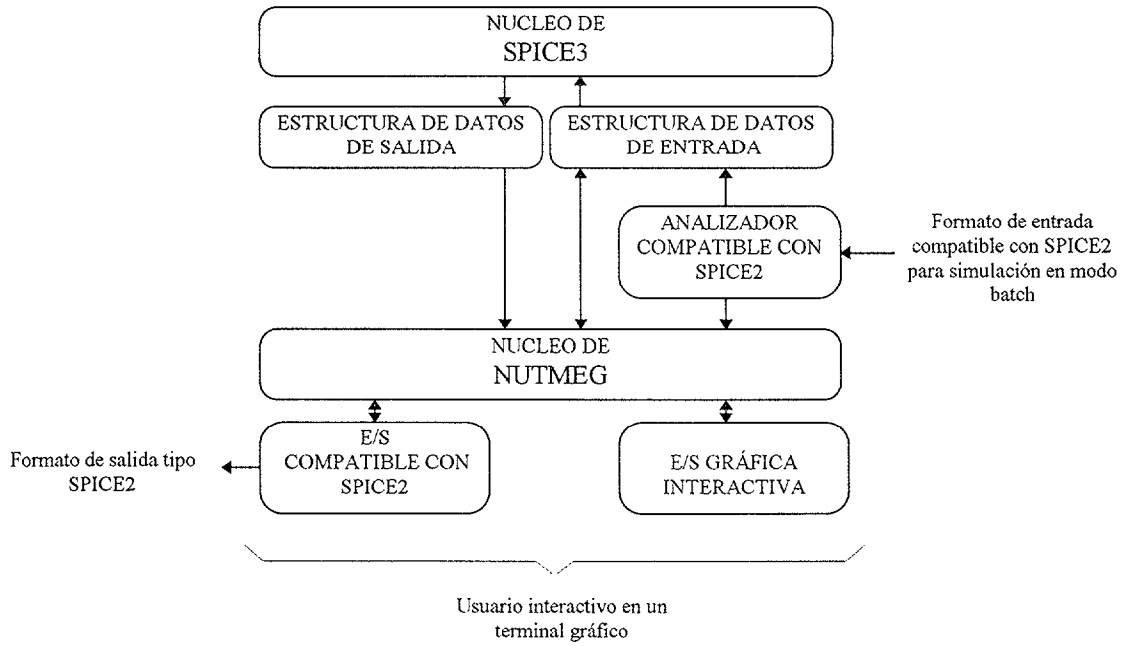


fig. 5.1 Estructura de SPICE3.

Esta descripción que hemos hecho es útil a la hora de entender el funcionamiento de cara al usuario del programa. Sin embargo, para poder introducirnos dentro del programa y hacer modificaciones dentro del mismo tenemos que estudiar su estructura interna de forma más profunda. Para entender la estructura de un programa tan grande como SPICE es necesario dividirlo en módulos para luego estudiarlos de forma individual y finalmente observar como interactuan unos con otros para producir los resultados finales.

Debido al siempre cambiante mundo del diseño de circuitos y las nuevas tecnologías, las técnicas de simulación deben ser mejoradas constantemente. Por

tanto se requieren simuladores que sean lo más flexibles posible. Esto implica la posibilidad de poder integrar nuevos dispositivos, nuevos tipos de simulación, y nuevas vías de introducir y extraer datos. Con esta finalidad, SPICE3 se ha diseñado de forma modular, lo cual hace posible hacer esas tareas con el menor trabajo posible.

Para acercarnos un poco más a la estructura interna del programa a continuación pasamos a describir el programa haciéndolo desde tres puntos de vista diferentes. En primer lugar se realiza la descripción desde la perspectiva de la estructura básica de llamada del programa. De esta manera tenemos una estructura como la de la figura 5.2. En esta figura el bloque llamado "Dispositivos" representa todos los paquetes relacionados con los dispositivos. Esos paquetes utilizan y son utilizados por los algoritmos numéricos de SPICE3. Tanto las rutinas de dispositivos como las numéricas manipulan la matriz *sparse* a través del paquete "Matriz". Los algoritmos numéricos y de simulación se entremezclan ya que están íntimamente relacionados y ambos tienen un gran efecto el uno en el otro. El interfaz con el usuario sólo ha de saber de los puntos de entrada y de salida de las rutinas de control de la simulación.

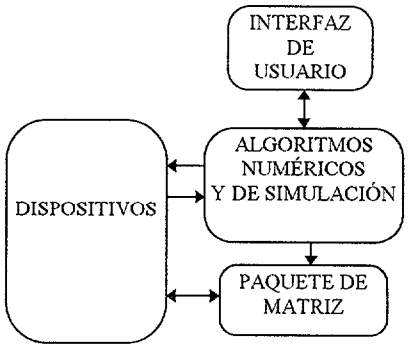


fig. 5.2 Estructura básica de llamada del programa.

En segundo lugar podemos estudiar el programa desde un punto de vista descriptivo. Según este punto de vista el cual se muestra en la figura 5.3, podemos describir el programa dividiéndolo en paquetes cada uno de los cuales consiste en una o más listas de descriptores de parámetros y en un conjunto de punteros a funciones que implementan parte de las capacidades de ese paquete en concreto. Toda esa información se recolecta en una estructura simple y se exporta a los paquetes que estén por encima. De esta forma tenemos en los niveles inferiores unos descriptores de parámetros y subrrutinas los cuales pasan a formar parte a su vez de los descriptores de análisis y los descriptores de dispositivos. Ambos descriptores junto con una serie de subrrutinas pasan a formar parte del

simulador. De forma similar, el paquete denominado *front-end* tiene asociado una serie de rutinas cuyos punteros son recolectados en una estructura que se exporta al simulador durante la inicialización. En tiempo de ejecución tanto el *front-end* como el simulador utilizan los descriptores que le proporciona el otro para determinar las capacidades disponibles, los parámetros utilizados por las sucesivas llamadas, y los tipos de argumentos pedidos por los parámetros.

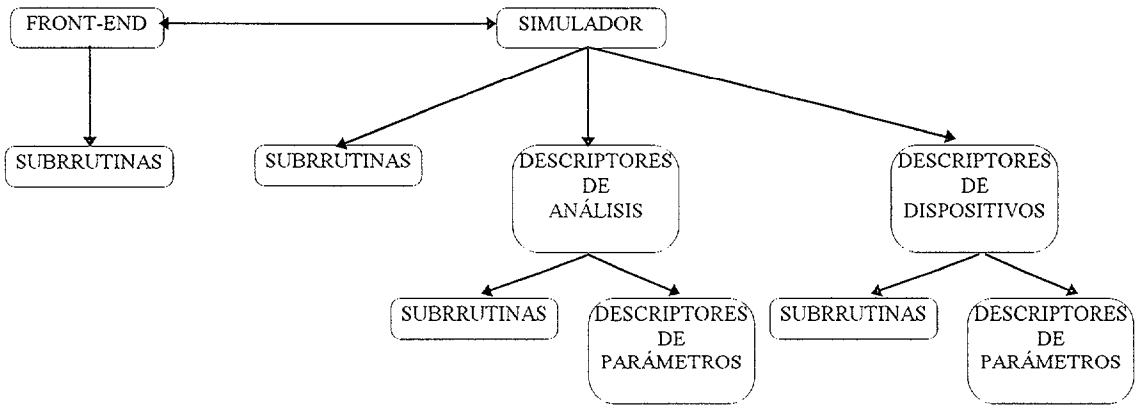


fig. 5.3 Estructura del Programa desde el punto de vista descriptivo.

Finalmente, podemos estudiar el programa teniendo en cuenta la información que debe almacenar de forma interna. Así tenemos una estructura como la que se muestra en la figura 5.4. En esta figura, la estructura "Circuito" se utiliza para encapsular todos los datos relacionados con un único circuito. Esta estructura contiene una considerable cantidad de información, así como punteros a estructuras más especializadas para paquetes individuales. Las otras estructuras utilizadas para almacenar datos son privadas de un paquete único aunque a veces podrían ser referenciadas de forma indirecta por otros paquetes a través de punteros. Por ejemplo, la estructura "Matriz" es completamente privada del paquete de matrices *sparse*, sin embargo, otros paquetes pueden obtener punteros a lugares específicos dentro de la matriz *sparse*. Los dispositivos tienen una estructura de datos más complicada la cual se verá en detalle más tarde. Los datos que son renombrados en cada iteración, el RHS y los vectores solución, se mantienen en un par de *arrays* gestionados en memoria de forma dinámica los cuales son referenciados desde múltiples partes del programa. Los datos que describen cada análisis se mantienen en una lista de estructuras dependientes del tipo de análisis para el uso privado del código que implementa cada tipo de análisis. Por último, las descripciones de los nodos del circuito se guardan en una tabla de nodos la cual es mantenida por un pequeño número de rutinas dentro del propio paquete "Circuito".

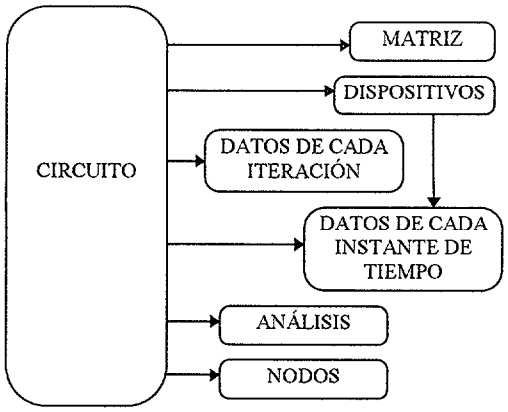


fig. 5.4 Estructura del programa según la información almacenada internamente.

La arquitectura global de SPICE3 es relativamente simple. El programa se divide en un grupo de bloques independientes o paquetes. Cada uno de los cuales sólo conoce de su interfase con los otros paquetes y de las operaciones que el realiza. Esto permite que los paquetes puedan ser mantenidos, mejorados, o reemplazados de forma independiente. La clave para entender el funcionamiento y la estructura de SPICE3 como un todo es echar un vistazo global de todos esos paquetes y sus estructuras de datos correspondientes, así como su interacción. Los detalles de esos interfaces se encuentran en la documentación y se muestran de forma evidente cuando se observa el código que utiliza un paquete determinado. Lo más importante es entender lo suficiente para saber que paquete hay que mirar cuando se necesita una funcionalidad determinada o cuando se encuentra un error que aparece asociado a un tipo de operación particular.

5.1 Principales Estructuras de Datos

El primer paso consiste en entender las estructuras de datos básicas y su radio de alcance. En la mayoría de los casos la estructura del programa sigue la estructura de datos, y por lo tanto es más fácil de entender si conocemos bien las estructuras de datos.

El área de datos del programa está dividido en tres categorías:

- Datos estáticos, los cuales se utilizan para describir módulos del simulador a otros módulos del simulador, el simulador al *front-end*, y ciertas constantes físicas. Este área se define de antemano y se inicializa en varias partes del

simulador, siendo posteriormente almacenadas como variables estáticas globales. Todas las variables globales de que dispone este simulador son constantes.

- Otras variables que podrían ser normalmente variables globales en un simulador. En SPICE3 no hay variables globales. Las variables más usadas se han empaquetado en una estructura de datos, de forma que a la práctica mayoría de las rutinas se les pasa un puntero de dicha estructura lo cual permite que puedan existir múltiples replicas de ella o instancias de forma independiente. Esto facilita el uso del programa en múltiples subcircuitos o como un analizador de subcircuitos dentro de un sistema mayor. Esta estructura de datos es la estructura **CKTcircuit**. La mayor parte de las rutinas del simulador conocen esta estructura lo cual equivale a conocer las variables "globales" del circuito.

- Por último, cada paquete o grupo de rutinas puede tener sus propias estructuras privadas las cuales se definen sin que afecten a cualquier otra parte del código.

Algunos paquetes, como por ejemplo el paquete de operaciones numéricas, simplemente almacenan sus variables en la estructura **CKTcircuit**; por el contrario, otros tienen sus propias estructuras privadas donde pueden almacenar diferentes réplicas o instancias de ella. En las siguientes secciones se presenta una visión general de las estructuras utilizadas por SPICE3. Esta no pretende ser una visión detallada pues sobrepasa los objetivos del presente trabajo. Para encontrar más detalles sobre dichas estructuras remitimos al lector a [M89/44].

5.1.1 Estructura CKTcircuit

Esta es la estructura de datos principal de SPICE3. Todas las variables relacionadas con la descripción y funcionamiento del circuito se puede encontrar dentro de esta estructura, sus subestructuras, o a través de los punteros de esta estructura. Como se ha mencionado anteriormente, SPICE3 no tiene variables globales como tales sino que dichas variables "globales" pasan a formar parte de la estructura **CKTcircuit**. Esto se debe a la decisión que tomaron los creadores de SPICE3 de que pueda manejar circuitos completamente diferentes. Además la estructura **CKTcircuit** contiene las variables de aquellos algoritmos o paquetes

que no tienen suficiente número de variables como para generar una estructura de datos privada para ellos. Por estas razones, esta estructura es la más conocida y a su vez la más frecuentemente cambiada a la hora de hacer cambios del programa en si.

5.1.2 Estructuras para las Matrices *Sparse*

El sistema de matrices *sparse* utiliza dos estructuras para representar las matrices. Esas estructuras son *privadas* del paquete de matrices *sparse*. A otras partes de SPICE3 se le permite tener punteros a esas estructuras, pero el uso de cualquiera de sus campos se restringe a dicho paquete de forma que los cambios en la estructura sólo afectan al paquete de matrices.

La primera estructura, **SMPmatrix**, se utiliza para proporcionar una descripción global de la matriz. Una única réplica o instancia de esta estructura representa una matriz *sparse* entera. Esta estructura contiene el esqueleto o armazón básico de la matriz, pero no los valores actuales de los elementos o los detalles de la estructura cero/no-cero interna. En cambio, en esta estructura si que se encuentran punteros a los primeros elementos no cero de cada fila y columna, así como información de mapeo para convertir los números de las filas y las columnas utilizados internamente y los utilizados por el programa llamante. Esto proporciona un acceso razonablemente rápido a cualquier parte de la matriz.

La segunda estructura, **SMPelement**, se utiliza para representar los términos no cero de la matriz. Cada par fila columna que ha tenido alguna vez un valor no nulo tendrá asociado una réplica o instancia de esta estructura que lo describe y contiene sus valores. Todas las estructuras **SMPelement** que representan elementos en la misma fila están enlazadas en orden ascendente del número de columna, y todas las estructuras **SMPelement** que representan elementos en la misma columna están enlazadas en orden ascendente del número de fila. Como vemos, los valores actuales de cada elemento de la matriz se almacenan en esta estructura.

5.1.3 Estructuras para los Análisis

SPICE3 utiliza estas estructuras para mantener la información sobre los análisis que van a ser llevados a cabo. Cada análisis específico que va a ser llevado a cabo por SPICE3 se denomina trabajo (*job*) y tiene una estructura de datos propia. Todos los trabajos tienen un prefijo estándar en su estructura de datos de forma que el *software* de los niveles superiores puede examinar los trabajos y organizarlos en grupos para ejecutarlos todos a la vez. Estos grupos de trabajos se denominan tareas (*tasks*). Además, cada tipo de análisis concreto posee una estructura que lo describe.

5.1.3.1 Estructura SPICEanalisis

La estructura **SPICEanalisis** básicamente proporciona información a las rutinas de los niveles superiores sobre los análisis que SPICE3 es capaz de llevar a cabo.

5.1.3.2 Estructura TSKtask

La estructura **TSKtask** define un conjunto de análisis que se van a ejecutar juntos. Los análisis compartirán elementos tales como los valores opcionales definidos por el usuario a través del comando *.options*. Las tareas consistirán en un conjunto de valores relacionados con la tarea completa y una lista enlazada de trabajos individuales a realizar como parte de la tarea. Por conveniencia y por presentar un interfaz más uniforme hacia el *front-end*, el comando *.options* se presenta como un análisis más, y por tanto estará en la lista enlazada de trabajos a ejecutar como un caso especial. Sólo podrá haber un trabajo *.options* en la tarea, y los valores de todos los comandos *.options* se juntan para crear ese único trabajo. Dichas opciones deben ser accesibles de forma rápida al comienzo de la tarea, de forma que el análisis *.options* ha de ser designado especialmente y colocado directamente en la estructura **TSKtask**. El análisis *.options* utiliza entonces directamente la estructura **TSKtask** como su propia estructura de datos de trabajo en vez de tener una propia.

5.1.3.3 Estructura JOB

La estructura **job** se utiliza para almacenar la información necesaria para ejecutar un único análisis. Realmente se trata de una estructura prefijo, y se incluye como primer elemento de cada estructura de datos de un análisis específico. Esto permite al código el manipular las estructuras sin conocer los detalles de cada estructura de datos específica de un tipo de análisis.

5.1.3.4 Estructuras Específicas de Cada Tipo de Análisis

El resto de las estructuras de datos necesarias para los análisis son específicas de cada tipo de análisis concreto y almacenan la información específica de cada uno de ellos. Estas estructuras de datos son:

- Estructura **ACAN**: contiene las estructuras de datos necesarias para controlar los análisis en ac.

- Estructura **PZAN**: contiene las estructuras de datos necesarias para controlar los análisis de polos y ceros. Esta estructura necesita el concurso de otra estructura llamada **root** que no es más que la definición de un número complejo con su parte real e imaginaria por separado. De esta forma podemos almacenar los polos y los ceros como números complejos.

- Estructura **OP**: contiene simplemente un indicador que indica que se va ha hacer un análisis de un punto de operación. Debido a que este análisis es muy sencillo esta estructura no necesita más parámetros adicionales.

- Estructura **TFan**: contiene las estructuras de datos necesarias para controlar los análisis de funciones de transferencia.

- Estructura **TRANan**: contiene las estructuras de datos necesarias para controlar los análisis en régimen transitorio.

- Estructura **TRCV**: contiene las estructuras de datos necesarias para controlar los análisis de curvas de transferencia.

- Estructura **SENstruct**: contiene las estructuras de datos necesarias para controlar los análisis de sensibilidad.

5.1.4 Estructuras para los Dispositivos

Los dispositivos en SPICE3 forman una clase de objetos que tienen un interfaz estandarizado. Cada tipo de dispositivo viene representado por tres estructuras que describen sus necesidades y sus capacidades. La clase entera de estructuras tiene un conjunto de propiedades en común las cuales permiten que se puedan realizar una serie de operaciones sin saber a que tipo de dispositivo se refiere. Los detalles adicionales de la estructura se dejan a la propia implementación del dispositivo por lo que pueden variar mucho entre dispositivos diferentes. Para cada tipo de dispositivo existe una estructura de datos bidimensional como la que se muestra en la figura 5.5, la cual contiene toda la información de todos los modelos y de todos los elementos representados por dicho modelo (instancias). Esta estructura consiste en una lista enlazada de modelos de un tipo determinado de dispositivo, conteniendo cada uno de ellos una lista enlazada de los elementos o instancias de ese modelo específico que existen en el circuito. Por ejemplo, supongamos que la figura 5.5 representa los transistores bipolares de un circuito. En este caso, tendríamos que hay 9 transistores bipolares de los cuales 3 son por ejemplo del tipo Q2N2907A, uno del tipo Q2N2222, 3 del tipo Q2N3904 y finalmente 2 del tipo Q2N3906.

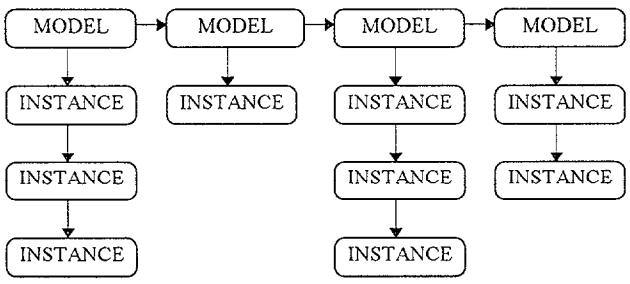


fig. 5.5 Estructura de datos para cada dispositivo.

5.1.4.1 Estructura DEVmodel

Esta estructura contiene todas las variables de cada modelo. Por cada modelo de cada tipo de dispositivo diferente que haya en el circuito se crea una copia o instancia de esta estructura. La mayor parte de la estructura es

especificada por el que implementa el dispositivo, sin embargo, se requiere un prefijo estándar en cada estructura que permita al código genérico atravesar la lista enlazada de modelos para localizar los modelos y las instancias específicas que necesite. Este prefijo contiene la lista enlazada de punteros, el nombre del modelo, y el tipo del modelo.

5.1.4.2 Estructura DEVinstance

Esta estructura contiene todas las variables de cada instancia. Cada instancia de un modelo del circuito estará representada por una estructura instancia. Esta estructura contiene la información básica necesaria para enlazarla a la estructura de datos estándar así como el nombre de la instancia, un puntero hacia el modelo, y cualquier información específica para el tipo de dispositivo particular que el implementador del modelo estime necesaria.

5.1.4.3 Estructura SPICEdev

Esta es una estructura estática utilizada para describir el dispositivo. Por cada modelo de cada tipo de dispositivo diferente definido en SPICE3 se crea una copia o instancia de esta estructura. Esta estructura contiene punteros a todas las funciones utilizadas para implementar el dispositivo así como la información requerida por el *front-end.* Esta estructura es la única información sobre el dispositivo disponible al resto del programa.

5.1.5 Estructuras de Entrada

Son las estructuras utilizadas por el analizador sintáctico de entrada para mantener la información sobre el circuito que se está analizando sintácticamente en un momento dado. Por ejemplo una de las estructuras más importantes dentro de las estructuras de entrada es la estructura **INPtables**. Para cada circuito que va a ser analizado sintácticamente se guarda una copia o instancia de esta estructura y se pasa a todas las rutinas de entrada para ayudar en el análisis sintáctico. Esta estructura está formada principalmente por una serie de punteros que apuntan hacia los modelos que se utilizan por defecto para cada dispositivo.

5.1.6 Estructuras de Interfase

Estas estructuras son las que utiliza el simulador para comunicarse con el *front-end* y el *back-end.*

5.2 Control de Flujo

El control de flujo del simulador es bastante similar a las estructuras de datos. El *front-end* realiza las funciones de análisis sintáctico y de interfaz con el usuario. Utilizando las especificaciones del interfase entre el *front-end* y el simulador, el *front-end* pasa al paquete de circuitos la descripción del circuito y los análisis que se desean realizar. En muchos casos la información es específica a un dispositivo o análisis particular, por tanto, una vez encontradas las estructuras de datos apropiadas, se llaman las rutinas específicas de los análisis o los dispositivos para llevar a cabo la simulación actual. Cuando se llama al análisis actual, el paquete de circuitos llama a los paquetes específicos de los dispositivos para realizar todas las operaciones necesarias de inicialización así como inicializar la matriz *sparse* para la simulación. Los paquetes específicos de los dispositivos utilizan después el paquete de matrices *sparse* para recolectar los punteros a las localizaciones de las matrices específicas que van a ser referenciadas de forma regular. Por último, se llama al paquete de iteración numérica y este repite el método de iteración de Newton-Raphson en cada punto que va a formar parte de la solución, utilizando las rutinas específicas de los dispositivos y de las matrices para realizar las operaciones a niveles inferiores.

5.3 Paquetes

En este apartado se describen los paquetes internos de SPICE3 utilizados para construir el programa entero. Cada uno de esos paquetes es independiente y funciona sin tener en cuenta los detalles de otros paquetes. Esta no pretende ser una descripción detallada sino una visión general que permita al lector hacerse una idea de lo que hace cada paquete. Sin embargo, nos centraremos en aquellos paquetes que sean más importantes para la comprensión del presente trabajo. Para encontrar más detalles sobre los paquetes que componen a SPICE3 remitimos al lector a [M89/44].

5.3.1 Paquete de Manejo de Circuitos

Este conjunto de rutinas lleva a cabo el control del sistema de simulación como un todo. Este paquete guía el secuenciamiento de los análisis y de los bucles a través de los diferentes dispositivos, y proporciona un interfase al *front-end* para permitirle el acceso a las estructuras de datos del simulador.

El paquete CKT o paquete de manejo de circuitos no es en realidad un paquete independiente, sino que consiste en un conjunto de subpaquetes íntimamente relacionados y de unas rutinas de interfase. Este paquete es el único bloque de código de SPICE3 que necesita conocer la estructura entera de SPICE3 y es el punto de interfase para todos los otros paquetes. En todas las rutinas que hacen referencia al paquete CKT se ha de poner en el encabezamiento el fichero "CKTdefs.h". La mayor parte de las rutinas de este paquete nunca son llamadas por nombre ya que son rutinas que son llamadas desde el *front-end* a través del interfase que existe entre el *front-end* y el simulador, y por ello estas rutinas se llaman a través de punteros a funciones. Estas funciones permanecen disponibles de forma que se les puede hacer llamadas tanto internamente como a través del interfase. Los subpaquetes de que consta el paquete CKT son los siguientes:

- *Rutinas de interfase entre el front-end y el simulador*: estas rutinas implementan el interfase estándar existente entre el *front-end* y el simulador.

- *Rutinas de unión*: se utilizan para mantener unido todo el paquete.

- *Rutinas de utilidad general*: estas rutinas se utilizan principalmente como simples utilidades para manipular las estructuras de datos de SPICE3. Eliminan los detalles de las subestructuras pequeñas de el resto del código así como las hace fácil de usar.

- *Rutinas obsoletas*: estas rutinas están presentes por razones históricas. En algunos casos, estas rutinas muestran caminos alternativos para hacer cosas, en otros casos muestran como se hacían en versiones anteriores, y en otros casos, proporcionan capacidades de depuración útiles para aquellos que mantienen el programa.

5.3.2 Paquetes de los Distintos Análisis

Para cada tipo de análisis que puede realizar SPICE3 existe un paquete que lleva el secuenciamiento de los algoritmos básicos del análisis, llamando a las funciones de los dispositivos, a los paquetes de operaciones numéricas, las operaciones de las matrices, y las operaciones de salida necesarias para llevar a cabo el análisis requerido. Cada uno de los subpaquetes implementa un único tipo de análisis a ser realizado por el sistema global. Todos los paquetes de análisis tienen una forma similar. Consisten en tres rutinas y en una tabla de datos. Estas tres rutinas llevan a cabo las tareas de inicializar los parámetros, consulta de parámetros, y realización del análisis.

Los subpaquetes que podemos encontrar en este paquete son los siguientes:

- *Subpaquete para análisis en ac*: el análisis en ac calcula un punto de operación en dc y todos los parámetros de pequeña señal necesarios. Seguidamente hace un barrido por todas las fuentes ac del circuito para un conjunto de frecuencias, calculando la respuesta en ac en régimen de pequeña señal para cada una de esas frecuencias.

- *Subpaquete para análisis del **punto de operación en dc***: el análisis del punto de operación en dc realiza un análisis simple en dc con todas las capacidades en circuito abierto y todas las inductancias cortocircuitadas. El resultado además de presentarse como salida del análisis, se queda guardado en el vector CKTrhsOld para un uso posterior por parte de otros análisis. Este análisis no requiere parámetros adicionales y por tanto puede ser llamado por otros análisis como un paso preliminar a su trabajo.

- *Subpaquete para análisis de **funciones de transferencia***: este paquete permite realizar análisis de funciones de transferencia en dc en régimen de pequeña señal.

- *Subpaquete para **análisis transitorio***: el análisis en régimen transitorio es la simulación más grande y más complicada que se puede realizar

actualmente con SPICE3. Este análisis permite estudiar el comportamiento del circuito con respecto al tiempo.

- *Subpaquete para **análisis de polos y ceros***: este análisis calcula los polos y ceros de una función de transferencia en ac en régimen de pequeña señal.

- *Subpaquete para **análisis de sensibilidad***: el análisis de sensibilidad no es en realidad un análisis separado, sino es un conjunto de modificaciones a otros análisis que extrae la información de sensibilidad de ellos durante el curso de su cálculo. Debido a que es una operación lógica separada y no debería tener ningún efecto en el funcionamiento normal cuando no se ha seleccionado un análisis de sensibilidad, el código para hacer los análisis de sensibilidad es controlado por un conjunto separado de estructuras que son independientes de las estructuras de los otros análisis.

- *Subpaquete para el **control de los análisis***: aunque no se trata estrictamente de un análisis, el control global de varios parámetros que afectan a los análisis, tales como las tolerancias numéricas, se tratan como otro análisis más con el mismo interfase que las rutinas de un análisis cualquiera. La principal diferencia con respecto a los otros paquetes de análisis está en que sus variables poseen valores globales y no particulares de la propia tarea donde se encuentra el análisis.

5.3.3 Paquetes de Dispositivos

Cada dispositivo está representado por un paquete que puede realizar todas las acciones necesarias para la simulación de las instancias y modelos de ese tipo. Este paquete proporciona un interfaz estándar al simulador, lo cual permite tratar con todos los dispositivos de una forma uniforme. Los tipos de acciones disponibles para cada dispositivo deben ser un superconjunto de aquellos que necesita cada dispositivo para ser implementado en SPICE3, lo cual implica que los paquetes tendrán puntos de entrada no usados para casi todos los tipos de dispositivos.

Los dispositivos que actualmente soporta SPICE3 son los siguientes:

- BJT: transistor bipolar modelado según el modelo de Gummel and Poon [GUMPO70], el cual no es más que una extensión del modelo de Ebers-Moll [EBEMO54].
- JFET: JFET modelado según el modelo de Shichman and Hodges [SHIHO68].
- MES: MESFET modelado según el modelo del FET de Statz et al. [STATZ87].
- MOS1: modelo analítico simple y rápido para el MOSFET según el modelo de Shichman and Hodges [SHIHO68].
- MOS2: modelo semiempírico del MOSFET un poco más complejo pero más preciso que el anterior definido en [VLALI80].
- MOS3: modelo semiempírico del MOSFET mucho más complejo pero mucho más preciso que el anterior definido en [VLALI80].
- MOS6: modelo analítico muy rápido para el MOSFET de canal corto según el modelo descrito en [SAKNE90].
- BSIM1: MOSFET modelado según el modelo BSIM (*Berkeley Short Chanel IGFET*) descrito en detalle en [SHSCK85].
- BSIM2: MOSFET modelado según el modelo BSIM modificado (*Berkeley Short Chanel IGFET*) descrito en detalle en [MINCH90].
- DIO: diodo.
- RES: resistencia.
- CAP: condensador.
- IND: bobina.
- SW: conmutador ideal.
- CSW: conmutador controlado por corriente.
- LTRA: modelo para líneas de transmisión con pérdidas [ROYPE91].
- TRA: línea de transmisión sin pérdidas.
- URC: línea RC uniformemente distribuida.
- ISRC: fuente de corriente.
- VSRC: fuente de tensión.
- CCCS: fuente de corriente controlada por corriente.
- CCVS: fuente de tensión controlada por corriente.
- VCCS: fuente de corriente controlada por tensión.
- VCVS: fuente de tensión controlada por tensión.
- ASRC: fuente corriente-tensión arbitraria.

- HEMT: modelo del HEMT desarrollado por el FhGIAF derivado a partir del modelo de Angelov. La inclusión de este modelo en SPICE3 es el objeto del presente trabajo.

En este apartado no vamos a describir ningún paquete de dispositivo puesto que esto está más allá de los objetivos de este trabajo. En el capitulo 6 veremos esto con el dispositivo HEMT.

5.3.4 Paquete de Cálculos Numéricos

El paquete NI o paquete de cálculos numéricos, contiene los algoritmos numéricos de SPICE. Aunque originalmente sólo contenían las rutinas de Integración Numérica, en la actualidad contienen la mayoría de las rutinas numéricas, incluyendo el método iterativo de Newton-Raphson.

5.3.5 Paquetes de Matrices *Sparse*

Este paquete maneja las matrices *sparse* de los diferentes tipos que podemos encontrar a la hora de simular cualquier circuito eléctrico. No se trata de un paquete general, sino que debe ser ajustado para cada aplicación específica. En este caso, este paquete, también llamado paquete SMP (*Sparse Matrix Package*), ha sido especialmente ajustado por los creadores de SPICE3 para la resolución de problemas MNA (análisis nodal modificado). El paquete guarda matrices en memoria, crea elementos en ellas, realiza reordenación heurística basada en el conocimiento de la estructura de las matrices de simulación del circuito, realiza factorización LU con y sin reordenamiento, y realiza sustitución hacia adelante o hacia atrás según convenga con el fin de dar solución a las ecuaciones del circuito.

5.3.6 Paquete de Entrada

El paquete INP o paquete de entrada, es propiamente dicho una parte del *front-end*, pero fue originalmente desarrollado como parte del simulador SPICE3 antes de ser dividido completamente y quedar como está ahora. Este paquete proporciona las herramientas necesarias para analizar sintácticamente el formato

de entrada tipo SPICE2 y producir las llamadas a subrrutinas requeridas por el estándar existente entre el *front-end* y el simulador.

El principal componente del analizador sintáctico está en el fichero "INPpas2.c" el cual controla los principales pasos del análisis sintáctico. Durante la primera pasada, SPICE3 debe recolectar todas las tarjetas tipo *.model* que existen en el circuito ya que, a diferencia de SPICE2, SPICE3 necesita tener descritos todos los modelos antes de crear todas las instancias de los modelos. Durante la segunda pasada, se realiza el análisis sintáctico del resto de las tarjetas del circuito.

5.4 Estructura de Ficheros de SPICE3

Para acabar con el estudio de la estructura de SPICE3 sólo nos queda describir su organización de ficheros. En este punto hay que decir que los diseñadores de SPICE3 han utilizado la nomenclatura estándar del MS-DOS, es decir, los nombres de los ficheros están compuestos por un nombre principal de ocho caracteres como máximo, y una extensión de tres caracteres. El fin perseguido es la compatibilidad del simulador con sistemas basados en MS-DOS.

En la figura 5.6 se puede observar la estructura de ficheros de SPICE3. El directorio del nivel más alto "spice3e2/" contiene el fichero "readme" de ayuda para la instalación y un fichero "makedefs" utilizado por el comando de compilación "build". En todos los directorios que contienen ficheros fuente del programa vamos a encontrar un fichero "makedefs" parecido al que tenemos aquí. El resto de los subdirectorios de SPICE3 cuelgan de este directorio.

En primer lugar encontramos el subdirectorio "lib/". Este subdirectorio contiene una serie de ficheros estándares para SPICE, tales como los ficheros de ayuda y los ficheros que proporcionan las capacidades MFB (interfaz independiente del terminal gráfico). De esta forma dentro del subdirectorio "lib/" podemos encontrar lo siguiente:

- helpdir/spice.txt: información *on-line* de SPICE3.
- helpdir/spice.idx: indice del fichero spice.txt.
- helpdir/nutmeg.txt: información *on-line* de NUTMEG.
- helpdir/nutmeg.idx: índice del fichero nutmeg.txt.

- scripts/spinit: comandos de SPICE y de NUTMEG que se ejecutan al comienzo de la sesión.
- scripts/setplot: fichero de ejecución por lotes para el comando "setplot"
- news: mensaje de comienzo de sesión de elección del usuario.
- mfbcap: base de datos que proporciona las capacidades MFB (interfaz independiente del terminal gráfico).

El siguiente subdirectorio que encontramos es el de ejemplos "examples/", el cual contiene una serie de ejemplos de ficheros de entrada de SPICE.

El subdirectorio "man/" contiene las páginas de manual estilo UNIX de SPICE3. Estas páginas no se instalan automáticamente por lo que si se quiere que estén disponibles deberán ser instaladas a mano por el usuario.

En el subdirectorio "conf/" podemos encontrar los ficheros de configuración para las diferentes plataformas donde se puede instalar SPICE3 y para los diferentes compiladores que se pueden utilizar.

En el subdirectorio "util/" encontramos sólo dos ficheros. El primero de ellos, "internal", nos describe muy por encima la estructura de ficheros de SPICE3 y los cambios internos que se han producido respecto a las últimas versiones. En el segundo fichero, "porting", podemos encontrar información acerca de como hacer la migración de SPICE3 desde un sistema operativo a otro.

Otro subdirectorio que encontramos es el de utilidades o "util/", el cual contiene una serie de ficheros de ejecución por lotes útiles a la hora de compilar el programa. Por ejemplo, entre ellos podemos encontrar el fichero de ejecución por lotes "build", el cual ejecuta de forma recursiva el comando "make" que compila todo el programa según lo indicado en los ficheros "makedefs" que encontramos en todos los directorios que tienen ficheros fuente de programa.

Por último, encontramos el subdirectorio "src/". Este directorio contiene todo el código fuente C del programa. Dentro de este directorio podemos encontrar las siguientes subdivisiones:

- "src/lib/" es la porción de librería o *toolkit* de SPICE3. En él podemos encontrar separados en subdirectorios todos los paquetes que hemos descrito en la sección anterior (sec. 5.3). Destacar la inclusión en la librería de

dispositivos el directorio "src/lib/dev/hemt" donde se encuentra el código destinado a modelar los transistores HFET.

- "src/bin/" contiene el código fuente de todos los programas ejecutables. En este directorio podemos encontrar el fichero "main.c" el cual es utilizado por SPICE3, NUTMEG, BSPICE, y CSPICE. El resto de los ficheros más importantes que se encuentran en este directorio son:

    - "spice3": es el simulador válido solamente para UNIX.
    - "nutmeg": programa independiente de análisis de datos.
    - "bspice": simulador en modo *batch* válido solamente para MS-DOS. Para ejecutar el SPICE en modo *batch* o por lotes se utiliza un comando del tipo "bspice < input.cir" generando a la salida el fichero "rawspice.raw" el cual puede ser leído por NUTMEG.
    - "cspice" simulador en modo *batch* con entrada tipo SPICE2 válido solamente para MS-DOS. El comando "cspice < input.cir" genera ficheros de representación gráfica con caracteres ASCII (*asciiplots*) para los comandos *.plot*.
    - "help" se trata de un programa independiente para visualizar ayudas.
    - "proc2mod" convierte ficheros de caracterización de proceso en la definición del modelo BSIM1 de los transistores MOS
    - "sconvert" convierte ficheros de datos de SPICE entre formato ASCII y formato binario.
    - "multidec" es una utilidad para descomponer líneas de transmisión con perdidas acopladas en sus equivalentes no acopladas.
    - "tune" contiene las rutas donde se encuentran los ficheros compilados, es decir, donde se colocan los ficheros ejecutables una vez compilado el programa.

- "src/include" en donde se encuentran todas las librerías disponibles en SPICE3, es decir, en este directorio encontramos todos los ficheros .h propios de SPICE3.

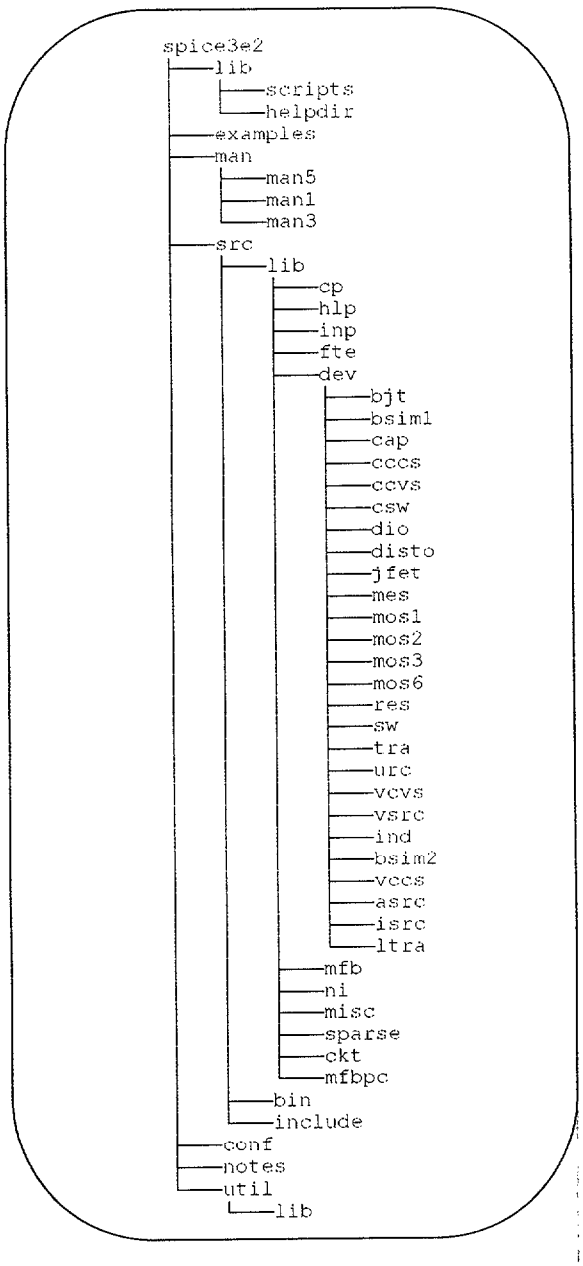


fig. 5.6 Estructura de ficheros de SPICE3.

Capítulo 6

6. Introducción del Modelo del HFET en SPICE3

En los capítulos anteriores hemos presentado cómo se modelan los HEMTs y la estructura básica de SPICE3. Una vez realizadas esas tareas preliminares, pasamos a llevar a cabo, en el presente capítulo, la introducción de dicho modelo en SPICE3. De la comprensión de los capítulos anteriores dependerá en gran medida el éxito de la tarea que pasamos a describir.

Para introducir un nuevo dispositivo dentro de SPICE3 hay, en principio, dos alternativas. La primera consiste en crear todos los ficheros fuente del dispositivo e integrarlo dentro de la estructura global del programa. Esta alternativa, además de lenta, puede no ser fructífera por su complejidad. La segunda alternativa consiste en hacer uso de un dispositivo ya implementado [PSPIC90]. En este caso, el primer problema que se plantea es seleccionar el dispositivo que más se parezca al que queremos introducir. Así, para el caso de los HEMTs tenemos básicamente dos posibilidades. La primera de ellas consiste en basarnos en la topología de los MOSFETs ya que estos transistores son los que se encuentran más y mejor modelados en SPICE y además, su circuito equivalente es bastante parecido al de los HEMTs. Sin embargo, los MOSFETs son dispositivos de 4 terminales, ya que en ellos se modela también el terminal de Substrato, mientras que nuestro modelo del HEMT sólo incluye 3. De esta forma, si se utiliza el MOSFET para modelar el HEMT, dejando la contribución del Substrato a cero, en la matriz *sparse* correspondiente aparecen una serie de ceros que hacen que el sistema no converja a solución alguna.

Por tanto, nos decantamos por la segunda posibilidad que consiste en utilizar el modelo de los MESFETs. Este tipo de transistor tiene la ventaja de poseer tres terminales al igual que los HEMTs y además su funcionamiento se parece bastante (de hecho, el modelo de Angelov es válido para ambos dispositivos).

Sin embargo, a primera vista parece que si se utilizan los MESFETs para modelar a los HEMTs, se eliminarán los primeros del programa. Este problema se evita haciendo uso de un parámetro adicional que es el nivel o LEVEL. Las rutinas de entrada de SPICE leen este parámetro y según sea su valor entenderán que se trata de un MESFET o de un HEMT. Por ejemplo, para nuestro caso el HEMT constituye el LEVEL 2 de los modelos de los MESFETs, mientras que para otro valor del parámetro LEVEL, SPICE entenderá que se trata de un MESFET.

Una vez elegido el dispositivo en el que nos basaremos, tenemos que ver la estructura de dicho dispositivo dentro del simulador. Ese será el objetivo del siguiente apartado.

6.1 Estructura de los Paquetes de Dispositivo

Según se vio en el capítulo 5, los paquetes correspondientes a cada uno de los dispositivos se encuentran en el subdirectorio "src/lib/dev" (ver figura 5.6). Dentro de cada uno de los subdirectorios correspondientes a cada dispositivo encontramos los siguientes ficheros fuente [M89/45]:

- DEV.C: este fichero contiene las estructuras de datos utilizadas para describir el dispositivo. Así, en él podemos encontrar la estructura *DEVpTable* que contiene información para describir una única instancia, y la estructura *DEVmPTable* la cual contiene los datos requeridos para describir a un modelo.
- DEVPARA.C: se trata de una función que, dada una instancia específica de un dispositivo, un parámetro, y un valor, asigna dicho valor a dicho campo del dispositivo.
- DEVMPAR.C: esta función es la equivalente a la anterior, pero aplicada a los parámetros de modelo.

- DEVLOAD.C: esta función es la que lleva a cabo el bucle interior de los análisis en dc y en régimen transitorio para cargar la matriz *sparse* y el RHS (ver figura 4.1).
- DEVSETU.C: esta función se ejecuta una sola vez durante el preprocesamiento de los parámetros. En ella se llevan a cabo todas las operaciones que sólo se tienen que realizar una vez, tales como la asignación de memoria y punteros para los valores de la matriz *sparse*, así como la asignación de espacio en la tabla de estado para los datos que se deben almacenar en cada intervalo de tiempo.
- DEVTEMP.C: en esta función es donde se realiza la mayor parte del preprocesamiento de los datos. Esta función se ejecuta antes de que comience la simulación y siempre que los parámetros o la temperatura de la simulación hayan sido cambiados.
- DEVTRUN.C: esta función realiza el cálculo del error de truncación sobre cada elemento de almacenamiento o componentes de elementos.
- DEVACL.C: esta función es equivalente a DEVLOAD solo que se utiliza en los análisis en ac donde los datos a almacenar son de tipo complejo.
- DEVDEST.C: esta función se utiliza para desmantelar todas las estructuras con lo que se libera todo el espacio utilizado por los modelos y las instancias del dispositivo.
- DEVMDEL.C: esta función se utiliza para desmantelar las estructuras de datos liberando un único modelo y todas las instancias relacionadas con él, dejando solos al resto de los dispositivos del mismo tipo
- DEVDEL.C: esta función elimina una única instancia de las estructuras de datos, dejando todo lo demás como estaba.
- DEVGETI.C: esta función se utiliza para dar valor a las condiciones iniciales del dispositivo a partir de las condiciones iniciales de los nodos. Antes de llamar a esta función, el vector RHS se carga con todas las condiciones iniciales de los nodos, de forma que cada dispositivo debe mirar a todos los nodos a los que está conectado y dar valor a sus campos de condiciones iniciales basándose en esas tensiones si el usuario no las ha especificado directamente en el dispositivo.
- DEVASK.C: esta es la función que permite al usuario el acceder a los valores internos del dispositivo.
- DEVMASK.C: esta función es exactamente igual que la anterior pero aplicada a los modelos.

Estas funciones son las encargadas de realizar los análisis más importantes del dispositivo. Sin embargo, pueden existir otras funciones que añaden análisis adicionales al dispositivo. Así, para los análisis de sensibilidad nos podemos encontrar con las funciones DEVSACL.C, DEVSPRT.C, DEVSSET.C, y DEVSVPD.C. Para los análisis de polos y ceros tenemos la función DEVPZLD.C. La función DEVNOI.C es la utilizada para los análisis de ruido. Finalmente, para los análisis de distorsión, es decir, para el estudio de los armónicos, nos podemos encontrar con las funciones DEVDIST.C, y DEVDSET.C.

Además de los ficheros donde se encuentra el código fuente asociado a cada dispositivo, existen también otros ficheros relacionados con el dispositivo.

En primer lugar encontramos los ficheros *include* en donde se declaran las funciones y variables que otras partes de programa tienen que ver. Estos ficheros tienen extensión .h y se encuentran en el subdirectorio "src/lib/include":

- DEVDEFS.H: este fichero contiene las estructuras de datos utilizadas para describir el dispositivo. Así, en él podemos encontrar la estructura *sHEMTinstance* que contiene la declaración de los parámetros necesarios para describir una única instancia, y la estructura *sHEMTmodel* la cual contiene las declaraciones de los parámetros requeridos para describir un modelo. En este fichero podemos encontrar también la definición de las constantes simbólicas utilizadas para referirnos a los parámetros. Estas constantes se agrupan en aquellas que se aplican a las instancias y aquellas que se aplican a los modelos. Los valores exactos que se utilizan para estas constantes simbólicas no son importantes, siendo las únicas restricciones el que sean diferentes unas de otras y que cada uno de los dos conjuntos utilicen valores enteros contiguos de forma que el código compilado sea óptimo.
- DEVEXT.H: este fichero de encabezamiento es el que contiene la declaración de todas las funciones utilizadas en la implementación del dispositivo.
- DEVITF.H: en él se encuentra la estructura *SPICEdev* la cual define la estructura global del dispositivo. Esta estructura se divide en dos secciones principales. La primera parte, *DEVpublic*, contiene la información requerida por el interfase del *front-end*. Así en ella

podemos encontrar información sobre el nombre del dispositivo, descripción del mismo, numero de terminales, etc.. El resto de la estructura tiene la finalidad de permitir a SPICE3 identificar las rutinas específicas que debe llamar para llevar a cabo cualquier operación sobre el dispositivo.

Por otro lado tenemos también los ficheros del paquete de entrada que hacen que el programa detecte que hay un dispositivo nuevo y sea capaz de cargarlo. Estos ficheros se encuentran en el subdirectorio "src/lib/inp" y son:

- INPDOMOD.C: esta función proporciona una equivalencia entre los tipos de modelo que pueden aparecer en la tarjeta *.model*, tales como NMF, y el nombre interno del dispositivo, que en este caso puede ser tanto MES como HEMT dependiendo del valor de la variable LEVEL. Seguidamente, realiza una conversión de dicho nombre en el número asociado a ese tipo de dispositivo haciendo uso de la función *CKTtypelook*. En realidad esta función constituye la primera pasada del análisis del fichero de entrada de SPICE3.
- INPPAS2.C e INP2*x*.C: la función INPPAS2.C realiza la segunda pasada del análisis del fichero de entrada de SPICE3, y, por tanto, es la que realiza la mayor parte del trabajo. Por cada posible primera letra de un dispositivo, existe una sentencia tipo "case" que procesa la línea de entrada del dispositivo. Entonces INPPAS2.C llama a una serie de subrrutinas, una para cada tipo de dispositivo, para hacer el análisis sintáctico (*parsing*) de la misma. De esta manera simplificamos el trabajo de los compiladores ya que se reduce la cantidad de código que debe ser recompilado cuando estamos trabajando en el análisis sintáctico de un único dispositivo. Las subrrutinas que llama la función INPAS2.C tienen el nombre genérico INP2*x*.C, en donde *x* es el carácter por el que empieza el dispositivo. Por ejemplo, para el caso de los MESFETs o los HEMTs, la rutina correspondiente tiene el nombre INP2Z.C ya que cada uno de estros transistores debe comenzar con la letra Z. Estas rutinas se encargan de analizar sintácticamente la primera parte de la línea que normalmente es bastante irregular, y luego utiliza la función *INPdevParse* para procesar el resto de la línea.

Una vez detallado como se define el dispositivo dentro de SPICE pasamos a describir a continuación el proceso de introducción de uno de ellos dentro del programa. En nuestro caso lo haremos con el modelo del HEMT.

6.2 Introducción del Modelo del HFET en SPICE3

Como ya se dijo antes en la introducción a este capítulo, la implementación del modelo del HEMT dentro de SPICE3 la realizaremos basándonos en el modelo del MESFET. Por ello, la primera tarea consistirá en duplicar el directorio correspondiente al MESFET en otro directorio para el HEMT y seguidamente hacer los cambios pertinentes para conseguir que SPICE admita un dispositivo nuevo. Como resultado de esta tarea, obtendremos dos modelos del MESFET idénticos uno de ellos con el nombre del HEMT y el otro llamado MESFET. Una vez hecho esto comprobamos que cuando introducimos en el fichero de entrada de SPICE un transistor MESFET con LEVEL=1 toma el modelo del MESFET, y si el valor del parámetro LEVEL es 2 toma el modelo duplicado del MESFET. A partir de aquí nuestros esfuerzos se enfocarán principalmente en personalizar el modelo duplicado del MESFET para convertirlo en un HEMT con lo que tendremos que modificar todos y cada uno de los ficheros correspondientes a cada análisis.

6.2.1 Crear el dispositivo HEMT a partir de un MESFET

Para llevar a cabo esta tarea se crea en primer lugar un directorio llamado "hemt" dentro del subdirectorio "src/lib/dev" y se copia todo lo que hay en el directorio "mes" en este directorio.

Seguidamente tenemos que modificar todos y cada uno de los ficheros tanto sus nombres como su contenido de forma que todo lo que diga "MES" quede reemplazado por "HEMT" y todo lo que diga "mes" quede reemplazado por "hemt". Hay que tener cuidado de que estos cambios se hagan solamente cuando las palabras "MES" o "mes" hacen referencia a los MESFETs.

El siguiente paso consistirá en crear los ficheros *include* dentro del directorio "src/lib/include". Para ello copiamos los ficheros MESDEFS.H, MESEXT.H, y MESITF.H a los ficheros HEMTDEFS.H, HEMTEXT.H, y

HEMTITF.H. En este caso también tenemos que cambiar todo lo que diga "MES" o "mes" por "HEMT" o "hemt" respectivamente, teniendo igual cuidado que antes en hacer esos cambios sólo cuando se refiera a los MESFETs.

Resta ahora modificar los ficheros de entrada que mencionamos en el apartado anterior para hacer visible a SPICE el nuevo dispositivo. Por tanto deberemos modificar los ficheros INPDOMOD.C y INP2Z.C para que se utilice el modelo del HEMT cuando el parámetro LEVEL es 2, y en caso contrario se utilice el modelo del MESFET. Las modificaciones a estos ficheros se encuentran detalladas en el apéndice A.

Hechas estas modificaciones, el siguiente paso consiste en compilar el programa. El proceso que enunciamos a continuación es el que hay que llevar a cabo para compilar el programa en sistemas UNIX. Para compilar el programa en otras plataformas remitimos al lector al fichero "readme" del directorio "spice3e2/".

Para compilar el programa sobre una plataforma UNIX se edita el fichero "defaults" que está en el directorio "conf/" y se cambian los parámetros que allí se listan para reflejar la configuración de nuestro sistema. Esta tarea no es muy complicada ya que dentro del propio fichero hay una descripción detallada de cada parámetro. Una vez hecho esto, ejecutamos desde el directorio "spice3e2/" el comando de compilación:

```
util/build nombre_dir
```

donde nombre_dir es el nombre del directorio donde queremos que se instale el programa. Con este comando sólo se realiza la compilación dejando los ficheros ejecutables dentro del directorio "/src/bin". Si queremos que, además de compilar el programa, se instalen los ficheros ejecutables dentro del directorio nombre_dir, el comando a ejecutar es:

```
util/build nombre_dir install
```

El proceso de compilación puede durar desde unos 20 minutos a 4 horas dependiendo de la velocidad y la carga del sistema. La primera vez que se instala el programa se compilan todos los ficheros fuente. Sin embargo, en las sucesivas veces que se ejecuta el comando de instalación, sólo se compilan los ficheros que han sido modificados desde la última vez. De esta forma el proceso es más

rápido. Sin embargo, existen ocasiones en las que interesa que se recompile de nuevo todo el programa pues se han modificado ficheros *include*. En estos casos antes de ejecutar el comando de compilación debemos ejecutar el comando:

```
make clean
```

Así se eliminan todos los ficheros objeto, con lo cual, al ejecutar de nuevo el comando de instalar se compila todo el programa completo.

Una vez completado todo el proceso obtenemos un nuevo SPICE3 en el que tenemos dos modelos del MESFET idénticos, uno de ellos con el nombre HEMT. Es el momento de comprobar que cuando introducimos en el fichero de entrada de SPICE un transistor MESFET con el parámetro LEVEL=1 toma el modelo del MESFET, y si el valor del parámetro LEVEL es 2 toma el modelo duplicado del MESFET. Lógicamente, los resultados serán por ahora los mismos. El objetivo de los siguientes apartados será la personalización el modelo duplicado del MESFET para convertirlo en un HEMT. Para ello tendremos que modificar todos los ficheros correspondientes a cada tipo de análisis.

6.2.2 Introducción de los Parámetros de Modelo

En primer lugar debemos determinar qué parámetros del modelo del HEMT son comunes al del MESFET, para de esta forma saber qué nuevos parámetros debemos incluir. Así tenemos que los parámetros comunes son: ALPHA, BETA, LAMBDA, RD, RS, IS, FC; y que los parámetros nuevos son: NP, AFACT, GEXP, CDVC, VC, GAMMA, CDVSB, DELTA, VSB, DXL, VAT, VBT, CGSO, CGS1, VST, VS, MFACT, RG, RGS, RGD, QC1, TD, CDS.

El procedimiento que debemos seguir para incluir un nuevo parámetro en un determinado modelo de dispositivo es el siguiente: en primer lugar debemos editar el fichero HEMTDEFS.H. Como ya se dijo anteriormente, dentro de ese fichero se encuentra la estructura de datos *sHEMTmodel*, que es donde se definen los parámetros de modelo. Por tanto es dentro de esa estructura de datos donde tenemos que definir la nueva variable. En este punto debemos mencionar que SPICE3 trabaja con datos de tipo *double*. Además de la definición de la variable en sí, debemos incluir una variable de tipo *unsigned* útil para que las rutinas de entrada detecten si un parámetro ha sido dado por el usuario o no. De esta

manera, por cada parámetro de entrada tendremos el siguiente par de variables dentro de la estructura *sHEMTmodel*:

```
double HEMTparametro;
unsigned HEMTparametroGiven : 1;
```

Dentro del mismo fichero HEMTDEFS.H también debemos definir el valor de la constante simbólica que ha de ser utilizada para referirnos a tal parámetro. Esto lo haremos dentro del grupo de aquellas que se aplican a los modelos, y siguiendo las pautas de numeración que se mencionan en el apartado 6.1. Siguiendo con el ejemplo anterior, la nueva línea que debemos incluir sería la siguiente:

```
#define HEMT_MOD_PARAMETRO NUMERO
```

El siguiente fichero que debemos editar es el HEMT.C. En él se encuentra la estructura de datos *HEMTmPTable* dentro de la cual debemos incluir una entrada para el nuevo parámetro de la forma:

```
IOP(nombre, const, tipo, descripción),
```

Cada una de estas líneas son una instancia de las macros IP, OP, o IOP. Estas macros se definen en el fichero HEMTDEFS.H y se utilizan para especificar si el parámetro es de Entrada (IP), Salida (OP), o Entrada/Salida (IOP). La función de estas macros es la de acortar el texto que se necesita para describir a un parámetro. El campo de nombre contiene el nombre corto o abreviación que el usuario va a utilizar para referirse al parámetro. El campo const contiene la constante simbólica correspondiente que definimos anteriormente en el fichero HEMTDEFS.H. El campo tipo indica que tipo de variable se trata de entre las que admite el analizador sintáctico de entrada. El tipo puede ser una de las constantes simbólicas IF_INTEGER, IF_REAL, IF_FLAG, IF_NODE, IF_COMPLEX, IF_STRING, IF_INSTANCE, o IF_PARSETREE. Estas constantes simbólicas indican si el parámetro requiere un valor entero, real, ningún valor, un nodo del circuito, un número complejo, una cadena de caracteres, el identificador de otro dispositivo, o un árbol de análisis sintáctico que representa una expresión. Por último, el campo descripción contiene una descripción del parámetro útil para los usuarios que no están familiarizados con la abreviación utilizada. Siguiendo con nuestro ejemplo, la línea a incluir es:

```
IOP( "PAR", HEMT_MOD_PARAMETRO, IF_REAL,"Parámetro ejemplo"),
```

Otro fichero que debemos editar es el HEMTMASK.C. Esta función permite al usuario acceder a los valores internos del modelo y, por tanto, dentro de ella debemos incluir tanto los parámetros de tipo IP como los de tipo IOP. El bloque que debemos incluir por cada nuevo parámetro es el siguiente:

```
case HEMT_MOD_PARAMETRO:
        value->rValue = here->HEMTparametro;
        return (OK);
```

Algo muy parecido tenemos que hacer con el fichero HEMTMPAR.C, salvo que en este caso sólo ha de hacerse con los parámetros de tipo IOP. El bloque de código que hay que introducir es exactamente el mismo que en el caso anterior.

Finalmente, debemos editar el fichero HEMTSETUP.C y definir los valores por defecto de los parámetros.

El proceso que hemos descrito es el que hay que seguir por cada nuevo parámetro de modelo que queramos introducir en el dispositivo. Algunos de los parámetros del HEMT dependen de la anchura del canal (W), la cual es un parámetro de instancia, es decir, depende del transistor en concreto y no del modelo. Sin embargo, esto no ocurre para el caso de los MESFETs ya que para este tipo de transistor la dependencia es con respecto al área del transistor. Por tanto, para acabar con la introducción de los parámetros debemos cambiar en el fichero HEMTDEFS.H el parámetro de instancia de área (*HEMTarea*) por uno de anchura del canal (*HEMTw*) y seguidamente hacer el mismo cambio en todos los ficheros fuente del directorio "/hemt". Este parámetro de anchura del canal ha de ser tenido en cuenta a la hora de implementar los diferentes análisis del modelo para obtener una correcta descripción del mismo.

Una vez hemos introducido todos los parámetros correspondientes al HEMT y hemos comprobado su correcto funcionamiento, podemos dar comienzo a la tarea de cambiar cada una de las funciones que realizan los diferentes análisis. Esta tarea es la más complicada y corresponde a la verdadera introducción del nuevo dispositivo dentro de SPICE3. Es en este punto donde toma relevancia lo visto en el apartado 3.2.1 de adaptación del modelo FhGIAF a SPICE.

6.2.3 Análisis en dc

Como se mencionó en el apartado 6.1, el código para llevar a cabo el análisis en dc se encuentra en el fichero HEMTLOAD.C. Ésta función es la más importante ya que es la encargada de evaluar todas las instancias en cada iteración en los análisis en dc y en régimen transitorio y cargar la matriz *sparse* y el vector RHS con los valores apropiados. Por tanto, dicho fichero será el que vamos a modificar tanto en este apartado como en el siguiente.

Debido a la variedad de las condiciones iniciales y de las condiciones especiales de cada tipo de análisis, esta función puede ser bastante complicada. Básicamente su estructura consta de los siguientes bloques:

- *Bloque de inicialización de parámetros:* en este bloque se precalculan aquellos parámetros que dependen de la anchura del canal o de otros factores.

- *Bloque de inicialización de tensiones y de verificación de bypass:* en este bloque se les da valor a las tensiones de los terminales para la evaluación. Existen seis formas principales de operación de la función HEMTLOAD, las cuales están controladas por el valor de la variable *ckt->CKTmode*. Esos seis casos son los siguientes:

    1. MODEINITFLOAT: Este es el caso más común y es el que se utiliza en las iteraciones a la hora de encontrar la convergencia. Las tensiones de los terminales se obtienen del vector solución anterior *ckt->CKTrhsOld*.
    2. MODEINITPRED: Este es el caso utilizado para la primera iteración en cualquier intervalo de tiempo. En este caso SPICE tiene que predecir las tensiones de los terminales. Para los dispositivos no lineales en los que las tensiones de las uniones han sido salvadas, se lleva a cabo una predicción lineal de las tensiones de las uniones. En otro caso, se utilizarán los valores de del vector solución.
    3. MODEINITTRAN: Este es un caso especial de MODEINITPRED y se utiliza para la primera iteración en el primer instante de tiempo después de la solución en dc. Este modo es necesario ya que para realizar la predicción son necesarias dos soluciones previas.

4. MODEINITFIX: Este es el caso que considera el efecto de las especificaciones *off* en las líneas del dispositivo. A menos que sea necesario poner a *off* los dispositivos, este caso se comporta exactamente igual que MODEINITFLOAT.
5. MODEINITJCT: Este caso se utiliza para la primera iteración del circuito y se utiliza para inicializar las tensiones de las uniones a algo razonable. Cuando no existen condiciones iniciales impuestas por el usuario, los mejores resultados se obtienen inicializando las uniones a V_{t0} o a su equivalente de forma que le sea sencillo al dispositivo el moverse en cualquier dirección en una única iteración.
6. MODEINITSMSIG: Este caso se utiliza para almacenar valores especiales necesarios para los análisis en pequeña señal. Las tensiones de los terminales se obtienen una vez más del vector RHS anterior, pero no se necesita recalcular la nueva matriz y el nuevo RHS.

Una vez obtenidas las tensiones de los terminales se evalúa la posibilidad de hacer un *bypass*, (ver apartado 4.2.2).

- *Bloque de cálculo de corrientes y de sus derivadas:* En este bloque se lleva a cabo el cálculo de las corrientes debidas a los diodos y a las fuentes de corriente que posea el dispositivo. Así mismo en este bloque se realiza también el cálculo de las derivadas de estas corrientes, obteniéndose, por tanto, las conductancias del circuito.

- *Bloque de cálculo de elementos de almacenamiento de cargas:* En este bloque se realiza el cálculo de los elementos de almacenamiento de carga del dispositivo.

- *Bloque de chequeo de convergencia:* Este bloque es el encargado de establecer si la solución está dentro de los límites impuestos por las tolerancias según se estudió en el apartado 4.2.2.

- *Bloque de almacenamiento del vector RHS:* En este bloque se procede a la carga del vector RHS con las corrientes equivalentes en cada nodo dadas por la segunda ley de Kirchoff o ley de las corrientes en los nudos.

- *Bloque de carga de la matriz de conductancias:* Esta es la última operación que se realiza dentro de la función HEMTLOAD y consiste en cargar la matriz *sparse* con los nuevos valores calculados.

Hay que tener en cuenta que la matriz de conductancias de los transistores HEMT es ligeramente diferente a la de los MESFETs. Esto se debe a que el circuito equivalente para los transistores HEMT (fig. 6.1.b) posee la resistencia de puerta cosa que no ocurre para el caso de los MESFETs (fig. 6.1.a).

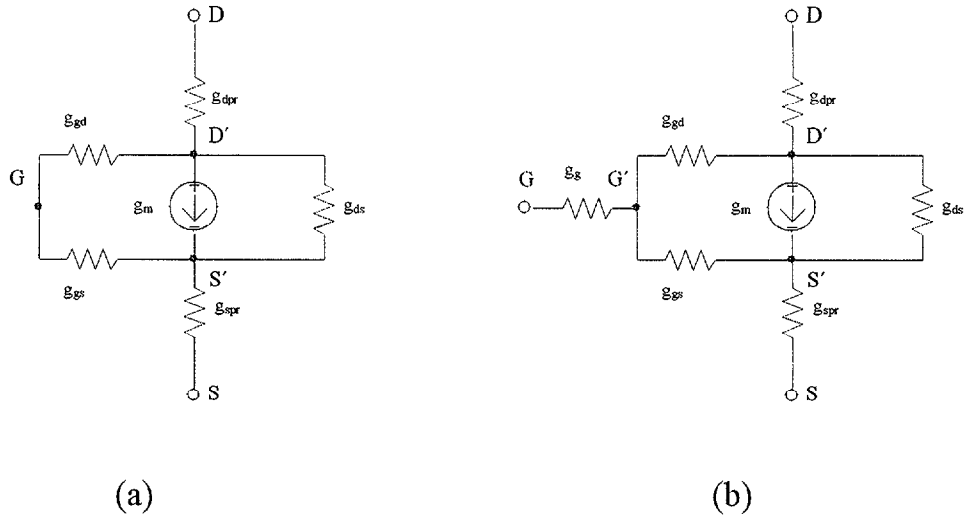


fig. 6.1 Circuitos equivalentes en dc de los transistores MESFETs (a), y HEMTs (b)

Como se vio en el capítulo 4, la matriz de conductancias de un dispositivo no es más que la representación matricial de las conductancias de su circuito equivalente. Por esta razón lo único que debemos hacer es incluir el nodo G' en la definición de cada instancia tipo HEMT y modificar la matriz de conductancias de las mismas. Para hacer esto tenemos que editar el fichero HEMTDEFS.H e incluir una nueva variable de tipo entero dentro de la estructura *sHEMTinstance* llamada *HEMTgatePrimeNode* la cual representa el nodo G'. El siguiente paso consiste en modificar la lista de punteros que define la matriz de conductancias para dar cabida a este nuevo nodo. (Ver cambios en el apéndice A).

Como consecuencia de modificar la matriz de conductancias asociada al dispositivo, debemos modificar el bloque de código encargado de la carga de la matriz de conductancias dentro de la función HEMTLOAD. En este bloque de código se asigna la conductancia asociada a cada nodo del dispositivo al puntero de la matriz de conductancias correspondiente. Es, por tanto, aquí donde se añade

el efecto que introducen las resistencias parásitas. Para ello lo único que debemos hacer es calcular las conductancias asociadas a cada resistencia (basta con hallar su inversa) y colocarlas dentro de la matriz de conductancias. Sin embargo, para el caso de la resistencia de puerta R_g debemos hacer algo más. Debemos editar el fichero HEMTSETUP.C y añadir una sección de código en el que se compruebe si existe o no la resistencia de la puerta. En caso afirmativo, haremos la distinción entre G y G´, pero en caso negativo los nodos G y G´ serán el mismo. Con esto evitaremos problemas de tener ceros en la matriz de conductancias lo cual traería como consecuencia la no convergencia del sistema. Esto sólo lo hemos de hacer para el caso de la puerta ya que es donde hemos incluido un nuevo nodo. Para los casos del drenador y del surtidor, al ser nodos que ya existían en los MESFETs, esto ya esta hecho.

Otra consecuencia de la aparición del nuevo nodo G´ es que debemos modificar la carga del vector RHS. Lo único que debemos hacer es cambiar el nodo *HEMTgateNode* por el nuevo nodo *HEMTgatePrimeNode*.

Una vez incluido el efecto de las resistencias parásitas, sólo nos queda implementar el cálculo de las corrientes de los diodos y de la fuente de corriente para acabar con el análisis en dc. Esto se hace dentro del bloque de cálculo de corrientes y de sus derivadas. Dentro de este bloque debemos establecer los valores de las corrientes de los diodos según las ecuaciones (3.30) y (3.31) y la corriente de la fuente de corriente, tal y como viene dada por la ecuación (3.22). El código que implementa estos cambios se puede observar en el apéndice A.

6.2.4 Análisis en Régimen Transitorio

Como se dijo en el apartado anterior, el análisis en régimen transitorio también lo lleva a cabo la función HEMTLOAD.C. Por tanto hemos de tener en cuenta las consideraciones hechas en el apartado anterior. Así mismo, debemos tener en cuenta también lo dicho en el capítulo 2.

El circuito equivalente que debemos implementar en este caso es el de la figura 3.6. En él aparecen las capacidades de puerta C_{gs} y C_{gd} en serie con las resistencias R_{gs} y R_{gd} respectivamente. Por tanto en el bloque destinado al cálculo de los elementos de almacenamiento de cargas debemos incluir el cálculo de las capacidades de puerta tal y como se expresa en las ecuación (3.26) y de las

cargas asociadas a ellas para las diferentes regiones de funcionamiento tal y como se expresa en las ecuaciones (3.37), (3.38) y (3.39). En este punto deberán ser tenidas en cuenta las condiciones que deben cumplir las variables Q_{c1}, Q_{c2} y Q_{c3} impuestas por las ecuaciones (3.42) y (3.43) para que se cumplan las condiciones de continuidad de la carga y de sus derivadas en las interfases de las tres regiones de funcionamiento.

Seguidamente, efectuamos la integración numérica de las ecuaciones diferenciales correspondientes a las capacidades C_{gs} y C_{gd}. De esta forma, dichas ecuaciones diferenciales se transforman en ecuaciones no lineales simples, lo cual permitirá la consecución del resultado final tal y como se vio en el capítulo 4.

En el circuito equivalente para régimen dinámico aparece también la capacidad de drenador a surtidor , C_{ds}. Esta capacidad tiene un tratamiento especial puesto que no existe en el modelo de los MESFETs y, por tanto, hay que incluirla en las estructuras de datos del programa. Para ello, en primer lugar editamos el fichero HEMTDEFS.H y definimos las variables *HEMTqds* y *HEMTcqds* en el vector de estado. Dicho vector se encuentra dentro de la estructura *sHEMTinstance* que es donde se definen los parámetros de cada instancia. Estas variables darán cuenta de la carga almacenada por la capacidad C_{ds} y la corriente asociada a dicha carga. Al igual que para los parámetros de modelo, para los parámetros de instancia también debemos editar el fichero HEMT.C. En este caso, la estructura de datos dentro de la cual debemos incluir una entrada para el nuevo parámetro es la denominada *HEMTpTable*. Lógicamente, al ser un parámetro de instancia, sólo se podrá poner como de salida (OP). Para permitir al usuario acceder a los valores internos del vector de estado debemos editar el fichero HEMTASK.C y añadir una entrada para cada nuevo elemento de dicho vector, es decir, una para *HEMTqds* y otra para *HEMTcqds*. Dichas entradas tendrán la siguiente forma:

```
case HEMT_MOD_PARAMETRO:
        value->rValue = *(ckt->CKTstate0+here->HEMTparametro);
        return (OK);
```

Por último, editamos el fichero HEMTSETUP.C en donde además de dar a la capacidad C_{ds} un valor por defecto de 0 F, debemos especificar dentro de la variable **states* el número de elementos del vector de estados correcto. En nuestro caso ese número es 15.

Una vez definidos los parámetros *HEMTqds* y *HEMTcqds*, ya estamos en condiciones de llevar a cabo la implementación del cálculo de sus valores dentro de la función HEMTLOAD.C. Esto se hace de forma parecida a como lo hicimos con C_{gs} y C_{gd}. En este caso no tenemos diferentes regiones de funcionamiento ya que C_{ds} no varía con la tensión y por tanto su carga viene dada por la expresión simple:

$$Q = C \cdot V \tag{6.1}$$

Como vimos en 4.3.1, SPICE utiliza el error de truncación local (*LTE*) para medir la precisión del método de integración de las ecuaciones constitutivas de ramas de condensadores y bobinas y controlar el intervalo de tiempo entre muestras. El cálculo del *LTE* se lleva a cabo dentro del fichero HEMTTRUNC.C gracias a la función *CKTerr*. Por tanto, dentro de dicho fichero debemos incluir una entrada del tipo:

```
CKTerr(here->HEMTqds,ckt,timeStep);
```

Una vez tenidos en cuenta todos los elementos de almacenamiento de carga, y para acabar con el modelado del comportamiento del dispositivo en régimen dinámico sólo nos queda incluir las ecuaciones para el cálculo de las conductancias, o lo que es lo mismo, las derivadas de las corrientes. Estas ecuaciones se introducirán, por tanto, dentro del bloque destinado al cálculo de las corrientes y de sus derivadas. Como se dijo en el apartado 6.2.3 este bloque se divide en dos secciones. La primera de ellas está destinada al cálculo de las corrientes de diodos y sus derivadas y, por tanto, será ahí donde introduzcamos las ecuaciones que describen las conductancias g_{gs} y g_{gd} (ecs. (3.32) y (3.33)). La otra sección en la que se divide el bloque mencionado es la que da cuenta de la fuente de corriente que modela la corriente de drenador (i_d). Es en esta sección donde introducimos el modelado de las conductancias g_m y g_{ds} tal y como se expresa en las ecuaciones (3.35) y (3.36).

6.2.5 Análisis en ac

Los análisis en ac se realizan con la función HEMTACL.C la cual es una variación de la función HEMTLOAD.C. Esta función no evalúa el dispositivo

basándose en la información actual en el vector RHS. Por contra, carga la matriz y el RHS basándose en la información obtenida en el último análisis en dc, el cual ha de ser seguido por una llamada a la función HEMTLOAD.C con el bit MODEINITSMSIG puesto a "1". Como consecuencia de esta sucesión de operaciones, se deberá haber salvado suficiente información en las estructuras de datos pertenecientes a las instancias para permitir que durante el análisis en ac sólo tengamos que cargar la matriz con las admitancias adecuadas evaluadas a la frecuencia de operación.

En los análisis en ac las admitancias de los nodos son números complejos. Estos números dependen de la frecuencia de la siguiente forma:

$$\vec{Y} = G + jB = G + j\Omega C + \frac{1}{j\Omega L} \tag{6.2}$$

Donde G es la conductancia y B la susceptancia la cual depende de $\Omega = 2\pi f$ que es la frecuencia angular medida en radianes por segundo siendo f la frecuencia en Hertzios. En el dominio de la frecuencia, las tensiones y las corrientes del circuito son también fasores:

$$\begin{aligned} \vec{V} &= V_R + jV_I = |\vec{V}|e^{j\phi} \\ |\vec{V}| &= \sqrt{V_R{}^2 + V_I{}^2} \\ \phi &= arctan\left(\frac{V_I}{V_R}\right) \end{aligned} \tag{6.3}$$

Es sabido que los fasores están compuestos de una parte real, V_R, y una imaginaria, V_I, y se pueden expresar también como magnitud, $|\vec{V}|$, y fase, ϕ. El factor de periodicidad, sen Ωt, se asume implícitamente para todas las variables en los análisis en ac.

Por tanto, a la hora de calcular las admitancias en los análisis en ac, debemos separar el cálculo de las conductancias y las susceptancias e introducirlas en las posiciones adecuadas. La matriz de admitancias consistirá por tanto en un conjunto de punteros para las conductancias y otro para las susceptancias. Ambos tipos de punteros poseen los mismos nombres con la única diferencia de que los asociados a las susceptancias ocupan una posición en

memoria contigua. Por ejemplo, para el elemento matricial que da cuenta de los nodos A y B, tendremos que el puntero

```
*(here->HEMTABPtr)
```

está asociado a conductancia que une los nodos A y B, y el puntero

```
*(here->HEMTABPtr + 1)
```

está asociado a la admitancia.

Las conductancias se calculan y se introducen dentro de la matriz de admitancias de igual forma a como lo hacíamos en los análisis en dc o en régimen transitorio. Sin embargo, las susceptancias deben ser calculadas teniendo en cuenta la frecuencia de operación Ω. Esta frecuencia viene dada en la variable circuital *ckt->CKTomega*. Una vez calculadas las susceptancias se introducirán en la matriz de admitancias dentro de las posiciones adecuadas como vimos antes.

Un caso especial de cálculo de conductancia y susceptancia lo constituye la transconductancia modificada asociada a la fuente de corriente de la figura 3.7.b, $\hat{g}_m$. Según se vio en el apartado 3.2.1, la expresión para tal transconductancia modificada es la siguiente:

$$\hat{g}_m = g_m \cdot e^{-j\Omega\tau} = g_m \cdot (\cos\Omega\tau - j\,\mathrm{sen}\,\Omega\tau) \tag{6.4}$$

donde g_m es la transconductancia y τ es un exceso de fase similar al parámetro *ptf* de los transistores bipolares. La parte real y la parte imaginaria nos darán la conductancia y la susceptancia respectivamente:

$$\begin{aligned} G_m &= g_m \cdot \cos\Omega\tau \\ B_m &= -g_m \cdot j\,\mathrm{sen}\,\Omega\tau \end{aligned} \tag{6.5}$$

Capítulo 7

7. Validación de Resultados y Conclusiones

En el presente trabajo se ha estudiado, por un lado, el simulador eléctrico SPICE y, por otro, los transistores HEMTs así como sus modelos eléctricos más importantes. Comenzamos en el primer capítulo haciendo una introducción a SPICE y a sus dos versiones más importantes SPICE2 y SPICE3 desarrolladas por la Universidad de California en Berkeley. En este capítulo se vieron tanto los dispositivos que son capaces de simular como los distintos análisis que pueden llevar a cabo. Se hizo hincapié en el hecho de que SPICE3 es un programa interactivo de forma que el usuario, cuando lo invoca, pasa a entrar al *shell* del programa dentro del cual puede ejecutar los comandos que desee.

En el segundo capítulo se estudiaron las heteroestructuras y los transistores de efecto de campo de heteroestructura (HFETs). El estudio de este tipo de transistores se organizó señalando cual es la estructura básica así como el principio de funcionamiento de los mismos. También se presentaron algunas consideraciones tecnológicas en cuanto a la fabricación de dichos transistores.

Siguiendo con el estudio de los transistores HFET, en el tercer capítulo revisamos el modelo de Angelov [ANGEL92] el cual, a pesar de ser un modelo semiempírico, es hoy por hoy el modelo SPICE más aceptado y sobre el que se basan la mayoría de modelos que hay en la actualidad para los HFET. Un ejemplo de esto es el modelo del FhGIAF que es el que nosotros hemos implementado, y que, por tanto, también fue objeto de estudio en este capítulo.

Una vez estudiados los transistores HFET tanto su estructura física como su modelo eléctrico, el siguiente paso consiste en analizar el simulador eléctrico SPICE que va a dar cabida a dicho modelo. De esta forma, comenzamos en el capítulo 4 estudiando los algoritmos más importantes en los que se basa SPICE así como los principales problemas que pueden hacer que una simulación no llegue a buen término.

En el capítulo 5 se presentó la estructura global de SPICE3 así como de los diferentes paquetes que lo componen. SPICE3 fue diseñado utilizando una estructura modular de forma que cada paquete de rutinas es relativamente independiente de cualquier otro. Esto permite que los paquetes puedan ser mantenidos, mejorados o reemplazados de forma independiente. El objetivo fundamental de este capítulo es entender lo suficiente como para saber qué paquete hay que observar cuando se necesita una funcionalidad determinada o cuando se encuentra un error asociado a una operación particular.

En el capítulo 6 se hace una descripción detallada de cómo se introduce un modelo determinado dentro de SPICE3. Esta tarea no puede ser llevada a cabo sin conocer, por un lado, al dispositivo y su modelo y, por el otro, el simulador y su forma de implementar los diferentes dispositivos. Por tanto, los estudios realizados en los capítulos anteriores se convierten en imprescindibles. El modelo implementado es el modelo del HFET del FhGIAF que se estudió en el capítulo 3.

Por otro lado, es importante notar que a pesar de que la forma en como se implementa un determinado modelo dentro de SPICE depende directamente del modelo y del simulador, los procedimientos explicados en este capítulo pueden ser aplicados a la implementación de otros dispositivos. Por tanto, este trabajo podría ser de gran interés en investigaciones futuras sobre el mismo transistor o sobre otros dispositivos.

Para finalizar, en este capítulo comenzaremos llevando a cabo la validación de los resultados obtenidos comparándolos con los resultados que da IAFSPICE, para acabar presentamos una serie de conclusiones acerca de los posibles trabajos futuros que se pueden llevar a cabo en base a este proyecto.

7.1 Validación de los Resultados Obtenidos

La validación de los resultados obtenidos se realiza comparando las simulaciones hechas con el modelo que implementamos en SPICE3 y las hechas con IAFSPICE. Obviamente, los resultados van a ser muy parecidos ya que ambos simuladores implementan las mismas ecuaciones. Sin embargo, las principales diferencias vendrán motivadas por el hecho de que ambos simuladores utilizan diferente precisión. En efecto, SPICE3 utiliza datos de 6 dígitos decimales mientras que IAFSPICE utiliza datos con sólo 3 dígitos decimales. Aún así, las diferencias son mínimas entre los resultados de ambos simuladores y, al no disponer de datos reales con los que comparar, no podemos decir cual de ambos es el que más se ajusta al funcionamiento real del dispositivo.

El circuito utilizado para la validación de los resultados en dc es el que se muestra en la figura 7.1. Se trata del circuito típico utilizado para realizar la medida de las características de salida y entrada en dc de un transistor. De esta forma, haciendo un barrido de la tensión V_{DS} para diferentes valores de la tensión V_{GS} tenemos la característica I_{DS} *vs.* V_{DS}. De la misma forma, haciendo un barrido de V_{GS} para una V_{DS} fija, obtenemos la característica I_{DS} *vs.* V_{GS}.

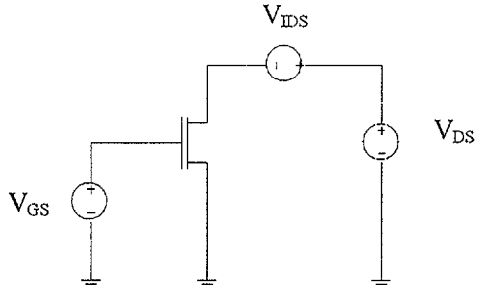


fig. 7.1 Circuito para evaluar las curvas características en dc de un transistor HFET.

En las figuras 7.2.a y 7.2.b se pueden observar los resultados obtenidos para un transistor de enriquecimiento de 0.5 μm. Las figuras 7.3.a y 7.3.b representan las mismas medidas para un transistor de deplexión. En línea continua se muestran los resultados de IAFSPICE y con círculo la de nuestra implementación. Se observa que tanto para un caso como para el otro la coincidencia de los resultados es total. En el apéndice B se presenta el fichero de entrada de nuestro simulador para la obtención de la figura 7.2.a.

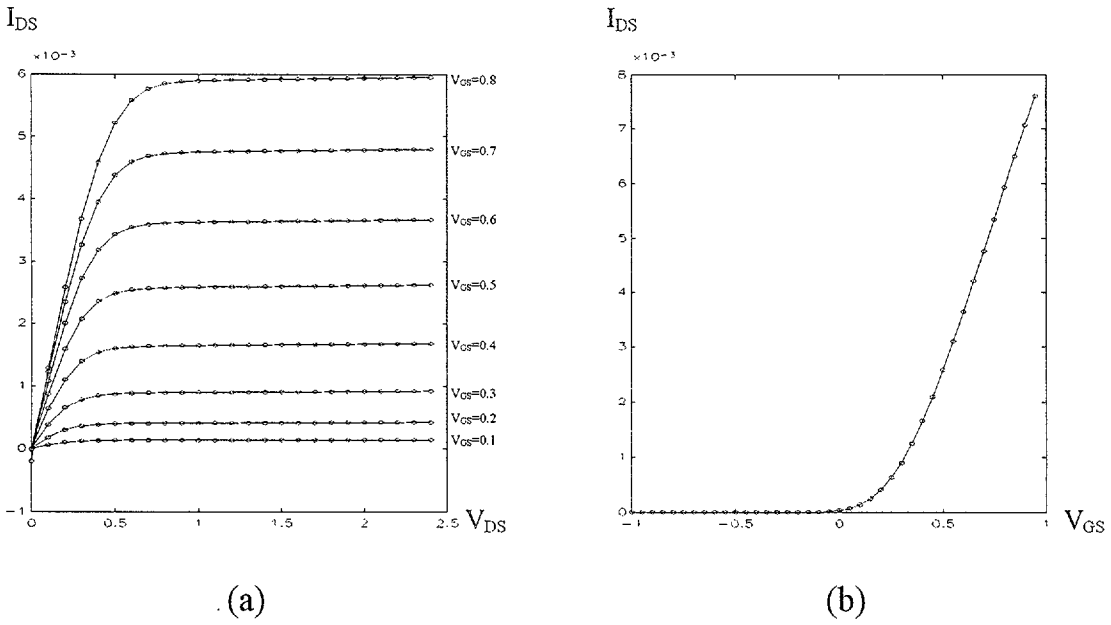


fig. 7.2 Características (a) I_{DS} *vs.* V_{DS} y (b) I_{DS} *vs.* V_{GS} de un HFET de enriquecimiento de 0.5 μm.

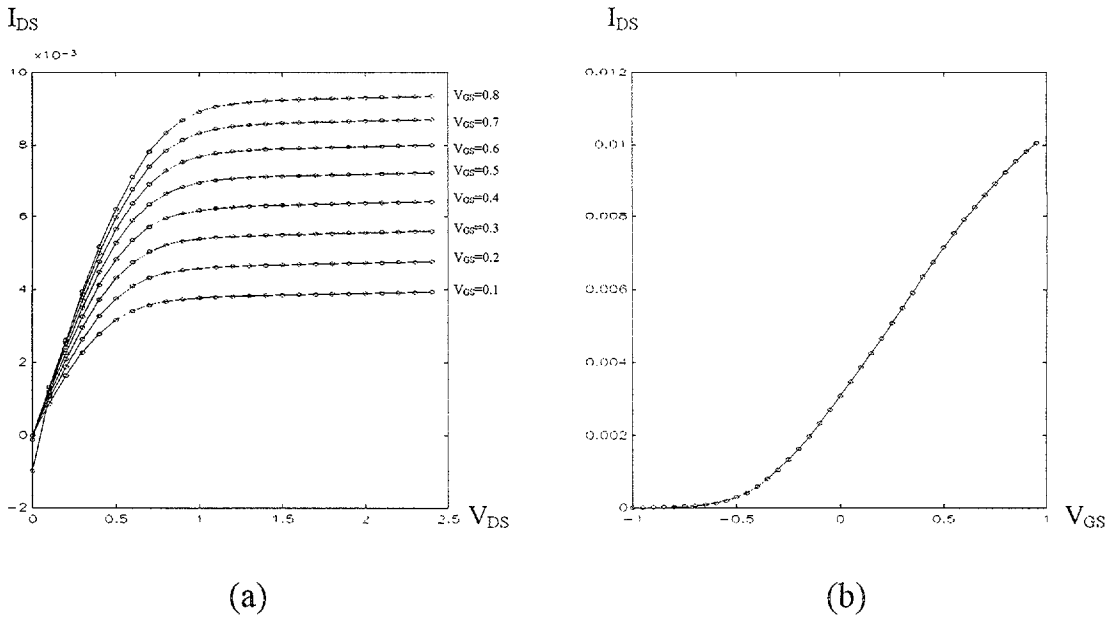


fig. 7.3 Características (a) I_{DS} *vs.* V_{DS} y (b) I_{DS} *vs.* V_{GS} de un HFET de enriquecimiento de 0.5 μm.

En cuanto al análisis en régimen transitorio, el circuito utilizado es el de la figura 7.4. Esta figura consiste en tres inversores DCFL conectados uno detrás del otro a los que se le aplica un pulso a la entrada. Los inversores DCFL se construyen poniendo un transistor de carga activa de tipo deplexión de anchura aproximadamente la mitad del transistor activo que es de enriquecimiento.

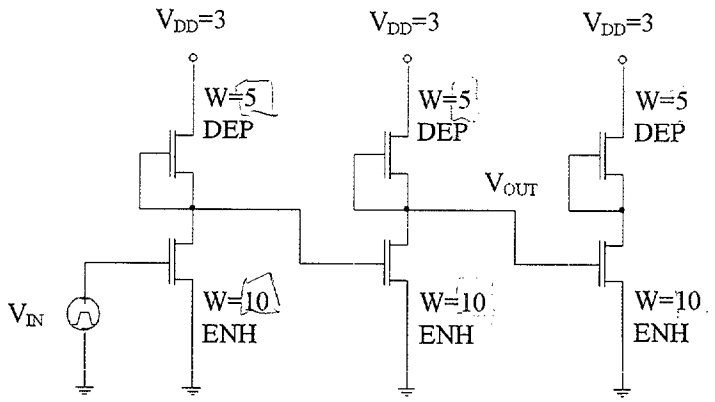


fig. 7.4 Cadena de tres inversores DCFL conectados en serie para comprobación del análisis en régimen transitorio.

Los resultados obtenidos se muestran en la figura 7.5. En este punto es importante destacar que SPICE3 y SPICE2 tienen diferente forma de presentar los resultados del análisis en régimen transitorio. SPICE2 analiza el circuito y da los resultados en los instantes de tiempo marcados por las tarjetas *.print* o *.plot*. Sin embargo, SPICE3 realiza el análisis en instantes de tiempo tales que el error cometido esté por debajo de la tolerancia admitida, y guarda todos los resultados obtenidos. De esta forma SPICE3 es capaz de detectar los cambios más mínimos de la señal de salida aun cuando estos sean muy pequeños. Para que la salida de SPICE3 contenga el mismo número de datos que la de SPICE2 ha de utilizarse el comando *linearize*, el cual no hace otra cosa que interpolar los vectores de salida a una escala de tiempo lineal. Así, sólo se mantendrán aquellos picos de la señal de salida que son significativos con respecto al intervalo de muestreo que ha especificado el usuario en las tarjetas *.print* o *.plot*. El resultado es, por tanto, una señal similar a la que da SPICE2 y que se parece más a la señal real de salida del circuito.

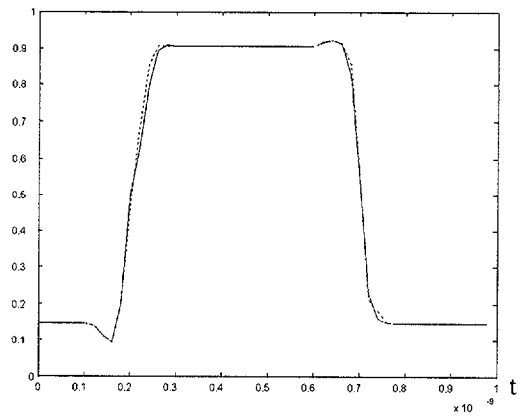


fig. 7.5 Resultados del análisis en régimen transitorio para tres inversores. V_{OUT}(IAFSPICE): línea continua; V_{OUT}(SPICE3): línea de puntos.

El mismo análisis lo hemos hecho sobre una cadena de 16 inversores para comprobar el funcionamiento del programa con un mayor número de transistores. El fichero de entrada para esta simulación se muestra en el apéndice B. Las curvas obtenidas a la salida del decimocuarto inversor tanto en IAFSPICE como en SPICE3 se muestran en la figura 7.6. Se puede observar como se ha retrasado la salida respecto a la señal que salía del segundo inversor (ver fig. 7.5) debido al efecto de las capacidades internas del dispositivo.

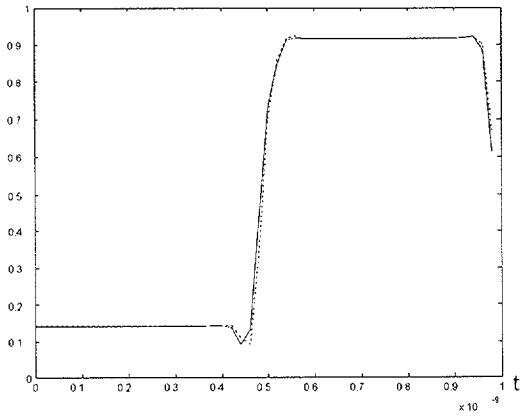


fig. 7.6 Resultados del análisis en régimen transitorio para 16 inversores. V_{OUT}(IAFSPICE): línea continua; V_{OUT}(SPICE3): línea de puntos.

Se ha realizado un análisis más para comprobar el funcionamiento del simulador para circuitos aún mayores. Se trata de un sumador tipo *full-adder* cuyo esquema de bloques es el que se presenta en la figura 7.7. En el apéndice B se presenta el fichero de entrada de SPICE3 de dicho circuito. Está realizado con puertas DCFL y consta de 67 transistores HFET. Las señales aplicadas a las entradas se hacen pasar primero por dos inversores conectados en serie siendo sus formas de onda a la salida de los inversores las que se muestran en la figura 7.8. Las formas de onda a las salidas Sout y Cout se muestran en las figuras 7.9.a y 7.9.b respectivamente. En las figuras se observa cómo de nuevo se superponen las predicciones que arroja cada versión del simulador.

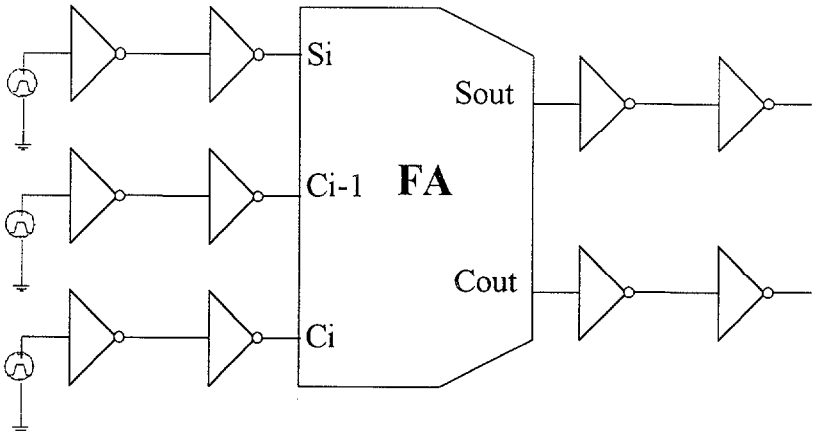


fig. 7.7 Diagrama de bloques de un sumador FA.

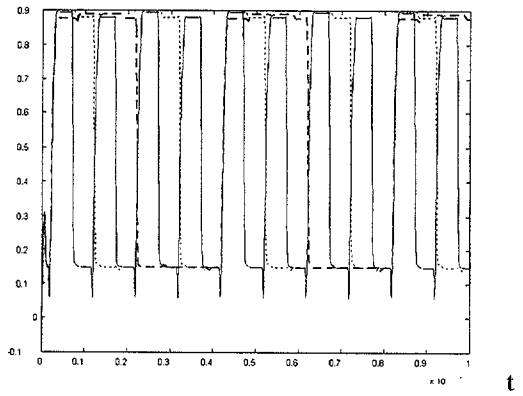


fig. 7.8 Señales de entrada Si (línea continua), Ci (línea de puntos) y Ci-1 (línea discontinua).

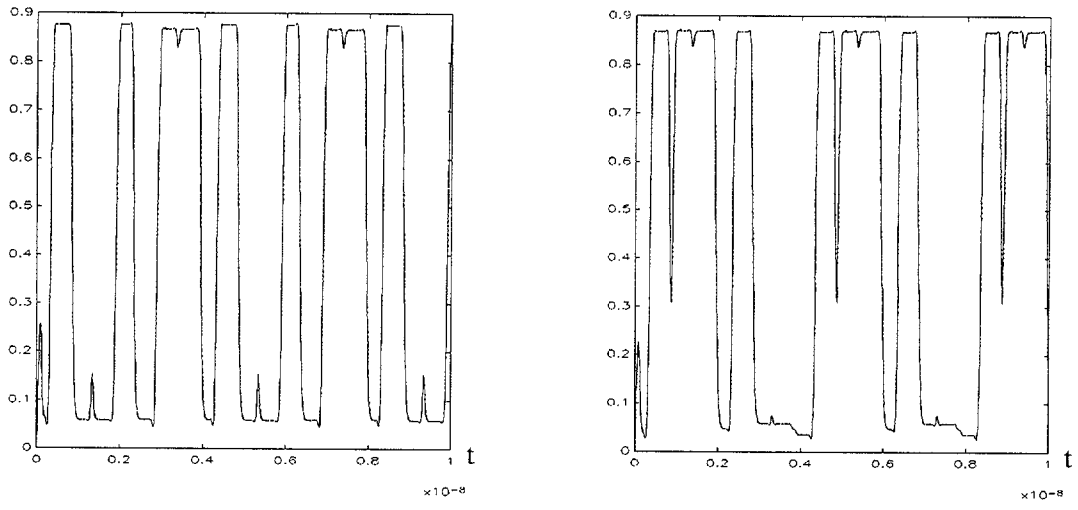


fig. 7.9 Señales de salida (a) Sout y (b) Cout. IAFSPICE: línea continua, SPICE3: línea de puntos.

Por último, el circuito utilizado para comprobar el análisis en ac es el que se muestra en la figura 7.10. El fichero de entrada correspondiente a este circuito se presenta el apéndice B. Consiste básicamente en un transistor polarizado en el que se realiza un análisis en ac. La señal de salida es la corriente que circula por la fuente de tensión V_{MEAS}, la cual va a ser un número complejo por la propia definición de los análisis en ac. Por tanto, las posibles medidas que podemos obtener son la parte real y la parte imaginaria de la corriente (fig. 7.11), su magnitud y su fase (fig. 7.12) y su magnitud en decibelios (fig. 7.13). Se observa que hay una ligera diferencia entre los resultados de uno y otro simulador motivada fundamentalmente por el problema de la diferencia de precisión comentado anteriormente.

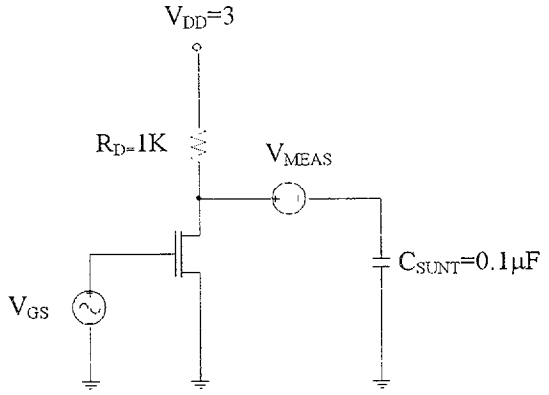


fig. 7.10 Circuito para comprobar el análisis en ac.

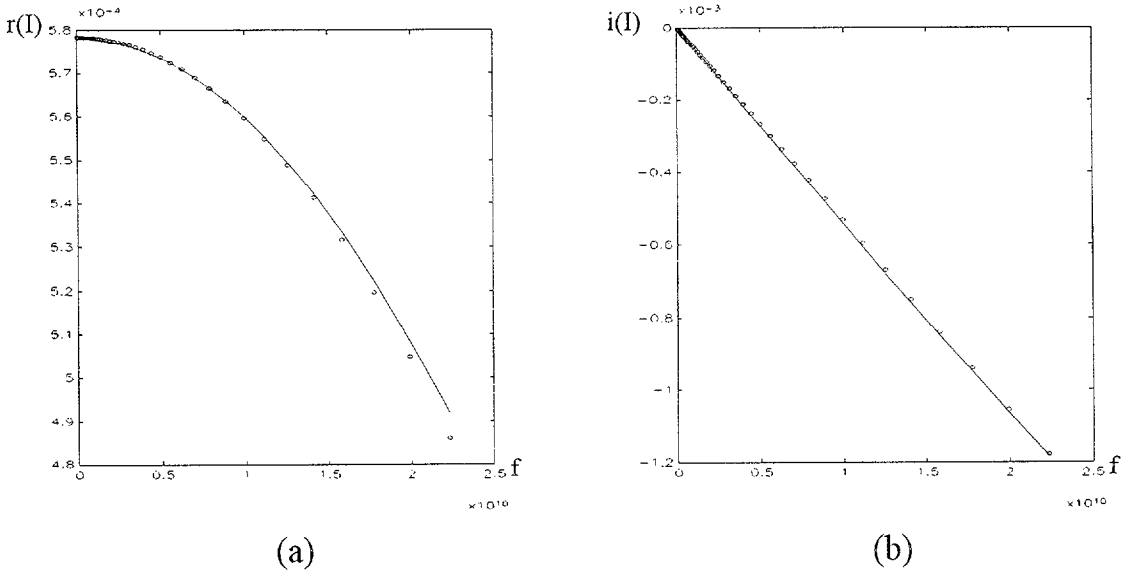
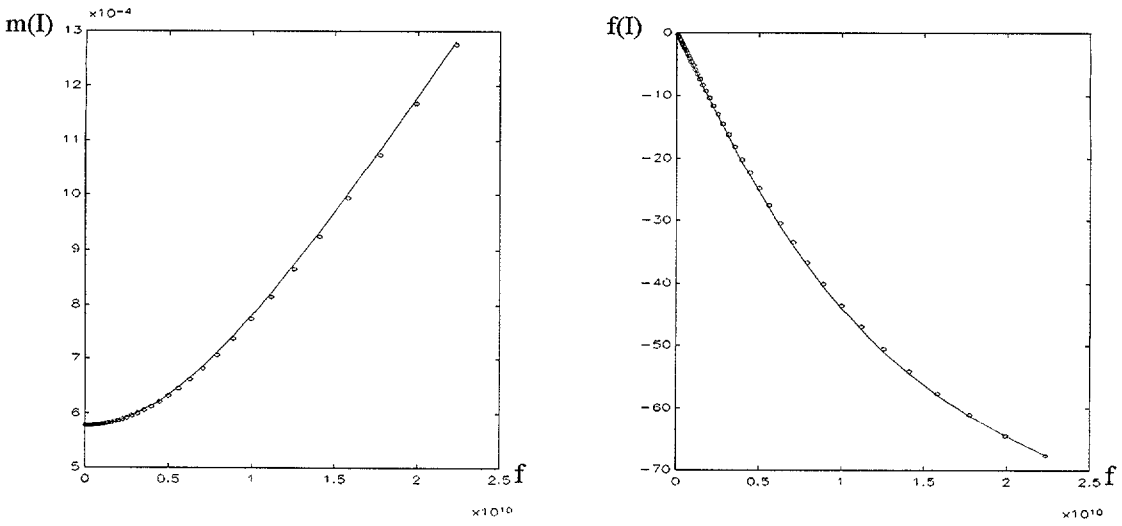


fig. 7.11 Parte real y parte imaginaria de la corriente $I(V_{MEAS})$ del circuito de la figura 7.10.



(a) (b)

fig. 7.12 Magnitud y fase de la corriente $I(V_{MEAS})$ del circuito de la figura 7.10.

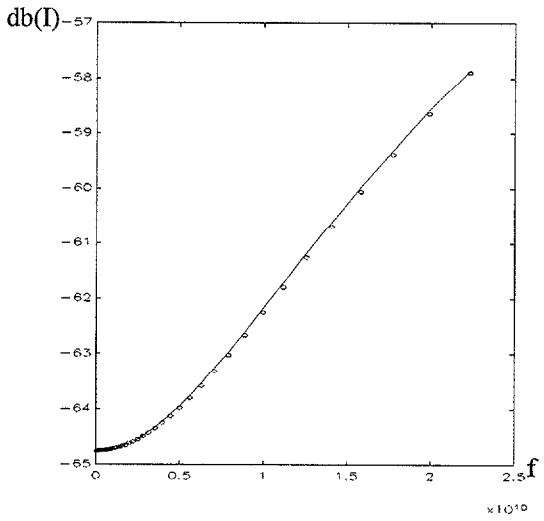


fig. 7.13 Magnitud de la corriente $I(V_{MEAS})$ del circuito de la figura 7.10.

7.2 Conclusiones finales

Además de lo ya reseñado y probado respecto a la validez del trabajo realizado queremos indicar que con este trabajo el HFET no queda completamente modelado. Falta incorporar los análisis de temperatura, polos y ceros, ruido, distorsión, sensibilidad, etc.

En cuanto al análisis de temperatura, decir que actualmente no existe un modelo fiable de su influencia y, por tanto, no se puede implementar. Para llevar a cabo esta tarea, lo primero que deberíamos hacer es fabricar un transistor HFET (en fábrica o en su defecto con simuladores físicos tipo T-SUPREM4) y obtener sus parámetros de modelo. Posiblemente tendríamos que modificar el modelo de IAFSPICE puesto que dicho modelo, al ser semiempírico, depende de la tecnología utilizada y, por tanto, puede no servirnos para nuestro transistor. Sin embargo, el modelo resultante sería siempre muy parecido al de Angelov. Una vez hecho esto se estudiaría el efecto de la temperatura sobre el comportamiento eléctrico de nuestro HFET haciéndose un modelo e implementándolo dentro de nuestro simulador. Actualmente estas tareas son objeto de una tesis dentro del Departamento de Electrónica Telemática y Automática de la Universidad de las Palmas de Gran Canaria.

Por otro lado, se ha intentado implementar el análisis de polos y ceros y formalmente no es muy difícil. Sin embargo, este tipo de análisis da problemas en la versión SPICE3e2, que es la que hemos utilizado, ejecutada sobre estaciones SUN, que es nuestro caso. Por tanto, un posible trabajo futuro podría ser el introducir el modelo de los transistores HFET en las últimas versiones de SPICE incluyendo el análisis de polos y ceros.

Para el resto de análisis la situación es muy parecida al caso de la temperatura ya que en la actualidad no tenemos noticia de que existan modelos adecuados y, por tanto, ello puede ser objeto de futuros trabajos.

APÉNDICES

Apéndice A

Código de la Implementación del HFET en SPICE3

A continuación se listan los ficheros de código relacionados con la implementación del HFET dentro de SPICE3. En primer lugar presentamos el contenido de los ficheros fuente que se encuentran dentro del directorio "src/lib/dev/hemt":

- HEMT.C
- HEMTACL.C
- HEMTASK.C
- HEMTDEL.C
- HEMTDEST.C
- HEMTGETI.C
- HEMTLOAD.C
- HEMTMASK.C
- HEMTMDEL.C
- HEMTMPAR.C
- HEMTPARA.C
- HEMTSETU.C
- HEMTTEMP.C
- HEMTTRUN.C

Igualmente se presentan los listados de los ficheros *include* relacionados con el dispositivo. Estos ficheros se pueden encontrar dentro del subdirectorio "src/lib/include":

- HEMTDEFS.H
- HEMTEXT.H
- HEMTITF.H

Por último, también se presentan los ficheros del paquete de entrada que hacen que el programa detecte que hay un dispositivo nuevo y sea capaz de cargarlo. Estos ficheros se encuentran en el directorio "src/lib/inp"

- INPDOMOD.C
- INPPAS2.C
- INP2Z.C

En los diferentes listados, además de los comentarios hechos sobre el programa, se he marcado en diferente color los cambios hechos en el código del MESFET para personalizarlo y convertirlo en el modelo del HEMT.

A.1 HEMT.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "ifsim.h"
#include "devdefs.h"
#include "hemtdefs.h"
#include "suffix.h"


IFparm HEMTpTable[] = { /* parametros */
 OP("off",  HEMT_OFF,    IF_FLAG   ,"Dispositivo inicialmente off"),
 IOP("w", HEMT_W,   IF_REAL   ,"factor w"),
 IOP("icvds",  HEMT_IC_VDS, IF_REAL   ,"tension D-S initial"),
 IOP("icvgs",  HEMT_IC_VGS, IF_REAL   ,"tension G-S initial"),
 OP("dnode", HEMT_DRAINNODE, IF_INTEGER,"Numero del nodo drenador"),
 OP("gnode", HEMT_GATENODE, IF_INTEGER,"Numero del nodo puerta"),
 OP("snode", HEMT_SOURCENODE, IF_INTEGER,"Numero del nodo surtidor"),
 OP("dprimenode",HEMT_DRAINPRIMENODE,IF_INTEGER,
      "Numero del nodo drenador interno"),
 OP("gprimenode",HEMT_GATEPRIMENODE,IF_INTEGER,
      "Numero del nodo puerta interno"),
 OP("sprimenode",HEMT_SOURCEPRIMENODE,IF_INTEGER,
      "Numero del nodo surtidor interno"),
 OP("vgs", HEMT_VGS, IF_REAL,"tension Puerta-Surtidor"),
 OP("vgd", HEMT_VGD, IF_REAL,"tension Puerta-Drenador"),
 OP("cg", HEMT_CG, IF_REAL,"capacidad de Puerta"),
 OP("cd", HEMT_CD, IF_REAL,"capacidad de Drenador"),
 OP("cgd", HEMT_CGD, IF_REAL,"capacidad Puerta Drenador"),
 OP("gm", HEMT_GM, IF_REAL,"transconductancia"),
 OP("gds", HEMT_GDS, IF_REAL,"conductancia Drenador-Surtidor"),
 OP("ggs", HEMT_GGS, IF_REAL,"conductancia Puerta-Surtidor"),
 OP("ggd", HEMT_GGD, IF_REAL,"conductancia Puerta-Drenador"),
 OP("qgs", HEMT_QGS, IF_REAL,"carga Puerta-Surtidor"),
 OP("cqgs", HEMT_CQGS, IF_REAL,"corriente debida a la carga Puerta-Surtidor"),
 OP("qgd", HEMT_QGD, IF_REAL,"carga Puerta-Drenador"),
 OP("cqgd", HEMT_CQGD, IF_REAL,"corriente debida a la carga Puerta-Drenador"),
 OP("qds", HEMT_QDS, IF_REAL,"carga Drenador-Surtidor"),
 OP("cqds", HEMT_CQDS, IF_REAL,"corriente debida a la carga Drenador-Surtidor"),
 OP("cs",   HEMT_CS,     IF_REAL   ,"corriente de Surtidor"),
 OP("p",    HEMT_POWER,  IF_REAL   ,"Potencia disipada por el hemt")

};

IFparm HEMTmPTable[] = { /* model parameters */
 IOP( "vt0",     HEMT_MOD_VTO,     IF_REAL,"Tension umbral"),
 IOP( "vto",     HEMT_MOD_VTO,     IF_REAL,"Tension umbral"),
 IOP( "alpha",   HEMT_MOD_ALPHA,   IF_REAL,"Parametro de tension de saturacion"),
 IOP( "beta",    HEMT_MOD_BETA,    IF_REAL,"Parametro de ajuste beta"),
 IOP( "lambda", HEMT_MOD_LAMBDA, IF_REAL,"Parametro de modulacion del canal"),
 IOP( "b",       HEMT_MOD_B,       IF_REAL,"Doping tail extending parameter"),
 IOP( "rd",      HEMT_MOD_RD,      IF_REAL,"Resistencia de Drenador"),
 IOP( "rs",      HEMT_MOD_RS,      IF_REAL,"Resistencia de Surtidor"),
 IOP( "cgs",     HEMT_MOD_CGS,     IF_REAL,"G-S junction capacitance"),
 IOP( "cgd",     HEMT_MOD_CGD,     IF_REAL,"G-D junction capacitance"),
 IOP( "pb",      HEMT_MOD_PB,      IF_REAL,"Gate junction potential"),
```

```
 IOP( "is",      HEMT_MOD_IS,      IF_REAL,
      "Corriente de saturacion de las uniones"),
 IOP( "fc",      HEMT_MOD_FC,      IF_REAL,"Parametro de ajuste cf"),
 IOP( "np",      HEMT_MOD_NP,      IF_REAL,
      "Coeficiente de emision o de no idealidad"),
 IOP( "afact",   HEMT_MOD_AFACT,   IF_REAL,"Parametro de ajuste A"),
 IOP( "gexp",    HEMT_MOD_GEXP,    IF_REAL,"Parametro de ajuste B"),
 IOP( "cdvc",    HEMT_MOD_CDVC,    IF_REAL,
      "Corriente de Drenador para transconductancia maxima"),
 IOP( "vc",      HEMT_MOD_VC,      IF_REAL,
      "Tension de Puerta para transconductancia maxima"),
 IOP( "gamma",   HEMT_MOD_GAMMA,   IF_REAL,"Parametro de ajuste gamma"),
 IOP( "cdvsb",   HEMT_MOD_CDVSB,   IF_REAL,
      "Corriente de Drenador para transconductancia minima"),
 IOP( "delta",   HEMT_MOD_DELTA,   IF_REAL,"Parametro de ajuste delta"),
 IOP( "vsb",     HEMT_MOD_VSB,     IF_REAL,
      "Tension de Puerta para transconductancia minima"),
 IOP( "dxl",     HEMT_MOD_DXL,     IF_REAL,
      "Variacion del parametro de modulacion del canal"),
 IOP( "vat",     HEMT_MOD_VAT,     IF_REAL,"Tension de pinch-off"),
 IOP( "vbt",     HEMT_MOD_VBT,     IF_REAL,"Tension de conduccion"),
 IOP( "cgs0",    HEMT_MOD_CGS0,    IF_REAL,"Capacidad para alimentacion nula"),
 IOP( "cgs1",    HEMT_MOD_CGS1,    IF_REAL,
      "Factor de capacidad para alimentacion nula"),
 IOP( "vst",     HEMT_MOD_VST,     IF_REAL,"Parametro de ajuste vst"),
 IOP( "vs",      HEMT_MOD_VS,      IF_REAL,"Parametro de ajuste vs"),
 IOP( "mfact",   HEMT_MOD_MFACT,   IF_REAL,"Parametro de ajuste m"),
 IOP( "rg",      HEMT_MOD_RG,      IF_REAL,"Resistencia de Puerta"),
 IOP( "rgs",     HEMT_MOD_RGS,     IF_REAL,"Resistencia Puerta-Surtidor"),
 IOP( "rgd",     HEMT_MOD_RGD,     IF_REAL,"Resistencia Puerta-Drenador"),
 IOP( "qc1",     HEMT_MOD_QC1,     IF_REAL,"Carga residual del dispositivo"),
 IOP( "cds",     HEMT_MOD_CDS,     IF_REAL,"Capacidad Drenador-Surtidor"),
 IOP( "td",      HEMT_MOD_TD,      IF_REAL,"Exceso de fase"),
 OP( "nmf",    HEMT_MOD_NMF,    IF_FLAG,"modelo del HEMT tipo N"),
 OP( "pmf",    HEMT_MOD_PMF,    IF_FLAG,"modelo del HEMT tipo P"),
 OP( "gd",   HEMT_MOD_DRAINCONDUCT,   IF_REAL,"Conductancia de Drenador"),
 OP( "gs",   HEMT_MOD_SOURCECONDUCT,  IF_REAL,"conductancia de Surtidor"),
 OP( "depl_cap",    HEMT_MOD_DEPLETIONCAP,   IF_REAL,"Capacidad de Deplexion"),
 OP( "vcrit",   HEMT_MOD_VCRIT,  IF_REAL,"Tension critica"),
 P("kf",         HEMT_MOD_KF,           IF_REAL,"Flicker noise coefficient"),
 P("af",         HEMT_MOD_AF,           IF_REAL,"Flicker noise exponent"),
};

char *HEMTnames[] = {
    "Drain",
    "Gate",
    "Source"
};

int   HEMTnSize = NUMELEMS(HEMTnames);
int   HEMTpTSize = NUMELEMS(HEMTpTable);
int   HEMTmPTSize = NUMELEMS(HEMTmPTable);
int   HEMTiSize = sizeof(HEMTinstance);
int   HEMTmSize = sizeof(HEMTmodel);
```

A.2 HEMTACL.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "cktdefs.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


int
HEMTacLoad(inModel,ckt)
    GENmodel *inModel;
    register CKTcircuit *ckt;
{
    register HEMTmodel *model = (HEMTmodel*)inModel;
    register HEMTinstance *here;
    double gdpr;
    double gspr;
    double ggpr;
    double gm;
    double xm;
    double gds;
    double xds;
    double ggs;
    double xgs;
    double ggd;
    double xgd;

    /*  bucle por todos los modelos del dispositivo */
    for( ; model != NULL; model = model->HEMTnextModel ) {

        /* bucle por todas las istancias del modelo */
        for( here = model->HEMTinstances; here != NULL;
                here = here->HEMTnextInstance) {

            gdpr= model->HEMTdrainConduct * here->HEMTw;
            gspr= model->HEMTsourceConduct * here->HEMTw;
            ggpr = 1 / (model->HEMTrg * here->HEMTw);
            gm= *(ckt->CKTstate0 + here->HEMTgm);
            xm= 0;
            if(model->HEMTtd != 0){
                  gm= gm * cos(model->HEMTtd * ckt->CKTomega);
                  xm= -(gm * sin(model->HEMTtd *ckt->CKTomega));
            }
            gds= *(ckt->CKTstate0 + here->HEMTgds);
            xds= *(ckt->CKTstate0 + here->HEMTqds) * ckt->CKTomega;
            ggs= *(ckt->CKTstate0 + here->HEMTggs) ;
            xgs= *(ckt->CKTstate0 + here->HEMTqgs) * ckt->CKTomega;
            ggd= *(ckt->CKTstate0 + here->HEMTggd) ;
            xgd= *(ckt->CKTstate0 + here->HEMTqgd) * ckt->CKTomega;

            *(here->HEMTdrainDrainPtr) += gdpr;
            *(here->HEMTgatePrimeGatePrimePtr) += ggd+ggs+ggpr;
            *(here->HEMTgatePrimeGatePrimePtr +1) += xgd+xgs;
```

```
            *(here->HEMTsourceSourcePtr) += gspr;
            *(here->HEMTdrainPrimeDrainPrimePtr ) += gdpr+gds+ggd;
            *(here->HEMTdrainPrimeDrainPrimePtr +1) += xgd+xds;
            *(here->HEMTsourcePrimeSourcePrimePtr ) += gspr+gds+gm+ggs;
            *(here->HEMTsourcePrimeSourcePrimePtr +1) += xgs+xds+xm;
            *(here->HEMTdrainDrainPrimePtr) -= gdpr;
            *(here->HEMTgatePrimeDrainPrimePtr ) -= ggd;
            *(here->HEMTgatePrimeDrainPrimePtr +1) -= xgd;
            *(here->HEMTgatePrimeSourcePrimePtr ) -= ggs;
            *(here->HEMTgatePrimeSourcePrimePtr +1) -= xgs;
            *(here->HEMTsourceSourcePrimePtr) -= gspr;
            *(here->HEMTdrainPrimeDrainPtr) -= gdpr;
            *(here->HEMTdrainPrimeGatePrimePtr) += (-ggd+gm);
            *(here->HEMTdrainPrimeGatePrimePtr +1) += (-xgd+xm);
            *(here->HEMTdrainPrimeSourcePrimePtr) += (-gds-gm);
            *(here->HEMTdrainPrimeSourcePrimePtr +1) += (-xds-xm);
            *(here->HEMTsourcePrimeGatePrimePtr ) += (-ggs-gm);
            *(here->HEMTsourcePrimeGatePrimePtr +1) += (-xgs-xm);
            *(here->HEMTsourcePrimeSourcePtr) -= gspr;
            *(here->HEMTsourcePrimeDrainPrimePtr) -= gds;
            *(here->HEMTsourcePrimeDrainPrimePtr +1) -= xds;
            *(here->HEMTgateGatePtr) += ggpr;
            *(here->HEMTgateGatePrimePtr) += (-ggpr);
            *(here->HEMTgatePrimeGatePtr) += (-ggpr);

        }
    }
    return(OK);
}
```

A.3 HEMTASK.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "cktdefs.h"
#include "devdefs.h"
#include "ifsim.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "util.h"
#include "suffix.h"


/* ARGSUSED */
int
HEMTask(ckt,inst,which,value,select)
    CKTcircuit *ckt;
    GENinstance *inst;
    int which;
    IFvalue *value;
    IFvalue *select;
{
    HEMTinstance *here = (HEMTinstance*)inst;
    static char *msg = "Current and power not available in ac analysis";
    switch(which) {
        case HEMT_W:
            value->rValue = here->HEMTw;
            return (OK);
        case HEMT_IC_VDS:
            value->rValue = here->HEMTicVDS;
            return (OK);
        case HEMT_IC_VGS:
            value->rValue = here->HEMTicVGS;
            return (OK);
        case HEMT_OFF:
            value->iValue = here->HEMToff;
            return (OK);
        case HEMT_DRAINNODE:
            value->iValue = here->HEMTdrainNode;
            return (OK);
        case HEMT_GATENODE:
            value->iValue = here->HEMTgateNode;
            return (OK);
        case HEMT_SOURCENODE:
            value->iValue = here->HEMTsourceNode;
            return (OK);
        case HEMT_DRAINPRIMENODE:
            value->iValue = here->HEMTdrainPrimeNode;
            return (OK);
        case HEMT_VGS:
            value->rValue = *(ckt->CKTstate0 + here->HEMTvgs);
            return (OK);
        case HEMT_VGD:
            value->rValue = *(ckt->CKTstate0 + here->HEMTvgd);
            return (OK);
        case HEMT_CG:
```

```
        value->rValue = *(ckt->CKTstate0 + here->HEMTcg);
        return (OK);
    case HEMT_CD:
        value->rValue = *(ckt->CKTstate0 + here->HEMTcd);
        return (OK);
    case HEMT_CGD:
        value->rValue = *(ckt->CKTstate0 + here->HEMTcgd);
        return (OK);
    case HEMT_GM:
        value->rValue = *(ckt->CKTstate0 + here->HEMTgm);
        return (OK);
    case HEMT_GDS:
        value->rValue = *(ckt->CKTstate0 + here->HEMTgds);
        return (OK);
    case HEMT_GGS:
        value->rValue = *(ckt->CKTstate0 + here->HEMTggs);
        return (OK);
    case HEMT_GGD:
        value->rValue = *(ckt->CKTstate0 + here->HEMTggd);
        return (OK);
    case HEMT_QGS:
        value->rValue = *(ckt->CKTstate0 + here->HEMTqgs);
        return (OK);
    case HEMT_CQGS:
        value->rValue = *(ckt->CKTstate0 + here->HEMTcqgs);
        return (OK);
    case HEMT_QGD:
        value->rValue = *(ckt->CKTstate0 + here->HEMTqgd);
        return (OK);
    case HEMT_CQGD:
        value->rValue = *(ckt->CKTstate0 + here->HEMTcqgd);
        return (OK);
    case HEMT_QDS:
        value->rValue = *(ckt->CKTstate0 + here->HEMTqds);
        return (OK);
    case HEMT_CQDS:
        value->rValue = *(ckt->CKTstate0 + here->HEMTcqds);
        return (OK);
    case HEMT_CS :
        if (ckt->CKTcurrentAnalysis & DOING_AC) {
            errMsg = MALLOC(strlen(msg)+1);
            errRtn = "HEMTask";
            strcpy(errMsg,msg);
            return(E_ASKCURRENT);
        } else {
            value->rValue = -*(ckt->CKTstate0 + here->HEMTcd);
            value->rValue -= *(ckt->CKTstate0 + here->HEMTcg);
        }
        return(OK);
    case HEMT_POWER :
        if (ckt->CKTcurrentAnalysis & DOING_AC) {
            errMsg = MALLOC(strlen(msg)+1);
            errRtn = "HEMTask";
            strcpy(errMsg,msg);
            return(E_ASKPOWER);
        } else {
            value->rValue = *(ckt->CKTstate0 + here->HEMTcd) *
                    *(ckt->CKTrhsOld + here->HEMTdrainNode);
            value->rValue += *(ckt->CKTstate0 + here->HEMTcg) *
                    *(ckt->CKTrhsOld + here->HEMTgateNode);
            value->rValue -= (*(ckt->CKTstate0+here->HEMTcd) +
                    *(ckt->CKTstate0 + here->HEMTcg)) *
                    *(ckt->CKTrhsOld + here->HEMTsourceNode);
        }
        return(OK);
    default:
```

```
            return (E_BADPARM);
    }
    /* NOTREACHED */
}
```

A.4 HEMTDEL.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


int
HEMTdelete(inModel,name,inst)
    GENmodel *inModel;
    IFuid name;
    GENinstance **inst;
{
    HEMTmodel *model = (HEMTmodel*)inModel;
    HEMTinstance **fast = (HEMTinstance**)inst;
    HEMTinstance **prev = NULL;
    HEMTinstance *here;

    for( ; model ; model = model->HEMTnextModel) {
        prev = &(model->HEMTinstances);
        for(here = *prev; here ; here = *prev) {
            if(here->HEMTname == name || (fast && here==*fast) ) {
                *prev= here->HEMTnextInstance;
                FREE(here);
                return(OK);
            }
            prev = &(here->HEMTnextInstance);
        }
    }
    return(E_NODEV);
}
```

A.5 HEMTDEST.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "hemtdefs.h"
#include "suffix.h"


void
HEMTdestroy(inModel)
    GENmodel **inModel;
{
    HEMTmodel **model = (HEMTmodel**)inModel;
    HEMTinstance *here;
    HEMTinstance *prev = NULL;
    HEMTmodel *mod = *model;
    HEMTmodel *oldmod = NULL;

    for( ; mod ; mod = mod->HEMTnextModel) {
        if(oldmod) FREE(oldmod);
        oldmod = mod;
        prev = (HEMTinstance *)NULL;
        for(here = mod->HEMTinstances ; here ; here = here->HEMTnextInstance) {
            if(prev) FREE(prev);
            prev = here;
        }
        if(prev) FREE(prev);
    }
    if(oldmod) FREE(oldmod);
    *model = NULL;
}
```

A.6 HEMTGETI.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "cktdefs.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


int
HEMTgetic(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
{
    HEMTmodel *model = (HEMTmodel*)inModel;
    HEMTinstance *here;
    /*
     * Coge las condiciones iniciales fuera del vector rhs.
     *
     */

    for( ; model ; model = model->HEMTnextModel) {
        for(here = model->HEMTinstances; here ; here = here->HEMTnextInstance) {
            if(!here->HEMTicVDSGiven) {
                here->HEMTicVDS =
                        *(ckt->CKTrhs + here->HEMTdrainNode) -
                        *(ckt->CKTrhs + here->HEMTsourceNode);
            }
            if(!here->HEMTicVGSGiven) {
                here->HEMTicVGS =
                        *(ckt->CKTrhs + here->HEMTgateNode) -
                        *(ckt->CKTrhs + here->HEMTsourceNode);
            }
        }
    }
    return(OK);
}
```

A.7 HEMTLOAD.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "devdefs.h"
#include "cktdefs.h"
#include "hemtdefs.h"
#include "const.h"
#include "trandefs.h"
#include <math.h>
#include "sperror.h"
#include "suffix.h"


int
HEMTload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
        /* carga el valor de la resistencia actual dentro de
         * la matriz sparse previamente proporcionada
         */
{
    register HEMTmodel *model = (HEMTmodel*)inModel;
    register HEMTinstance *here;
    double afact;
    double capgd;
    double capgs;
    double capds;
    double cd;
    double cdhat;
    double cdrain;
    double cdreq;
    double ceq;
    double ceqgd;
    double ceqgs;
    double cg;
    double cgd;
    double cghat;
    double csat;
    double delvds;
    double delvgd;
    double delvgs;
    double evgd;
    double evgs;
    double gdpr;
    double gds;
    double geq;
    double ggd;
    double ggs;
    double gm;
    double gspr;
    double vcrit;
    double vds;
    double vgd;
    double vgdt;
```

```
double vgs;
double vgst;
double vto;
double xfact;
int icheck;
int ichk1;
int error;
double raiz1;
double raiz2;
double f1;
double f2;
double f3;
double dif1;
double dif2;
double dif3;
double dif4;
double dif5;
double geme;
double gedese;
double vgsm;
double vdsm;
double cdvc;
double cdvsb;
double vbtvat;
double vgsvbt;
double vgsvat;
double fc1;
double fc2;
double q;
double parentes1;
double parentes2;
double cgs0;
double cgs1;
double rgs;
double rgd;
double ggpr;


/*  bucle por todos los modelos del dispositivo */
for( ; model != NULL; model = model->HEMTnextModel ) {

    /* bucle por todas las istancias del modelo */
    for (here = model->HEMTinstances; here != NULL ;
            here=here->HEMTnextInstance) {

        /*
         *  parametros de modelo dc
         */
        cdvc = model->HEMTcdvc * here->HEMTw;
        cdvsb = model->HEMTcdvsb * here->HEMTw;
        gdpr = model->HEMTdrainConduct * here->HEMTw;
        gspr = model->HEMTsourceConduct * here->HEMTw;
        ggpr = 1.0 / (here->HEMTw * model->HEMTrg);
        csat = model->HEMTgateSatCurrent * here->HEMTw;
        vcrit = model->HEMTvcrit;
        vto = model->HEMTvc-(2.0/model->HEMTbeta);
        cgs0 = (model->HEMTcgs0 * here->HEMTw);
        cgs1 = (model->HEMTcgs1 * here->HEMTw);
        rgs = model->HEMTrgs / here->HEMTw;
        rgd = model->HEMTrgd / here->HEMTw;

        /*
         *    inicializacion
         */
        icheck = 1;
        if( ckt->CKTmode & MODEINITSMSIG) {
```

```
                vgs = *(ckt->CKTstate0 + here->HEMTvgs);
                vgd = *(ckt->CKTstate0 + here->HEMTvgd);
            } else if (ckt->CKTmode & MODEINITTRAN) {
                vgs = *(ckt->CKTstate1 + here->HEMTvgs);
                vgd = *(ckt->CKTstate1 + here->HEMTvgd);
            } else if ( (ckt->CKTmode & MODEINITJCT) &&
                    (ckt->CKTmode & MODETRANOP) &&
                    (ckt->CKTmode & MODEUIC) ) {
                vds = model->HEMTtype*here->HEMTicVDS;
                vgs = model->HEMTtype*here->HEMTicVGS;
                vgd = vgs-vds;
            } else if ( (ckt->CKTmode & MODEINITJCT) &&
                    (here->HEMToff == 0)  ) {
                vgs = -1;
                vgd = -1;
            } else if( (ckt->CKTmode & MODEINITJCT) ||
                    ((ckt->CKTmode & MODEINITFIX) && (here->HEMToff))) {
                vgs = 0;
                vgd = 0;
            } else {
#ifndef PREDICTOR
                if(ckt->CKTmode & MODEINITPRED) {
                    xfact = ckt->CKTdelta/ckt->CKTdeltaOld[2];
                    *(ckt->CKTstate0 + here->HEMTvgs) =
                            *(ckt->CKTstate1 + here->HEMTvgs);
                    vgs = (1+xfact) * *(ckt->CKTstate1 + here->HEMTvgs) -
                           xfact * *(ckt->CKTstate2 + here->HEMTvgs);
                    *(ckt->CKTstate0 + here->HEMTvgd) =
                            *(ckt->CKTstate1 + here->HEMTvgd);
                    vgd = (1+xfact)* *(ckt->CKTstate1 + here->HEMTvgd) -
                           xfact * *(ckt->CKTstate2 + here->HEMTvgd);
                    *(ckt->CKTstate0 + here->HEMTcg) =
                            *(ckt->CKTstate1 + here->HEMTcg);
                    *(ckt->CKTstate0 + here->HEMTcd) =
                            *(ckt->CKTstate1 + here->HEMTcd);
                    *(ckt->CKTstate0 + here->HEMTcgd) =
                            *(ckt->CKTstate1 + here->HEMTcgd);
                    *(ckt->CKTstate0 + here->HEMTgm) =
                            *(ckt->CKTstate1 + here->HEMTgm);
                    *(ckt->CKTstate0 + here->HEMTgds) =
                            *(ckt->CKTstate1 + here->HEMTgds);
                    *(ckt->CKTstate0 + here->HEMTggs) =
                            *(ckt->CKTstate1 + here->HEMTggs);
                    *(ckt->CKTstate0 + here->HEMTggd) =
                            *(ckt->CKTstate1 + here->HEMTggd);
                } else {
#endif /* PREDICTOR */
                    /*
                     *  calcula las nuevas tensiones no lineales de rama
                     */
                    vgs = model->HEMTtype*
                        (*(ckt->CKTrhsOld+here->HEMTgatePrimeNode)-
                        *(ckt->CKTrhsOld+here->HEMTsourcePrimeNode));
                    vgd = model->HEMTtype*
                        (*(ckt->CKTrhsOld+here->HEMTgatePrimeNode)-
                        *(ckt->CKTrhsOld+here->HEMTdrainPrimeNode));
#ifndef PREDICTOR
                }
#endif /* PREDICTOR */
                delvgs=vgs - *(ckt->CKTstate0 + here->HEMTvgs);
                delvgd=vgd - *(ckt->CKTstate0 + here->HEMTvgd);
                delvds=delvgs - delvgd;
                cghat= *(ckt->CKTstate0 + here->HEMTcg) +
                        *(ckt->CKTstate0 + here->HEMTggd)*delvgd +
                        *(ckt->CKTstate0 + here->HEMTggs)*delvgs;
                cdhat= *(ckt->CKTstate0 + here->HEMTcd) +
```

```
                *(ckt->CKTstate0 + here->HEMTgm)*delvgs +
                *(ckt->CKTstate0 + here->HEMTgds)*delvds -
                *(ckt->CKTstate0 + here->HEMTggd)*delvgd;
        /*
         *   bypass si la solucion no ha cambiado
         */
        if((ckt->CKTbypass) &&
            (!(ckt->CKTmode & MODEINITPRED)) &&
            (FABS(delvgs) < ckt->CKTreltol*MAX(FABS(vgs),
                FABS(*(ckt->CKTstate0 + here->HEMTvgs)))+
                ckt->CKTvoltTol) )
        if ( (FABS(delvgd) < ckt->CKTreltol*MAX(FABS(vgd),
                FABS(*(ckt->CKTstate0 + here->HEMTvgd)))+
                ckt->CKTvoltTol))
        if ( (FABS(cghat-*(ckt->CKTstate0 + here->HEMTcg))
                < ckt->CKTreltol*MAX(FABS(cghat),
                FABS(*(ckt->CKTstate0 + here->HEMTcg)))+
                ckt->CKTabstol) ) if ( /* hack - expression too big */
            (FABS(cdhat-*(ckt->CKTstate0 + here->HEMTcd))
                < ckt->CKTreltol*MAX(FABS(cdhat),
                FABS(*(ckt->CKTstate0 + here->HEMTcd)))+
                ckt->CKTabstol) ) {
            /* podemos hacer un bypass */
            vgs= *(ckt->CKTstate0 + here->HEMTvgs);
            vgd= *(ckt->CKTstate0 + here->HEMTvgd);
            vds= vgs-vgd;
            cg= *(ckt->CKTstate0 + here->HEMTcg);
            cd= *(ckt->CKTstate0 + here->HEMTcd);
            cgd= *(ckt->CKTstate0 + here->HEMTcgd);
            gm= *(ckt->CKTstate0 + here->HEMTgm);
            gds= *(ckt->CKTstate0 + here->HEMTgds);
            ggs= *(ckt->CKTstate0 + here->HEMTggs);
            ggd= *(ckt->CKTstate0 + here->HEMTggd);
            goto load;
        }
        /*
         *  limitamos las tensiones de rama no lineales
         */
        ichk1=1;
        vgs = DEVpnjlim(vgs,*(ckt->CKTstate0 + here->HEMTvgs),CONSTvt0,
                vcrit, &icheck);
        vgd = DEVpnjlim(vgd,*(ckt->CKTstate0 + here->HEMTvgd),CONSTvt0,
                vcrit,&ichk1);
        if (ichk1 == 1) {
            icheck=1;
        }
        vgs = DEVfetlim(vgs,*(ckt->CKTstate0 + here->HEMTvgs),
                vto);
        vgd = DEVfetlim(vgd,*(ckt->CKTstate0 + here->HEMTvgd),
                vto);
    }
    /*
     *   determinamos las corrientes dc y sus derivadas
     */
    vds = vgs-vgd;
    if (vgs <= 0) {
          raiz1 = sqrt(FABS(model->HEMTgexp * vgs));
          evgs = exp(MIN(MAX_EXP_ARG,raiz1/(model->HEMTnp * CONSTvt0)));
          ggs = csat * model->HEMTafact *
                model->HEMTgexp * evgs /
                (2.0 * model->HEMTnp * CONSTvt0 * raiz1) +
                ckt->CKTgmin;
          cg = -csat * model->HEMTafact * (evgs-1.0);
    } else {
          evgs = exp (MIN(MAX_EXP_ARG,vgs/(model->HEMTnp * CONSTvt0)));
          ggs = csat*evgs/(model->HEMTnp * CONSTvt0) +
```

```
                ckt->CKTgmin;
        cg = csat * (evgs-1.0);
}

if (vgd <= 0) {
        raiz2 = sqrt(FABS(model->HEMTgexp * vgd));
        evgd = exp(MIN(MAX_EXP_ARG,raiz2/(model->HEMTnp * CONSTvt0)));
        ggd = csat * model->HEMTafact *
              model->HEMTgexp * evgd /
              (2.0 * model->HEMTnp * CONSTvt0 * raiz2) +
              ckt->CKTgmin;
        cgd = -csat * model->HEMTafact * (evgd-1.0);
} else {
        evgd = exp (MIN(MAX_EXP_ARG,vgd/(model->HEMTnp * CONSTvt0)));
        ggd = csat*evgd/(model->HEMTnp * CONSTvt0) +
              ckt->CKTgmin;
        cgd = csat * (evgd-1.0);
}

cg = cg+cgd;
/*
 *   calculamos la corriente de drenador y su derivada
 */
if (vds >= 0) {
    /*
     *   calculamos la corriente de drenador y su derivada
     *   para modo normal
     */
    vgst = vgs-vto;
    if (vgst <= 0) {
        /*
         *   modo normal, region de corte
         */
        cdrain = 0.0;
        gm = 0.0;
        gds = 0.0;
    } else {
        /*
         *   modo normal, regiones lineal y saturacion
         */
    dif1 = (vgs - model->HEMTvc);
    dif5 = (model->HEMTbeta * dif1) +
      (model->HEMTgamma * dif1 * dif1 * dif1);
    f1 = cdvc * (1.0 + tanh(dif5));
    dif4 = model->HEMTdelta * (vgs - model->HEMTvsb);
    f2 = cdvsb * (1.0 + tanh(dif4));
    f1 += f2;
    dif2 = (vgs - vto);
    dif3 = 1.0 + (model->HEMTdx1 * dif2 * dif2);
    f2 = 1.0 + (model->HEMTlModulation * vds / dif3);
    f3 = tanh(model->HEMTalpha * vds);
    cdrain = f1 * f2 * f3;
    geme = model->HEMTbeta + model->HEMTgamma *
      3.0 * dif1 * dif1;
    geme = cdvc * geme /
      (cosh(dif5) * cosh(dif5));
    geme = geme + (cdvsb * model->HEMTdelta /
      (cosh(dif4) * cosh(dif4)));
    geme = geme * f2;
    gedese = model->HEMTlModulation * vds * model->HEMTdx1 *
      2.0 * dif2;
    gedese = f1 * gedese / (dif3 * dif3);
    geme = f3 * (geme - gedese);
    gm = geme;
    gedese = cosh(model->HEMTalpha * vds);
    gedese = f2 * model->HEMTalpha / (gedese * gedese);
```

```
        gedese = gedese + (model->HEMTlModulation * f3 / dif3);
        gedese = f1 * gedese;
        gds = gedese;
        }
    } else {
        /*
         *   calculamos la corriente de drenador y su derivada
         *   para modo invertido
         */
        vgdt = vgd - vto;
        if (vgdt <= 0) {
            /*
             *   modo invertido, region de corte
             */
            cdrain = 0.0;
            gm = 0.0;
            gds = 0.0;
        } else {
            /*
             *   modo invertido, regiones lineal y saturacion
             */
        dif1 = (vgd - model->HEMTvc);
        dif5 = (model->HEMTbeta * dif1) +
          (model->HEMTgamma * dif1 * dif1 * dif1);
        f1 = cdvc * (1.0 + tanh(dif5));
        dif4 = model->HEMTdelta * (vgd - model->HEMTvsb);
        f2 = cdvsb * (1.0 + tanh(dif4));
        f1 += f2;
        dif2 = (vgd - vto);
        dif3 = 1.0 + (model->HEMTdx1 * dif2 * dif2);
        f2 = 1.0 - (model->HEMTlModulation * vds / dif3);
        f3 = -tanh(model->HEMTalpha * vds);
        cdrain = f1 * f2 * f3;
        geme = model->HEMTbeta + model->HEMTgamma *
          3.0 * dif1 * dif1;
        geme = cdvc * geme /
          (cosh(dif5) * cosh(dif5));
        geme = geme + (cdvsb * model->HEMTdelta /
          (cosh(dif4) * cosh(dif4)));
        geme = geme * f2;
        gedese = model->HEMTlModulation * vds * model->HEMTdx1 *
          2.0 * dif2;
        gedese = f1 * gedese / (dif3 * dif3);
        geme = f3 * (geme + gedese);
        gm = geme;
        gedese = cosh(model->HEMTalpha * vds);
        gedese = -f2 * model->HEMTalpha / (gedese * gedese);
        gedese = gedese - (model->HEMTlModulation * f3 / dif3);
        gedese = f1 * gedese;
        gds = gedese;
        }
    }
    /*
     *   calculo de la fuente de corriente de drenador equivalente
     */
    cd = cdrain - cgd;

    if ( (ckt->CKTmode & (MODETRAN|MODEINITSMSIG)) ||
            ((ckt->CKTmode & MODETRANOP) && (ckt->CKTmode & MODEUIC)) ){
        /*
         *    elementos de almacenamiento de carga
         */
    *(ckt->CKTstate0 + here->HEMTqgs) = cgs0 * vgs;
    if (vgs < model->HEMTvat){
        fc1 = 0.0;
        fc2 = 0.0;
```

```
        q = model->HEMTqc1;
        *(ckt->CKTstate0 + here->HEMTqgs) += q;
    } else if (vgs < model->HEMTvbt){
            vgsvat = vgs - model->HEMTvat;
           vbtvat = model->HEMTvbt - model->HEMTvat;
           fc1 = cgs1 * vgsvat / vbtvat;
           fc2 = 1.0 + model->HEMTvst *
                 tanh(model->HEMTalpha * model->HEMTvs * vds);
           q = model->HEMTqc1;
           *(ckt->CKTstate0 + here->HEMTqgs) +=
                 (cgs1 * fc2 * vgsvat * vgsvat /
                 (2.0 * vbtvat)) + q;
    } else {
           vgsvbt = vgs - model->HEMTvbt;
           fc1 = cgs1 *
                 (1.0 + model->HEMTdepletionCapCoeff *
                 pow(vgsvbt,model->HEMTmfact));
           fc2 = 1.0 + model->HEMTvst *
                 tanh(model->HEMTalpha * model->HEMTvs * vds);
           parentes1 = pow(vgsvbt,model->HEMTmfact +1.0) /
                 (model->HEMTmfact + 1.0);
           parentes2 = (model->HEMTvbt + model->HEMTvat)/2.0;
           q = model->HEMTqc1 -
                 (cgs1 * fc2 * parentes2);
           *(ckt->CKTstate0 + here->HEMTqgs) += (cgs1 * fc2 *
                 (vgs + model->HEMTdepletionCapCoeff *
                 parentes1)) + q;
    }
    capgs = cgs0 + fc1 * fc2;

    *(ckt->CKTstate0 + here->HEMTqgd) = cgs0 * vgd;
    if (vgd < model->HEMTvat){
           fc1 = 0.0;
           fc2 = 0.0;
           q = model->HEMTqc1;
           *(ckt->CKTstate0 + here->HEMTqgd) += q;
    } else if (vgd < model->HEMTvbt){
           vgsvat = vgd - model->HEMTvat;
           vbtvat = model->HEMTvbt - model->HEMTvat;
           fc1 = cgs1 * vgsvat / vbtvat;
           fc2 = 1.0 - model->HEMTvst *
                 tanh(model->HEMTalpha * model->HEMTvs * (vds));
           q = model->HEMTqc1;
           *(ckt->CKTstate0 + here->HEMTqgd) +=
                 (cgs1 * fc2 * vgsvat * vgsvat /
                 (2.0 * vbtvat)) + q;
    } else {
           vgsvbt = vgd - model->HEMTvbt;
           fc1 = cgs1 *
                 (1.0 + model->HEMTdepletionCapCoeff *
                 pow(vgsvbt,model->HEMTmfact));
           fc2 = 1.0 - model->HEMTvst *
                 tanh(model->HEMTalpha * model->HEMTvs * (vds));
           parentes1 = pow(vgsvbt,model->HEMTmfact +1.0) /
                 (model->HEMTmfact + 1.0);
           parentes2 = (model->HEMTvbt + model->HEMTvat)/2.0;
           q = model->HEMTqc1 -
                 (cgs1 * fc2 * parentes2);
           *(ckt->CKTstate0 + here->HEMTqgd) += (cgs1 * fc2 *
                 (vgd + model->HEMTdepletionCapCoeff *
                 parentes1)) + q;
    }
    capgd = cgs0 + fc1 * fc2;

    capds = model->HEMTcds * here->HEMTw;
    *(ckt->CKTstate0 + here->HEMTqds) = capds * vds;
```

```
                /*
                 *   almacenamos los parametros de pequeña señal
                 */
                if( (!(ckt->CKTmode & MODETRANOP)) ||
                        (!(ckt->CKTmode & MODEUIC)) ) {
                    if(ckt->CKTmode & MODEINITSMSIG) {
                        *(ckt->CKTstate0 + here->HEMTqgs) = capgs;
                        *(ckt->CKTstate0 + here->HEMTqgd) = capgd;
                        *(ckt->CKTstate0 + here->HEMTqds) = capds;
                        continue;
                    }
                    /*
                     *   analisis transitorio
                     */
                    if(ckt->CKTmode & MODEINITTRAN) {
                        *(ckt->CKTstate1 + here->HEMTqgs) =
                                *(ckt->CKTstate0 + here->HEMTqgs);
                        *(ckt->CKTstate1 + here->HEMTqgd) =
                                *(ckt->CKTstate0 + here->HEMTqgd);
                        *(ckt->CKTstate1 + here->HEMTqds) =
                                *(ckt->CKTstate0 + here->HEMTqds);
                    }
                    error = NIintegrate(ckt,&geq,&ceq,capgs,here->HEMTqgs);
                    if(error) return(error);
                    ggs = ggs + geq;
                    cg = cg + *(ckt->CKTstate0 + here->HEMTcqgs);
                    error = NIintegrate(ckt,&geq,&ceq,capgd,here->HEMTqgd);
                    if(error) return(error);
                    ggd = ggd + geq;
                    cg = cg + *(ckt->CKTstate0 + here->HEMTcqgd);
                    cd = cd - *(ckt->CKTstate0 + here->HEMTcqgd);
                    cgd = cgd + *(ckt->CKTstate0 + here->HEMTcqgd);
                    error = NIintegrate(ckt,&geq,&ceq,capds,here->HEMTqds);
                    if(error) return(error);
                    gds = gds + geq;
                    cd = cd + *(ckt->CKTstate0 + here->HEMTcqds);
                    if (ckt->CKTmode & MODEINITTRAN) {
                        *(ckt->CKTstate1 + here->HEMTcqgs) =
                                *(ckt->CKTstate0 + here->HEMTcqgs);
                        *(ckt->CKTstate1 + here->HEMTcqgd) =
                                *(ckt->CKTstate0 + here->HEMTcqgd);
                        *(ckt->CKTstate1 + here->HEMTcqds) =
                                *(ckt->CKTstate0 + here->HEMTcqds);
                    }
                }
            }
            /*
             *  chequeo de convergencia
             */
            if( (!(ckt->CKTmode & MODEINITFIX)) | (!(ckt->CKTmode & MODEUIC))) {
                if( (icheck == 1)
#ifndef NEWCONV
                        || (FABS(cghat-cg) >= ckt->CKTreltol*
                            MAX(FABS(cghat),FABS(cg))+ckt->CKTabstol) ||
                        (FABS(cdhat-cd) > ckt->CKTreltol*
                            MAX(FABS(cdhat),FABS(cd))+ckt->CKTabstol)
#endif /* NEWCONV */
                        ) {
                    ckt->CKTnoncon++;
                }
            }

            /*
             *    guardamos cosas para la proxima vez
             */
```

```
            *(ckt->CKTstate0 + here->HEMTvgs) = vgs;
            *(ckt->CKTstate0 + here->HEMTvgd) = vgd;
            *(ckt->CKTstate0 + here->HEMTcg) = cg;
            *(ckt->CKTstate0 + here->HEMTcd) = cd;
            *(ckt->CKTstate0 + here->HEMTcgd) = cgd;
            *(ckt->CKTstate0 + here->HEMTgm) = gm;
            *(ckt->CKTstate0 + here->HEMTgds) = gds;
            *(ckt->CKTstate0 + here->HEMTggs) = ggs;
            *(ckt->CKTstate0 + here->HEMTggd) = ggd;
            /*
             *    carga del vector corriente RHS
             */
load:
            ceqgd=model->HEMTtype*(cgd-ggd*vgd);
            ceqgs=model->HEMTtype*((cg-cgd)-ggs*vgs);
            cdreq=model->HEMTtype*((cd+cgd)-gds*vds-gm*vgs);
            *(ckt->CKTrhs + here->HEMTgatePrimeNode) += (-ceqgs-ceqgd);
            *(ckt->CKTrhs + here->HEMTdrainPrimeNode) +=
                    (-cdreq+ceqgd);
            *(ckt->CKTrhs + here->HEMTsourcePrimeNode) +=
                    (cdreq+ceqgs);
            /*
             *    carga de la matriz Y
             */
            *(here->HEMTdrainDrainPrimePtr) += (-gdpr);
            *(here->HEMTgatePrimeDrainPrimePtr) += (-ggd);
            *(here->HEMTgatePrimeSourcePrimePtr) += (-ggs);
            *(here->HEMTsourceSourcePrimePtr) += (-gspr);
            *(here->HEMTdrainPrimeDrainPtr) += (-gdpr);
            *(here->HEMTdrainPrimeGatePrimePtr) += (gm-ggd);
            *(here->HEMTdrainPrimeSourcePrimePtr) += (-gds-gm);
            *(here->HEMTsourcePrimeGatePrimePtr) += (-ggs-gm);
            *(here->HEMTsourcePrimeSourcePtr) += (-gspr);
            *(here->HEMTsourcePrimeDrainPrimePtr) += (-gds);
            *(here->HEMTdrainDrainPtr) += (gdpr);
            *(here->HEMTgatePrimeGatePrimePtr) += (ggd+ggs+ggpr);
            *(here->HEMTsourceSourcePtr) += (gspr);
            *(here->HEMTdrainPrimeDrainPrimePtr) += (gdpr+gds+ggd);
            *(here->HEMTsourcePrimeSourcePrimePtr) += (gspr+gds+gm+ggs);

            *(here->HEMTgateGatePtr) += ggpr;
            *(here->HEMTgateGatePrimePtr) += (-ggpr);
            *(here->HEMTgatePrimeGatePtr) += (-ggpr);

        }
    }
    return(OK);
}
```

A.8 HEMTMASK.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "cktdefs.h"
#include "devdefs.h"
#include "ifsim.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


/* ARGSUSED */
int
HEMTmAsk(ckt,inst,which,value)
    CKTcircuit *ckt;
    GENmodel *inst;
    int which;
    IFvalue *value;
{
    HEMTmodel *here = (HEMTmodel*)inst;
    switch(which) {
        case HEMT_MOD_VTO:
            value->rValue = here->HEMTthreshold;
            return (OK);
        case HEMT_MOD_ALPHA:
            value->rValue = here->HEMTalpha;
            return (OK);
        case HEMT_MOD_BETA:
            value->rValue = here->HEMTbeta;
            return (OK);
        case HEMT_MOD_LAMBDA:
            value->rValue = here->HEMTlModulation;
            return (OK);
        case HEMT_MOD_B:
            value->rValue = here->HEMTb;
            return (OK);
        case HEMT_MOD_RD:
            value->rValue = here->HEMTdrainResist;
            return (OK);
        case HEMT_MOD_RS:
            value->rValue = here->HEMTsourceResist;
            return (OK);
        case HEMT_MOD_CGS:
            value->rValue = here->HEMTcapGS;
            return (OK);
        case HEMT_MOD_CGD:
            value->rValue = here->HEMTcapGD;
            return (OK);
        case HEMT_MOD_PB:
            value->rValue = here->HEMTgatePotential;
            return (OK);
        case HEMT_MOD_IS:
            value->rValue = here->HEMTgateSatCurrent;
            return (OK);
        case HEMT_MOD_FC:
```

```
            value->rValue = here->HEMTdepletionCapCoeff;
            return (OK);
        case HEMT_MOD_DRAINCONDUCT:
            value->rValue = here->HEMTdrainConduct;
            return (OK);
        case HEMT_MOD_SOURCECONDUCT:
            value->rValue = here->HEMTsourceConduct;
            return (OK);
        case HEMT_MOD_DEPLETIONCAP:
            value->rValue = here->HEMTdepletionCap;
            return (OK);
        case HEMT_MOD_VCRIT:
            value->rValue = here->HEMTvcrit;
            return (OK);
        case HEMT_MOD_NP:
            value->rValue = here->HEMTnp;            /*nuevo*/
            return (OK);
        case HEMT_MOD_AFACT:
            value->rValue = here->HEMTafact;         /*nuevo*/
            return (OK);
        case HEMT_MOD_GEXP:
            value->rValue = here->HEMTgexp;          /*nuevo*/
            return (OK);
        case HEMT_MOD_CDVC:
            value->rValue = here->HEMTcdvc;          /*nuevo*/
            return (OK);
        case HEMT_MOD_VC:
            value->rValue = here->HEMTvc;            /*nuevo*/
            return (OK);
        case HEMT_MOD_GAMMA:
            value->rValue = here->HEMTgamma;         /*nuevo*/
            return (OK);
        case HEMT_MOD_CDVSB:
            value->rValue = here->HEMTcdvsb;         /*nuevo*/
            return (OK);
        case HEMT_MOD_DELTA:
            value->rValue = here->HEMTdelta;         /*nuevo*/
            return (OK);
        case HEMT_MOD_VSB:
            value->rValue = here->HEMTvsb;           /*nuevo*/
            return (OK);
        case HEMT_MOD_DXL:
            value->rValue = here->HEMTdxl;           /*nuevo*/
            return (OK);
        case HEMT_MOD_VAT:
            value->rValue = here->HEMTvat;           /*nuevo*/
            return (OK);
        case HEMT_MOD_VBT:
            value->rValue = here->HEMTvbt;           /*nuevo*/
            return (OK);
        case HEMT_MOD_CGS0:
            value->rValue = here->HEMTcgs0;          /*nuevo*/
            return (OK);
        case HEMT_MOD_CGS1:
            value->rValue = here->HEMTcgs1;          /*nuevo*/
            return (OK);
        case HEMT_MOD_VST:
            value->rValue = here->HEMTvst;           /*nuevo*/
            return (OK);
        case HEMT_MOD_VS:
            value->rValue = here->HEMTvs;            /*nuevo*/
            return (OK);
        case HEMT_MOD_MFACT:
            value->rValue = here->HEMTmfact;         /*nuevo*/
            return (OK);
        case HEMT_MOD_RG:
```

```
            value->rValue = here->HEMTrg;           /*nuevo*/
            return (OK);
        case HEMT_MOD_RGS:
            value->rValue = here->HEMTrgs;          /*nuevo*/
            return (OK);
        case HEMT_MOD_RGD:
            value->rValue = here->HEMTrgd;          /*nuevo*/
            return (OK);
        case HEMT_MOD_QC1:
            value->rValue = here->HEMTqc1;          /*nuevo*/
            return (OK);
        case HEMT_MOD_CDS:
            value->rValue = here->HEMTcds;          /*nuevo*/
            return (OK);
        case HEMT_MOD_TD:
            value->rValue = here->HEMTtd;           /*nuevo*/
            return (OK);

        default:
            return (E_BADPARM);
    }
    /* NOTREACHED */
}
```

A.9 HEMTMDEL.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


int
HEMTmDelete(inModel,modname,kill)
    GENmodel **inModel;
    IFuid modname;
    GENmodel *kill;
{
    HEMTmodel **model = (HEMTmodel**)inModel;
    HEMTmodel *modfast = (HEMTmodel*)kill;
    HEMTinstance *here;
    HEMTinstance *prev = NULL;
    HEMTmodel **oldmod;
    oldmod = model;
    for( ; *model ; model = &((*model)->HEMTnextModel)) {
        if( (*model)->HEMTmodName == modname ||
                (modfast && *model == modfast) ) goto delgot;
        oldmod = model;
    }
    return(E_NOMOD);

delgot:
    *oldmod = (*model)->HEMTnextModel;
    for(here = (*model)->HEMTinstances ; here ; here = here->HEMTnextInstance) {
        if(prev) FREE(prev);
        prev = here;
    }
    if(prev) FREE(prev);
    FREE(*model);
    return(OK);

}
```

A.10 HEMTMPAR.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "ifsim.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


int
HEMTmParam(param,value,inModel)
    int param;
    IFvalue *value;
    GENmodel *inModel;
{
    HEMTmodel *model = (HEMTmodel*)inModel;
    switch(param) {
        case HEMT_MOD_VTO:
            model->HEMTthresholdGiven = TRUE;
            model->HEMTthreshold = value->rValue;
            break;
        case HEMT_MOD_ALPHA:
            model->HEMTalphaGiven = TRUE;
            model->HEMTalpha = value->rValue;
            break;
        case HEMT_MOD_BETA:
            model->HEMTbetaGiven = TRUE;
            model->HEMTbeta = value->rValue;
            break;
        case HEMT_MOD_LAMBDA:
            model->HEMTlModulationGiven = TRUE;
            model->HEMTlModulation = value->rValue;
            break;
        case HEMT_MOD_B:
            model->HEMTbGiven = TRUE;
            model->HEMTb = value->rValue;
            break;
        case HEMT_MOD_RD:
            model->HEMTdrainResistGiven = TRUE;
            model->HEMTdrainResist = value->rValue;
            break;
        case HEMT_MOD_RS:
            model->HEMTsourceResistGiven = TRUE;
            model->HEMTsourceResist = value->rValue;
            break;
        case HEMT_MOD_CGS:
            model->HEMTcapGSGiven = TRUE;
            model->HEMTcapGS = value->rValue;
            break;
        case HEMT_MOD_CGD:
            model->HEMTcapGDGiven = TRUE;
            model->HEMTcapGD = value->rValue;
            break;
        case HEMT_MOD_PB:
```

```
            model->HEMTgatePotentialGiven = TRUE;
            model->HEMTgatePotential = value->rValue;
            break;
        case HEMT_MOD_IS:
            model->HEMTgateSatCurrentGiven = TRUE;
            model->HEMTgateSatCurrent = value->rValue;
            break;
        case HEMT_MOD_FC:
            model->HEMTdepletionCapCoeffGiven = TRUE;
            model->HEMTdepletionCapCoeff = value->rValue;
            break;
        case HEMT_MOD_NP:
            model->HEMTnpGiven = TRUE;
            model->HEMTnp = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_AFACT:
            model->HEMTafactGiven = TRUE;
            model->HEMTafact = value->rValue;       /*nuevo*/
            break;
        case HEMT_MOD_GEXP:
            model->HEMTgexpGiven = TRUE;
            model->HEMTgexp = value->rValue;        /*nuevo*/
            break;
        case HEMT_MOD_CDVC:
            model->HEMTcdvcGiven = TRUE;
            model->HEMTcdvc = value->rValue;        /*nuevo*/
            break;
        case HEMT_MOD_VC:
            model->HEMTvcGiven = TRUE;
            model->HEMTvc = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_GAMMA:
            model->HEMTgammaGiven = TRUE;
            model->HEMTgamma = value->rValue;       /*nuevo*/
            break;
        case HEMT_MOD_CDVSB:
            model->HEMTcdvsbGiven = TRUE;
            model->HEMTcdvsb = value->rValue;       /*nuevo*/
            break;
        case HEMT_MOD_DELTA:
            model->HEMTdeltaGiven = TRUE;
            model->HEMTdelta = value->rValue;       /*nuevo*/
            break;
        case HEMT_MOD_VSB:
            model->HEMTvsbGiven = TRUE;
            model->HEMTvsb = value->rValue;         /*nuevo*/
            break;
        case HEMT_MOD_DXL:
            model->HEMTdxlGiven = TRUE;
            model->HEMTdxl = value->rValue;         /*nuevo*/
            break;
        case HEMT_MOD_VAT:
            model->HEMTvatGiven = TRUE;
            model->HEMTvat = value->rValue;         /*nuevo*/
            break;
        case HEMT_MOD_VBT:
            model->HEMTvbtGiven = TRUE;
            model->HEMTvbt = value->rValue;         /*nuevo*/
            break;
        case HEMT_MOD_CGS0:
            model->HEMTcgs0Given = TRUE;
            model->HEMTcgs0 = value->rValue;        /*nuevo*/
            break;
        case HEMT_MOD_CGS1:
            model->HEMTcgs1Given = TRUE;
            model->HEMTcgs1 = value->rValue;        /*nuevo*/
```

```
            break;
        case HEMT_MOD_VST:
            model->HEMTvstGiven = TRUE;
            model->HEMTvst = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_VS:
            model->HEMTvsGiven = TRUE;
            model->HEMTvs = value->rValue;           /*nuevo*/
            break;
        case HEMT_MOD_MFACT:
            model->HEMTmfactGiven = TRUE;
            model->HEMTmfact = value->rValue;        /*nuevo*/
            break;
        case HEMT_MOD_RG:
            model->HEMTrgGiven = TRUE;
            model->HEMTrg = value->rValue;           /*nuevo*/
            break;
        case HEMT_MOD_RGS:
            model->HEMTrgsGiven = TRUE;
            model->HEMTrgs = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_RGD:
            model->HEMTrgdGiven = TRUE;
            model->HEMTrgd = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_QC1:
            model->HEMTqc1Given = TRUE;
            model->HEMTqc1 = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_CDS:
            model->HEMTcdsGiven = TRUE;
            model->HEMTcds = value->rValue;          /*nuevo*/
            break;
        case HEMT_MOD_TD:
            model->HEMTtdGiven = TRUE;
            model->HEMTtd = value->rValue;           /*nuevo*/
            break;
        case HEMT_MOD_NMF:
            if(value->iValue) {
                model->HEMTtype = NMF;
            }
            break;
        case HEMT_MOD_PMF:
            if(value->iValue) {
                model->HEMTtype = PMF;
            }
            break;
      case HEMT_MOD_KF:
          model->HEMTfNcoefGiven = TRUE;
          model->HEMTfNcoef = value->rValue;
          break;
      case HEMT_MOD_AF:
          model->HEMTfNexpGiven = TRUE;
          model->HEMTfNexp = value->rValue;
          break;
        default:
            return(E_BADPARM);
    }
    return(OK);
}
```

A.11 HEMTPARA.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "ifsim.h"
#include "util.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


/* ARGSUSED */
int
HEMTparam(param,value,inst,select)
    int param;
    IFvalue *value;
    GENinstance *inst;
    IFvalue *select;
{
    HEMTinstance *here = (HEMTinstance*)inst;
    switch(param) {
        case HEMT_W:
            here->HEMTw = value->rValue;
            here->HEMTwGiven = TRUE;
            break;
        case HEMT_IC_VDS:
            here->HEMTicVDS = value->rValue;
            here->HEMTicVDSGiven = TRUE;
            break;
        case HEMT_IC_VGS:
            here->HEMTicVGS = value->rValue;
            here->HEMTicVGSGiven = TRUE;
            break;
        case HEMT_OFF:
            here->HEMToff = value->iValue;
            break;
        case HEMT_IC:
            switch(value->v.numValue) {
                case 2:
                    here->HEMTicVGS = *(value->v.vec.rVec+1);
                    here->HEMTicVGSGiven = TRUE;
                case 1:
                    here->HEMTicVDS = *(value->v.vec.rVec);
                    here->HEMTicVDSGiven = TRUE;
                    break;
                default:
                    return(E_BADPARM);
            }
            break;
        default:
            return(E_BADPARM);
    }
    return(OK);
}
```

A.12 HEMTSETU.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "smpdefs.h"
#include "cktdefs.h"
#include "hemtdefs.h"
#include "const.h"
#include <math.h>
#include "sperror.h"
#include "suffix.h"


int
HEMTsetup(matrix,inModel,ckt,states)
    register SMPmatrix *matrix;
    GENmodel *inModel;
    CKTcircuit *ckt;
    int *states;
        /* carga la estructura del dispositivo con aquellos punteros
         * que se van a necesitar mas tarde para que la carga de la
         * matriz sea mas rapida
         */
{
    register HEMTmodel *model = (HEMTmodel*)inModel;
    register HEMTinstance *here;
    int error;
    CKTnode *tmp;

    /*  bucle por todos los modelos del dispositivo */
    for( ; model != NULL; model = model->HEMTnextModel ) {

        if( (model->HEMTtype != NMF) && (model->HEMTtype != PMF) ) {
            model->HEMTtype = NMF;
        }
        if(!model->HEMTthresholdGiven) {
            model->HEMTthreshold = -2;
        }
        if(!model->HEMTbetaGiven) {
            model->HEMTbeta = 2.5e-3;
        }
        if(!model->HEMTbGiven) {
            model->HEMTb = 0.3;
        }
        if(!model->HEMTalphaGiven) {
            model->HEMTalpha = 2;
        }
        if(!model->HEMTlModulationGiven) {
            model->HEMTlModulation = 0;
        }
        if(!model->HEMTdrainResistGiven) {
            model->HEMTdrainResist = 0;
        }
        if(!model->HEMTsourceResistGiven) {
            model->HEMTsourceResist = 0;
```

```
        }
        if(!model->HEMTcapGSGiven) {
            model->HEMTcapGS = 0;
        }
        if(!model->HEMTcapGDGiven) {
            model->HEMTcapGD = 0;
        }
        if(!model->HEMTcdsGiven) {
            model->HEMTcds = 0;
        }
        if(!model->HEMTgatePotentialGiven) {
            model->HEMTgatePotential = 1;
        }
        if(!model->HEMTgateSatCurrentGiven) {
            model->HEMTgateSatCurrent = 1e-14;
        }
        if(!model->HEMTdepletionCapCoeffGiven) {
            model->HEMTdepletionCapCoeff = .5;
        }
      if(!model->HEMTfNcoefGiven) {
          model->HEMTfNcoef = 0;
      }
      if(!model->HEMTfNexpGiven) {
          model->HEMTfNexp = 1;
      }

        /* bucle por todas las istancias del modelo */
        for (here = model->HEMTinstances; here != NULL ;
                here=here->HEMTnextInstance) {

            if(!here->HEMTwGiven) {
                here->HEMTw = 1;
            }
            here->HEMTstate = *states;
            *states += 15;

            if(model->HEMTsourceResist != 0 && here->HEMTsourcePrimeNode==0) {
                error = CKTmkVolt(ckt,&tmp,here->HEMTname,"source");
                if(error) return(error);
                here->HEMTsourcePrimeNode = tmp->number;
            } else {
                here->HEMTsourcePrimeNode = here->HEMTsourceNode;
            }
            if(model->HEMTdrainResist != 0 && here->HEMTdrainPrimeNode==0) {
                error = CKTmkVolt(ckt,&tmp,here->HEMTname,"drain");
                if(error) return(error);
                here->HEMTdrainPrimeNode = tmp->number;
            } else {
                here->HEMTdrainPrimeNode = here->HEMTdrainNode;
            }
            if(model->HEMTrg != 0 && here->HEMTgatePrimeNode==0) {
                error = CKTmkVolt(ckt,&tmp,here->HEMTname,"gate");
                if(error) return(error);
                here->HEMTgatePrimeNode = tmp->number;
            } else {
                here->HEMTgatePrimeNode = here->HEMTgateNode;
            }


/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==(double *)NULL){\
    return(E_NOMEM);\
}

            TSTALLOC(HEMTdrainDrainPrimePtr,HEMTdrainNode,HEMTdrainPrimeNode)
```

```
                TSTALLOC(HEMTgatePrimeDrainPrimePtr,HEMTgatePrimeNode,
                        HEMTdrainPrimeNode)
                TSTALLOC(HEMTgatePrimeSourcePrimePtr,HEMTgatePrimeNode,
                        HEMTsourcePrimeNode)
                TSTALLOC(HEMTsourceSourcePrimePtr,HEMTsourceNode,
                        HEMTsourcePrimeNode)
                TSTALLOC(HEMTdrainPrimeDrainPtr,HEMTdrainPrimeNode,HEMTdrainNode)
                TSTALLOC(HEMTdrainPrimeGatePrimePtr,HEMTdrainPrimeNode,
                        HEMTgatePrimeNode)
                TSTALLOC(HEMTdrainPrimeSourcePrimePtr,HEMTdrainPrimeNode,
                        HEMTsourcePrimeNode)
                TSTALLOC(HEMTsourcePrimeGatePrimePtr,HEMTsourcePrimeNode,
                        HEMTgatePrimeNode)
                TSTALLOC(HEMTsourcePrimeSourcePtr,HEMTsourcePrimeNode,
                        HEMTsourceNode)
                TSTALLOC(HEMTsourcePrimeDrainPrimePtr,HEMTsourcePrimeNode,
                        HEMTdrainPrimeNode)
                TSTALLOC(HEMTdrainDrainPtr,HEMTdrainNode,HEMTdrainNode)
                TSTALLOC(HEMTgatePrimeGatePrimePtr,HEMTgatePrimeNode,
                        HEMTgatePrimeNode)
                TSTALLOC(HEMTsourceSourcePtr,HEMTsourceNode,HEMTsourceNode)
                TSTALLOC(HEMTdrainPrimeDrainPrimePtr,HEMTdrainPrimeNode,
                        HEMTdrainPrimeNode)
                TSTALLOC(HEMTsourcePrimeSourcePrimePtr,HEMTsourcePrimeNode,
                        HEMTsourcePrimeNode)
                TSTALLOC(HEMTgateGatePtr,HEMTgateNode,HEMTgateNode)
                TSTALLOC(HEMTgateGatePrimePtr,HEMTgateNode,
                        HEMTgatePrimeNode)
                TSTALLOC(HEMTgatePrimeGatePtr,HEMTgatePrimeNode,
                        HEMTgateNode)

        }
    }
    return(OK);
}
```

A.13 HEMTTEMP.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "smpdefs.h"
#include "cktdefs.h"
#include "hemtdefs.h"
#include "const.h"
#include <math.h>
#include "sperror.h"
#include "suffix.h"


/* ARGSUSED */
int
HEMTtemp(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
        /* carga la estructructura del dispositivo con aquellos punteros
         * que van a hacer falta mas tarde para hacer que la carga de la
         * matriz sea mas rapida
         */
{
    register HEMTmodel *model = (HEMTmodel*)inModel;
    register HEMTinstance *here;
    double xfc, temp;

    /*  bucle por todos los modelos del dispositivo */
    for( ; model != NULL; model = model->HEMTnextModel ) {

        /* bucle por todas las istancias del modelo */
        for (here = model->HEMTinstances; here != NULL ;
                here=here->HEMTnextInstance) {
              if(model->HEMTdrainResist != 0) {
                model->HEMTdrainConduct = 1/model->HEMTdrainResist;
                  } else {
                model->HEMTdrainConduct = 0;
            }
            if(model->HEMTsourceResist != 0) {
                model->HEMTsourceConduct = 1/model->HEMTsourceResist;
            } else {
                model->HEMTsourceConduct = 0;
            }
            model->HEMTvcrit = CONSTvt0 * log(CONSTvt0/
                  (CONSTroot2 * model->HEMTgateSatCurrent));
      }
    }
    return(OK);
}
```

A.14 HEMTTRUN.C

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
/*
 */

#include "spice.h"
#include <stdio.h>
#include "cktdefs.h"
#include "hemtdefs.h"
#include "sperror.h"
#include "suffix.h"


int
HEMTtrunc(inModel,ckt,timeStep)
    GENmodel *inModel;
    register CKTcircuit *ckt;
    double *timeStep;
{
    register HEMTmodel *model = (HEMTmodel*)inModel;
    register HEMTinstance *here;

    for( ; model != NULL; model = model->HEMTnextModel) {
        for(here=model->HEMTinstances;here!=NULL;here = here->HEMTnextInstance){
            CKTterr(here->HEMTqgs,ckt,timeStep);
            CKTterr(here->HEMTqgd,ckt,timeStep);
            CKTterr(here->HEMTqds,ckt,timeStep);
        }
    }
    return(OK);
}
```

A.15 HEMTDEFS.H

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/

#ifndef HEMT
#define HEMT

#include "ifsim.h"
#include "cktdefs.h"
#include "gendefs.h"
#include "complex.h"
#include "noisedef.h"

    /* estructuras utilizadas para describir a los trans. HEMT */


/* information utilizada para describir una unica instancia */

typedef struct sHEMTinstance {
    struct sHEMTmodel *HEMTmodPtr;    /* puntero al modelo */
    struct sHEMTinstance *HEMTnextInstance; /* puntero a la siguiente instancia
                                             * del modelo actual*/
    IFuid HEMTname; /* puntero al string de caracteres donde se encuentra el
                     * nombre de esta instancia */

    int HEMTdrainNode;  /* numero del nodo de drenador del hemt */
    int HEMTgateNode;   /* numero del nodo de puerta del hemt */
    int HEMTsourceNode; /* numero del nodo de surtidor del hemt */
    int HEMTdrainPrimeNode;  /* numero del nodo interno de drenador del hemt  */
    int HEMTsourcePrimeNode; /* numero del nodo interno de surtidor del hemt */
    int HEMTgatePrimeNode;   /* numero del nodo interno de puerta del hemt */
    double HEMTw;    /* factor w para el hemt */
    double HEMTicVDS;   /* condicion inicial de la tension D-S*/
    double HEMTicVGS;   /* condicion inicial de la tension G-S*/
    double *HEMTdrainDrainPrimePtr; /* puntero a la matriz sparse
                                     * (drain,drain prime) */
    double *HEMTgatePrimeDrainPrimePtr;  /* puntero a la matriz sparse
                                          * (gate prime,drain prime) */
    double *HEMTgatePrimeSourcePrimePtr; /* puntero a la matriz sparse
                                          * (gate prime,source prime) */
    double *HEMTsourceSourcePrimePtr;    /* puntero a la matriz sparse
                                          * (source,source prime) */
    double *HEMTdrainPrimeDrainPtr; /* puntero a la matriz sparse
                                     * (drain prime,drain) */
    double *HEMTdrainPrimeGatePrimePtr;  /* puntero a la matriz sparse
                                          * (drain prime,gate prime) */
    double *HEMTdrainPrimeSourcePrimePtr;   /* puntero a la matriz sparse
                                             * (drain prime,source prime) */
    double *HEMTsourcePrimeGatePrimePtr; /* puntero a la matriz sparse
                                          * (source prime,gate prime) */
    double *HEMTsourcePrimeSourcePtr;   /* puntero a la matriz sparse
                                         * (source prime,source) */
    double *HEMTsourcePrimeDrainPrimePtr;   /* puntero a la matriz sparse
                                             * (source prime,drain prime) */
    double *HEMTdrainDrainPtr;  /* puntero a la matriz sparse
                                 * (drain,drain) */
    double *HEMTgatePrimeGatePrimePtr;    /* puntero a la matriz sparse
                                           * (gate prime,gate prime) */
    double *HEMTsourceSourcePtr;    /* puntero a la matriz sparse
                                     * (source,source) */
```

```
    double *HEMTdrainPrimeDrainPrimePtr;    /* puntero a la matriz sparse
                                             * (drain prime,drain prime) */
    double *HEMTsourcePrimeSourcePrimePtr;  /* puntero a la matriz sparse
                                             * (source prime,source prime) */

    double *HEMTgateGatePtr;  /* puntero a la matriz sparse
                               * (gate,gate) */
    double *HEMTgateGatePrimePtr; /* puntero a la matriz sparse
                                   * (gate,gate prime) */
    double *HEMTgatePrimeGatePtr; /* puntero a la matriz sparse
                                   * (gate prime,gate) */


    int HEMTstate; /* puntero inicio del vector de estado del hemt */
#define HEMTvgs HEMTstate
#define HEMTvgd HEMTstate+1
#define HEMTcg HEMTstate+2
#define HEMTcd HEMTstate+3
#define HEMTcgd HEMTstate+4
#define HEMTgm HEMTstate+5
#define HEMTgds HEMTstate+6
#define HEMTggs HEMTstate+7
#define HEMTggd HEMTstate+8
#define HEMTqgs HEMTstate+9
#define HEMTcqgs HEMTstate+10
#define HEMTqgd HEMTstate+11
#define HEMTcqgd HEMTstate+12
#define HEMTqds HEMTstate+13    /*nuevo*/
#define HEMTcqds HEMTstate+14   /*nuevo*/

    int HEMToff;   /* flag 'off' para el hemt */
    unsigned HEMTwGiven : 1;  /* flag para indicar que fue espacificada la w */
    unsigned HEMTicVDSGiven : 1;  /* flag de condicion inicial dada para V D-S*/
    unsigned HEMTicVGSGiven : 1;  /* flag de condicion inicial dada para V G-S*/

int HEMTmode;

/*
 * convencion de nomenclatura:
 * x = vgs
 * y = vgd
 * z = vds
 * cdr = cdrain
 */

#define HEMTNDCOEFFS    27

#ifndef NODISTO
      double HEMTdCoeffs[HEMTNDCOEFFS];
#else /* NODISTO */
      double *HEMTdCoeffs;
#endif /* NODISTO */

#ifndef CONFIG

#define     cdr_x       HEMTdCoeffs[0]
#define     cdr_z       HEMTdCoeffs[1]
#define     cdr_x2            HEMTdCoeffs[2]
#define     cdr_z2            HEMTdCoeffs[3]
#define     cdr_xz            HEMTdCoeffs[4]
#define     cdr_x3            HEMTdCoeffs[5]
#define     cdr_z3            HEMTdCoeffs[6]
#define     cdr_x2z           HEMTdCoeffs[7]
#define     cdr_xz2           HEMTdCoeffs[8]

#define     ggs3        HEMTdCoeffs[9]
```

```
#define     ggd3        HEMTdCoeffs[10]
#define     ggs2        HEMTdCoeffs[11]
#define     ggd2        HEMTdCoeffs[12]

#define     qgs_x2          HEMTdCoeffs[13]
#define     qgs_y2          HEMTdCoeffs[14]
#define     qgs_xy          HEMTdCoeffs[15]
#define     qgs_x3          HEMTdCoeffs[16]
#define     qgs_y3          HEMTdCoeffs[17]
#define     qgs_x2y         HEMTdCoeffs[18]
#define     qgs_xy2         HEMTdCoeffs[19]

#define     qgd_x2          HEMTdCoeffs[20]
#define     qgd_y2          HEMTdCoeffs[21]
#define     qgd_xy          HEMTdCoeffs[22]
#define     qgd_x3          HEMTdCoeffs[23]
#define     qgd_y3          HEMTdCoeffs[24]
#define     qgd_x2y         HEMTdCoeffs[25]
#define     qgd_xy2         HEMTdCoeffs[26]

#endif

/* indices al array de las fuentes de ruido del HEMT */

#define HEMTRDNOIZ       0
#define HEMTRSNOIZ       1
#define HEMTIDNOIZ       2
#define HEMTFLNOIZ 3
#define HEMTTOTNOIZ    4

#define HEMTNSRCS     5     /* numero de fuentes de ruido del HEMT */

#ifndef NONOISE
    double HEMTnVar[NSTATVARS][HEMTNSRCS];
#else /* NONOISE */
      double **HEMTnVar;
#endif /* NONOISE */

} HEMTinstance ;


/* datos por cada modelo */

typedef struct sHEMTmodel {       /* estructura del modelo para el hemt */
    int HEMTmodType; /* indice para el tipo de dispositivo */
    struct sHEMTmodel *HEMTnextModel;   /* puntero al siguiente modelo posible
                                         * dentro de la lista enlazada */
    HEMTinstance * HEMTinstances; /* puntero a la lista de instancias
                                   * que tiene este modelo */
    IFuid HEMTmodName; /* puntero al string de caracteres donde se encuentra el
                        * nombre de este modelo */
    int HEMTtype;

    double HEMTthreshold;
    double HEMTalpha;
    double HEMTbeta;
    double HEMTlModulation;
    double HEMTb;
    double HEMTdrainResist;
    double HEMTsourceResist;
    double HEMTcapGS;
    double HEMTcapGD;
    double HEMTgatePotential;
    double HEMTgateSatCurrent;
    double HEMTdepletionCapCoeff;
    double HEMTfNcoef;
```

```
double HEMTfNexp;
double HEMTnp;      /*nuevo*/
double HEMTafact;   /*nuevo*/
double HEMTgexp;    /*nuevo*/
double HEMTcdvc;    /*nuevo*/
double HEMTvc;      /*nuevo*/
double HEMTgamma;   /*nuevo*/
double HEMTcdvsb;   /*nuevo*/
double HEMTdelta;   /*nuevo*/
double HEMTvsb;     /*nuevo*/
double HEMTdxl;     /*nuevo*/
double HEMTvat;     /*nuevo*/
double HEMTvbt;     /*nuevo*/
double HEMTcgs0;    /*nuevo*/
double HEMTcgs1;    /*nuevo*/
double HEMTvst;     /*nuevo*/
double HEMTvs;      /*nuevo*/
double HEMTmfact;   /*nuevo*/
double HEMTrg;      /*nuevo*/
double HEMTrgs;     /*nuevo*/
double HEMTrgd;     /*nuevo*/
double HEMTqc1;     /*nuevo*/
double HEMTcds;     /*nuevo*/
double HEMTtd;      /*nuevo*/


double HEMTdrainConduct;
double HEMTsourceConduct;
double HEMTdepletionCap;
double HEMTf1;
double HEMTf2;
double HEMTf3;
double HEMTvcrit;

unsigned HEMTthresholdGiven : 1;
unsigned HEMTalphaGiven : 1;
unsigned HEMTbetaGiven : 1;
unsigned HEMTlModulationGiven : 1;
unsigned HEMTbGiven : 1;
unsigned HEMTdrainResistGiven : 1;
unsigned HEMTsourceResistGiven : 1;
unsigned HEMTcapGSGiven : 1;
unsigned HEMTcapGDGiven : 1;
unsigned HEMTgatePotentialGiven : 1;
unsigned HEMTgateSatCurrentGiven : 1;
unsigned HEMTdepletionCapCoeffGiven : 1;
unsigned HEMTfNcoefGiven : 1;
unsigned HEMTfNexpGiven : 1;
unsigned HEMTnpGiven : 1;           /*nuevo*/
unsigned HEMTafactGiven : 1;        /*nuevo*/
unsigned HEMTgexpGiven : 1;         /*nuevo*/
unsigned HEMTcdvcGiven : 1;         /*nuevo*/
unsigned HEMTvcGiven : 1;           /*nuevo*/
unsigned HEMTgammaGiven : 1;        /*nuevo*/
unsigned HEMTcdvsbGiven : 1;        /*nuevo*/
unsigned HEMTdeltaGiven : 1;        /*nuevo*/
unsigned HEMTvsbGiven : 1;          /*nuevo*/
unsigned HEMTdxlGiven : 1;          /*nuevo*/
unsigned HEMTvatGiven : 1;          /*nuevo*/
unsigned HEMTvbtGiven : 1;          /*nuevo*/
unsigned HEMTcgs0Given : 1;         /*nuevo*/
unsigned HEMTcgs1Given : 1;         /*nuevo*/
unsigned HEMTvstGiven : 1;          /*nuevo*/
unsigned HEMTvsGiven : 1;           /*nuevo*/
unsigned HEMTmfactGiven : 1;        /*nuevo*/
unsigned HEMTrgGiven : 1;           /*nuevo*/
```

```
    unsigned HEMTrgsGiven : 1;          /*nuevo*/
    unsigned HEMTrgdGiven : 1;          /*nuevo*/
    unsigned HEMTqc1Given : 1;          /*nuevo*/
    unsigned HEMTcdsGiven : 1;          /*nuevo*/
    unsigned HEMTtdGiven : 1;           /*nuevo*/


} HEMTmodel;

#ifndef NMF

#define NMF 1
#define PMF -1

#endif /*NMF*/

/* parametros de dispositivo */
#define HEMT_W 1
#define HEMT_IC_VDS 2
#define HEMT_IC_VGS 3
#define HEMT_IC 4
#define HEMT_OFF 5
#define HEMT_CS 6
#define HEMT_POWER 7

/* parametros de modelo */
#define HEMT_MOD_VTO 101
#define HEMT_MOD_ALPHA 102
#define HEMT_MOD_BETA 103
#define HEMT_MOD_LAMBDA 104
#define HEMT_MOD_B 105
#define HEMT_MOD_RD 106
#define HEMT_MOD_RS 107
#define HEMT_MOD_CGS 108
#define HEMT_MOD_CGD 109
#define HEMT_MOD_PB 110
#define HEMT_MOD_IS 111
#define HEMT_MOD_FC 112
#define HEMT_MOD_NMF 113
#define HEMT_MOD_PMF 114
#define HEMT_MOD_KF 115
#define HEMT_MOD_AF 116
#define HEMT_MOD_NP 117      /*nuevo*/
#define HEMT_MOD_AFACT 118   /*nuevo*/
#define HEMT_MOD_GEXP 119    /*nuevo*/
#define HEMT_MOD_CDVC 120    /*nuevo*/
#define HEMT_MOD_VC 121      /*nuevo*/
#define HEMT_MOD_GAMMA 122   /*nuevo*/
#define HEMT_MOD_CDVSB 123   /*nuevo*/
#define HEMT_MOD_DELTA 124   /*nuevo*/
#define HEMT_MOD_VSB 125     /*nuevo*/
#define HEMT_MOD_DXL 126     /*nuevo*/
#define HEMT_MOD_VAT 127     /*nuevo*/
#define HEMT_MOD_VBT 128     /*nuevo*/
#define HEMT_MOD_CGS0 129    /*nuevo*/
#define HEMT_MOD_CGS1 130    /*nuevo*/
#define HEMT_MOD_VST 131     /*nuevo*/
#define HEMT_MOD_VS 132      /*nuevo*/
#define HEMT_MOD_MFACT 133   /*nuevo*/
#define HEMT_MOD_RG 134      /*nuevo*/
#define HEMT_MOD_RGS 135     /*nuevo*/
#define HEMT_MOD_RGD 136     /*nuevo*/
#define HEMT_MOD_QC1 137     /*nuevo*/
#define HEMT_MOD_CDS 138     /*nuevo*/
#define HEMT_MOD_TD 139      /*nuevo*/
```

```
/* preguntas de dispositivo */

#define HEMT_DRAINNODE       201
#define HEMT_GATENODE        202
#define HEMT_SOURCENODE      203
#define HEMT_DRAINPRIMENODE  204
#define HEMT_GATEPRIMENODE   205
#define HEMT_SOURCEPRIMENODE 206

#define HEMT_VGS         207
#define HEMT_VGD         208
#define HEMT_CG          209
#define HEMT_CD          210
#define HEMT_CGD         211
#define HEMT_GM          212
#define HEMT_GDS         213
#define HEMT_GGS         214
#define HEMT_GGD         215
#define HEMT_QGS         216
#define HEMT_CQGS        217
#define HEMT_QGD         218
#define HEMT_CQGD        219
#define HEMT_QDS         220   /*nuevo*/
#define HEMT_CQDS        221   /*nuevo*/


/* preguntas de modelo */

#define HEMT_MOD_DRAINCONDUCT    301
#define HEMT_MOD_SOURCECONDUCT   302
#define HEMT_MOD_DEPLETIONCAP    303
#define HEMT_MOD_VCRIT       304

#include "hemtext.h"

#endif /*HEMT*/
```

A.16 HEMTEXT.H

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/

#ifdef __STDC__
extern int HEMTacLoad(GENmodel*,CKTcircuit*);
extern int HEMTask(CKTcircuit*,GENinstance*,int,IFvalue*,IFvalue*);
extern int HEMTdelete(GENmodel*,IFuid,GENinstance**);
extern void HEMTdestroy(GENmodel**);
extern int HEMTgetic(GENmodel*,CKTcircuit*);
extern int HEMTload(GENmodel*,CKTcircuit*);
extern int HEMTmAsk(CKTcircuit*,GENmodel*,int,IFvalue*);
extern int HEMTmDelete(GENmodel**,IFuid,GENmodel*);
extern int HEMTmParam(int,IFvalue*,GENmodel*);
extern int HEMTparam(int,IFvalue*,GENinstance*,IFvalue*);
extern int HEMTpzLoad(GENmodel*,CKTcircuit*,SPcomplex*);
extern int HEMTsetup(SMPmatrix*,GENmodel*,CKTcircuit*,int*);
extern int HEMTtemp(GENmodel*,CKTcircuit*);
extern int HEMTtrunc(GENmodel*,CKTcircuit*,double*);
extern int HEMTdisto(int,GENmodel*,CKTcircuit*);
extern int HEMTnoise(int,int,GENmodel*,CKTcircuit*,Ndata*,double*);

#else /* stdc */
extern int HEMTacLoad();
extern int HEMTask();
extern int HEMTdelete();
extern void HEMTdestroy();
extern int HEMTgetic();
extern int HEMTload();
extern int HEMTmAsk();
extern int HEMTmDelete();
extern int HEMTmParam();
extern int HEMTparam();
extern int HEMTpzLoad();
extern int HEMTsetup();
extern int HEMTtemp();
extern int HEMTtrunc();
extern int HEMTdisto();
extern int HEMTnoise();
#endif /* stdc */
```

A.17 HEMTITF.H

```
/**********
Universidad de Las Palmas de Gran Canaria
Autor: 1997 F.J. del Pino
**********/
#ifdef DEV_hemt

#ifndef DEV_HEMT
#define DEV_HEMT

#include "hemtext.h"
extern IFparm HEMTpTable[ ];
extern IFparm HEMTmPTable[ ];
extern char *HEMTnames[ ];
extern int HEMTpTSize;
extern int HEMTmPTSize;
extern int HEMTnSize;
extern int HEMTiSize;
extern int HEMTmSize;

SPICEdev HEMTinfo = {
    {
        "Hemt",
        "GaAs HEMT model",

        &HEMTnSize,
        &HEMTnSize,
        HEMTnames,

        &HEMTpTSize,
        HEMTpTable,

        &HEMTmPTSize,
        HEMTmPTable,
    },
    HEMTparam,
    HEMTmParam,
    HEMTload,
    HEMTsetup,
    HEMTsetup,
    HEMTtemp,
    HEMTtrunc,
    NULL,
    HEMTacLoad,
    NULL,
    HEMTdestroy,
#ifdef DELETES
    HEMTmDelete,
    HEMTdelete,
#else /* DELETES */
    NULL,
    NULL,
#endif /* DELETES */
    HEMTgetic,
    HEMTask,
    HEMTmAsk,
#ifdef AN_pz
    HEMTpzLoad,
#else /* AN_pz */
    NULL,
#endif /* AN_pz */
    NULL,
```

```
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
#ifdef AN_disto
    HEMTdisto,
#else /* AN_disto */
    NULL,
#endif      /* AN_disto */
#ifdef AN_noise
    HEMTnoise,
#else /* AN_noise */
    NULL,
#endif      /* AN_noise */

    &HEMTiSize,
    &HEMTmSize

};

#endif
#endif
```

A.18 INPDOMOD.C

```
/**********
Copyright 1990 Regents of the University of California.  All rights reserved.
Author: 1985 Thomas L. Quarles
**********/

#include "spice.h"
#include <stdio.h>
#include "iferrmsg.h"
#include "util.h"
#include "inpdefs.h"
#include "strext.h"
#include "suffix.h"

/*ARGSUSED*/
char *
INPdomodel(ckt,image,tab)
    GENERIC *ckt;
    card *image;
    INPtables *tab;
{

    char *modname;
    int type;
    int lev;
    char *typename;
    char *err = (char *)NULL;
    char *line;

    line = image->line;
    INPgetTok(&line,&modname,1);  /* throw away '.model' */
    INPgetTok(&line,&modname,1);
    INPinsert(&modname,tab);
    INPgetTok(&line,&typename,1);
    if( (strcmp(typename,"npn") == 0)||(strcmp(typename,"pnp")==0) ) {
        type = INPtypelook("BJT");
        if(type < 0) {
            err = INPmkTemp("Device type BJT not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"d") == 0) {
        type = INPtypelook("Diode");
        if(type < 0) {
            err = INPmkTemp("Device type Diode not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if( (strcmp(typename,"njf") == 0) || (strcmp(typename,"pjf") == 0)){
        type = INPtypelook("JFET");
        if(type < 0) {
            err = INPmkTemp("Device type JFET not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if( (strcmp(typename,"nmf") == 0) || (strcmp(typename,"pmf")==0) ) {
        err = INPfindLev(line,&lev);
        switch(lev) {
            case 1:
            default:
            type = INPtypelook("MES");
              if(type < 0) {
                  err = INPmkTemp(
                      "Device type MES not available in this binary\n");
```

```
                }
                break;
/* modificacion para incluir al HEMT */
          case 2:
                type = INPtypelook("Hemt");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type HEMT not available in this binary\n");
                }
                break;
/* fin modificacion */
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"urc") == 0) {
        type = INPtypelook("URC");
        if(type < 0) {
            err = INPmkTemp("Device type URC not available in this binary\n");
        }
        INPmakeMod(modname,type,image);


    } else if( (strcmp(typename,"nmos")==0) || (strcmp(typename,"pmos")==0) ) {
        err = INPfindLev(line,&lev);
        switch(lev) {
            case 1:
            default:
                type = INPtypelook("Mos1");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type MOS1 not available in this binary\n");
                }
                break;
            case 2:
                type = INPtypelook("Mos2");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type MOS2 not available in this binary\n");
                }
                break;
            case 3:
                type = INPtypelook("Mos3");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type MOS3 not available in this binary\n");
                }
                break;
            case 4:
                type = INPtypelook("BSIM1");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type BSIM1 not available in this binary\n");
                }
                break;
            case 5:
                type = INPtypelook("BSIM2");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type BSIM2 not available in this binary\n");
                }
                break;
            case 6:
                type = INPtypelook("Mos6");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type MOS6 not available in this binary\n");
                }
```

```
                break;
            case 7:
                type = INPtypelook("Mos7");
                if(type < 0) {
                    err = INPmkTemp(
                            "Device type MOS7 not available in this binary\n");
                }
                break;
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"r") == 0) {
        type = INPtypelook("Resistor");
        if(type < 0) {
            err = INPmkTemp(
                    "Device type Resistor not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"c") == 0) {
        type = INPtypelook("Capacitor");
        if(type < 0) {
            err = INPmkTemp(
                    "Device type Capacitor not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"sw") == 0) {
        type = INPtypelook("Switch");
        if(type < 0) {
            err = INPmkTemp(
                    "Device type Switch not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"csw") == 0) {
        type = INPtypelook("CSwitch");
        if(type < 0) {
            err = INPmkTemp(
                    "Device type CSwitch not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else if(strcmp(typename,"ltra") == 0) {
        type = INPtypelook("LTRA");
        if(type < 0) {
            err = INPmkTemp(
                    "Device type LTRA not available in this binary\n");
        }
        INPmakeMod(modname,type,image);
    } else {
        type = -1;
        err = (char *)MALLOC(35 + strlen(typename));
        (void)sprintf(err,"unknown model type %s - ignored\n",typename);
    }
    return(err);
}
```

A.19 INPPAS2.C

```
/**********
Copyright 1990 Regents of the University of California.  All rights reserved.
Author: 1985 Thomas L. Quarles
**********/

#include "spice.h"
#include "strext.h"
#include <ctype.h>
#include "ifsim.h"
#include "cpdefs.h"
#include "fteext.h"
#include "ftedefs.h"
#include "inpdefs.h"
#include "util.h"
#include "iferrmsg.h"
#include "tskdefs.h"
#include "inpmacs.h"
#include "suffix.h"

/* pass 2 - Scan through the lines.  ".model" cards have processed
 *  in pass1 and are ignored here.
 */

void
INPpas2(ckt,data,tab,task)
    GENERIC *ckt;
    card *data;
    INPtables *tab;
    GENERIC *task;
{

card *current;
char c;
char * groundname="0";
char * gname;
GENERIC *gnode;
int error;  /* used by the macros defined above */

    error = INPgetTok(&groundname,&gname,1);
    if(error) data->error = INPerrCat(data->error,INPmkTemp(
            "can't read internal ground node name!\n"));

    error = INPgndInsert(ckt,&gname,tab,&gnode);
    if(error && error!=E_EXISTS)data->error = INPerrCat(data->error,INPmkTemp(
            "can't insert internal ground node in symbol table!\n"));

    for(current = data; current != NULL; current = current->nextcard) {

        c = *(current->line);
        c = islower(c) ? toupper(c) : c;

      switch(c) {

        case ' ':   /* blank line (space leading) */
        case '\t':   /* blank line (tab leading) */
            break;

        case 'R':   /* Rname <node> <node> [<val>][<mname>][w=<val>][l=<val>] */
            INP2R(ckt,tab,current);
            break;
        case 'C':   /* Cname <node> <node> <val> [IC=<val>] */
```

```
        INP2C(ckt,tab,current);
        break;
    case 'L':   /* Lname <node> <node> <val> [IC=<val>] */
        INP2L(ckt,tab,current);
        break;
    case 'G':   /* Gname <node> <node> <node> <node> <val> */
        INP2G(ckt,tab,current);
        break;
    case 'E':   /* Ename <node> <node> <node> <node> <val> */
        INP2E(ckt,tab,current);
        break;
    case 'F':   /* Fname <node> <node> <vname> <val> */
        INP2F(ckt,tab,current);
        break;
    case 'H':   /* Hname <node> <node> <vname> <val> */
        INP2H(ckt,tab,current);
        break;
    case 'D':   /* Dname <node> <node> <model> [<val>] [OFF] [IC=<val>] */
        INP2D(ckt,tab,current);
        break;
    case 'J':   /* Jname <node> <node> <node> <model> [<val>] [OFF]
                 *       [IC=<val>,<val>] */
        INP2J(ckt,tab,current);
        break;
    case 'Z':   /* Zname <node> <node> <node> <model> [<val>] [OFF]
                 *       [IC=<val>,<val>] */
        INP2Z(ckt,tab,current);
        break;
    case 'M':   /* Mname <node> <node> <node> <node> <model> [L=<val>]
                 *       [W=<val>] [AD=<val>] [AS=<val>] [PD=<val>]
                 *       [PS=<val>] [NRD=<val>] [NRS=<val>] [OFF]
                 *       [IC=<val>,<val>,<val>] */
        INP2M(ckt,tab,current);
        break;
case 'O':   /* Oname <node> <node> <node> <node> <model>
                    *       [IC=<val>,<val>,<val>,<val>] */
    INP2O(ckt,tab,current);
    break;

    case 'V':   /* Vname <node> <node> [ [DC] <val>] [AC [<val> [<val> ] ] ]
                 *       [<tran function>] */
        INP2V(ckt,tab,current);
        break;
    case 'I':   /* Iname <node> <node> [ [DC] <val>] [AC [<val> [<val> ] ] ]
                 *       [<tran function>] */
        INP2I(ckt,tab,current);
        break;

    case 'Q':   /* Qname <node> <node> <node> [<node>] <model> [<val>] [OFF]
                 *       [IC=<val>,<val>] */
        INP2Q(ckt,tab,current,gnode);
        break;

    case 'T':   /* Tname <node> <node> <node> <node> [TD=<val>]
                 *       [F=<val> [NL=<val>]][IC=<val>,<val>,<val>,<val>] */
        INP2T(ckt,tab,current);
        break;

    case 'S':   /* Sname <node> <node> <node> <node> [<modname>] [IC] */
        INP2S(ckt,tab,current);
        break;

    case 'W':   /* Wname <node> <node> <vctrl> [<modname>] [IC] */
        /* CURRENT CONTROLLED SWITCH */
        INP2W(ckt,tab,current);
        break;
```

```
        case 'U':   /* Uname <node> <node> <model> [l=<val>] [n=<val>] */
            INP2U(ckt,tab,current);
            break;

        case 'K':   /* Kname Lname Lname <val> */
            INP2K(ckt,tab,current);
            break;

        case '*':   /* *<anything> - a comment - ignore */
            break;

        case 'B':   /* Bname <node> <node> [V=expr] [I=expr] */
            /* Arbitrary source. */
            INP2B(ckt,tab,current);
            break;

        case '.':   /* .<something> Many possibilities */
            if (INP2dot(ckt,tab,current,task,gnode)) goto end;
            break;

      case 0:
          break;

        default:
            /* the un-implemented device */
            LITERR(" unknown device type - error \n")
            break;
        }
    }
end:
    return;
}
```

A.20 INP2Z.C

```
/**********
Copyright 1990 Regents of the University of California.  All rights reserved.
Author: 1988 Thomas L. Quarles
**********/

#include "spice.h"
#include <stdio.h>
#include "ifsim.h"
#include "inpdefs.h"
#include "inpmacs.h"
#include "fteext.h"
#include "suffix.h"

void
INP2Z(ckt,tab,current)
    GENERIC *ckt;
    INPtables *tab;
    card *current;

{

    /* Zname <node> <node> <node> <model> [<val>] [OFF] [IC=<val>,<val>] */

int mytype; /* the type we looked up */
int type;   /* the type the model says it is */
char *line; /* the part of the current line left to parse */
char *name; /* the resistor's name */
char *nname1;   /* the first node's name */
char *nname2;   /* the second node's name */
char *nname3;   /* the third node's name */
GENERIC *node1; /* the first node's node pointer */
GENERIC *node2; /* the second node's node pointer */
GENERIC *node3; /* the third node's node pointer */
int error;      /* error code temporary */
GENERIC *fast;  /* pointer to the actual instance */
IFvalue ptemp;  /* a value structure to package resistance into */
int waslead;    /* flag to indicate that funny unlabeled number was found */
double leadval; /* actual value of unlabeled number */
char *model;    /* the name of the model */
INPmodel *thismodel;    /* pointer to model description for user's model */
GENERIC *mdfast;    /* pointer to the actual model */
IFuid uid;      /* uid for default model */

    mytype = INPtypelook("MES");
    if(mytype < 0 ) {
        LITERR("Device type MES not supported by this binary\n")
        return;
    }
    line = current->line;
    INPgetTok(&line,&name,1);
    INPinsert(&name,tab);
    INPgetTok(&line,&nname1,1);
    INPtermInsert(ckt,&nname1,tab,&node1);
    INPgetTok(&line,&nname2,1);
    INPtermInsert(ckt,&nname2,tab,&node2);
    INPgetTok(&line,&nname3,1);
    INPtermInsert(ckt,&nname3,tab,&node3);
    INPgetTok(&line,&model,1);
    INPinsert(&model,tab);
    thismodel = (INPmodel *)NULL;
    current->error = INPgetMod(ckt,model,&thismodel,tab);
```

```
    if(thismodel != NULL){
        if( (thismodel->INPmodType != mytype) &&
              (thismodel->INPmodType != INPtypelook("Hemt") )) {
            LITERR("incorrect model type")
            return;
        }
        type = thismodel->INPmodType;
        mdfast = (thismodel->INPmodfast);
    } else {
        type = mytype;
        if(!tab->defZmod) {
            /* create default Z model */
            IFnewUid(ckt,&uid,(IFuid)NULL,"Z",UID_MODEL,(GENERIC**)NULL);
            IFC(newModel,(ckt,type,&(tab->defZmod),uid))
        }
        mdfast = tab->defZmod;
    }
    IFC(newInstance,(ckt,mdfast,&fast,name))
    IFC(bindNode,(ckt,fast,1,node1))
    IFC(bindNode,(ckt,fast,2,node2))
    IFC(bindNode,(ckt,fast,3,node3))
    PARSECALL((&line,ckt,type,fast,&leadval,&waslead,tab))
    if(waslead) {
        ptemp.rValue = leadval;
        GCA(INPpName,("area",&ptemp,ckt,type,fast))
    }
}
```

Apéndice B

Ficheros SPICE3

Listado de los ficheros de entrada a SPICE3 utilizados para la validación de los resultados obtenidos. En primer lugar se lista el fichero de entrada utilizado para evaluar las curvas características I_{DS} *vs.* V_{DS} de un HFET de enriquecimiento. Para obtener las curvas I_{DS} *vs.* V_{GS} basta con cambiar la línea:

```
.DC VDS 0 2.5 .1 VGS 0.1 0.8 0.1
```

por:

```
.DC VGS -1 1 0.01
```

Para el caso de un HFET de deplexión basta con cambiar la descripción del transistor.

El siguiente fichero que listamos corresponde a una cadena de inversores DCFL sobre la que se aplica un análisis transitorio.

En tercer lugar, listamos el fichero correspondiente al sumador *full-adder* descrito en el capítulo 7.

Por último, se presenta el fichero correspondiente al análisis en ac.

B.1 Fichero para Análisis de Curvas Características en dc

```
CARACTERISTICA DE SALIDA DE UN HEMT
.OPTIONS NODE NODEPAGE
VDS 3 0 0
VGS 2 0 0
Z1 1 2 0  EFET03 W=25
*VIDS MIDE ID, PODIAMOS HABER USADO VDS, PERO ID SALDRIA NEGATIVA
VIDS 3 1
.MODEL EFET03 NMF LEVEL=2
+       GEXP=0.075
+       AFACT=4.0
+       VC=0.65
+       VSB=5.0
+       LAMBDA=0.05
+       RD=600.0
+       RS=600.0
+       RG=0.2
```

```
+       MFACT=1.0
+       GAMMA=4.0
+       IS=1.0E-14
+       DXL=10.0
+       VS=0.4
+       CDVC=0.23E-3
+       CDVSB=0.0
+       DELTA=0.0
+       NP=1.5
+       BETA=2.8
+       ALPHA=5.0
+       FC=1.5
+       VST=1.0
+       CGS0=2.5E-16
+       CGS1=9.0E-16
+       VAT=-0.3
+       VBT=0.3
+       QC1=0
+       RGS=0.0
+       RGD=12k
+       TD=3.0E-12

.DC VDS 0 2.5 .1 VGS 0.1 0.8 0.1
.END
```

B.2 Fichero Correspondiente a una Cadena de Inversores DCFL

```
CADENA DE 16 INVERSORES

.MODEL EFET03 NMF LEVEL=2
+       GEXP=0.075
+       AFACT=4.0
+       VC=0.65
+       VSB=5.0
+       LAMBDA=0.05
+       RD=600.0
+       RS=600.0
+       RG=0.2
+       MFACT=1.0
+       GAMMA=4.0
+       IS=1.0E-14
+       DXL=10.0
+       VS=0.4
+       CDVC=0.23E-3
+       CDVSB=0.0
+       DELTA=0.0
+       NP=1.5
+       BETA=2.8
+       ALPHA=5.0
+       FC=1.5
+       VAT=-0.3
+       VBT=0.3
+       QC1=0.0
+       VST=1.0
+       CGS0=2.5E-16
```

```
+       CGS1=9.0E-16
+       RGS=800.0
+       RGD=800.0
+       CDS=2.0E-16

.MODEL DFET03  NMF LEVEL=2
+       GEXP=0.075
+       AFACT=5.0
+       VC=0.05
+       VSB=0.78
+       LAMBDA=0.09
+       RD=500.0
+       RS=500.0
+       RG=0.2
+       MFACT=1.0
+       GAMMA=2.5
+       IS=1.0E-14
+       DXL=3.0
+       VS=0.4
+       CDVC=0.145E-3
+       CDVSB=0.12E-3
+       DELTA=1.9
+       NP=1.35
+       BETA=2.3
+       ALPHA=3.1
+       FC=0.42
+       VAT=-0.9
+       VBT=-0.4
+       QC1=0.0
+       VST=1.0
+       CGS0=2.3E-16
+       CGS1=6.5E-16
+       RGS=900.0
+       RGD=900.0
+       CDS=2.0E-16

.TRAN 20PS 1NS

VIN 3 0 PULSE(0 1 100P 100P 100P 400P 1N)
VDD 100 0 1.5

.SUBCKT INVERSOR 10 1 2
Z2 10 2 2 DFET03  W=5
Z1 2 1 0 EFET03  W=10
.ENDS INVERSOR

X1 100 3 4 INVERSOR
X2 100 4 5 INVERSOR
X3 100 5 6 INVERSOR
X4 100 6 7 INVERSOR
X5 100 7 8 INVERSOR
X6 100 8 9 INVERSOR
X7 100 9 10 INVERSOR
X8 100 10 11 INVERSOR
X9 100 11 12 INVERSOR
X10 100 12 13 INVERSOR
X11 100 13 14 INVERSOR
X12 100 14 15 INVERSOR
```

```
X13 100 15 16 INVERSOR
X14 100 16 17 INVERSOR
X15 100 17 18 INVERSOR
X16 100 18 19 INVERSOR

.END
```

B.3 Fichero Correspondiente al *full-adder*

```
****************************************
* Spice template control file          *
* Simulation file for Spice, using     *
* HEMT 0.3um technology.               *
****************************************

******** Include netlist ************
* net 1 = vdd!
* net 0 = gnd!
* net 9 = /Sout
* net 10 = /Si
* net 11 = /Ci-1
* net 12 = /Ci
* net 13 = /Cout

VDD 1 0 PULSE  0 1.5 10.e-12 100.e-12 1. 1. 1.
.TRAN  10.e-12 10.0e-9
.MODEL EFET03 NMF LEVEL=2
+       GEXP=0.075
+       AFACT=4.0
+       VC=0.65
+       VSB=5.0
+       LAMBDA=0.05
+       RD=600.0
+       RS=600.0
+       RG=0.2
+       MFACT=1.0
+       GAMMA=4.0
+       IS=1.0E-14
+       DXL=10.0
+       VS=0.4
+       CDVC=0.23E-3
+       CDVSB=0.0
+       DELTA=0.0
+       NP=1.5
+       BETA=2.8
+       ALPHA=5.0
+       FC=1.5
+       VAT=-0.3
+       VBT=0.3
+       QC1=0.0
+       VST=1.0
+       CGS0=2.5E-16
+       CGS1=9.0E-16
+       RGS=0.0
+       RGD=0.0
+       CDS=10.0E-16
```

```
Z2 3 6 0 EFET03 w=10
.MODEL DFET03  NMF LEVEL=2
+       GEXP=0.075
+       AFACT=5.0
+       VC=0.05
+       VSB=0.78
+       LAMBDA=0.09
+       RD=500.0
+       RS=500.0
+       RG=0.2
+       MFACT=1.0
+       GAMMA=2.5
+       IS=1.0E-14
+       DXL=3.0
+       VS=0.4
+       CDVC=0.145E-3
+       CDVSB=0.12E-3
+       DELTA=1.9
+       NP=1.35
+       BETA=2.3
+       ALPHA=3.1
+       FC=0.42
+       VAT=-0.9
+       VBT=-0.4
+       QC1=0.0
+       VST=1.0
+       CGS0=2.3E-16
+       CGS1=6.5E-16
+       RGS=0.0
+       RGD=0.0
+       CDS=10.0E-16

Z3 1 3 3 DFET03 w=5
Z4 6 13 0 EFET03 w=10
Z5 1 6 6 DFET03 w=5
Z6 4 15 0 EFET03 w=10
Z7 1 4 4 DFET03 w=5
Z8 15 9 0 EFET03 w=10
Z9 1 15 15 DFET03 w=5
Z10 8 7 0 EFET03 w=10
Z11 1 8 8 DFET03 w=5
Z12 2 5 0 EFET03 w=10
Z13 1 2 2 DFET03 w=5
Z14 7 13 0 EFET03 w=10
Z15 1 7 7 DFET03 w=5
Z16 5 9 0 EFET03 w=10
Z17 1 5 5 DFET03 w=5
VIN18 33 0 PULSE  0 0.8 150ps 25ps 25ps 0.475ns 1ns
Z19 10 34 0 EFET03 w=12
Z20 1 10 10 DFET03 w=6
Z21 34 33 0 EFET03 w=12
Z22 1 34 34 DFET03 w=6
VIN23 40 0 PULSE  0 0.8 150ps 25ps 25ps 0.975ns 2ns
Z24 12 41 0 EFET03 w=12
Z25 1 12 12 DFET03 w=6
Z26 41 40 0 EFET03 w=12
Z27 1 41 41 DFET03 w=6
```

```
VIN28 47 0 PULSE  0 0.8 150ps 25ps 25ps 1.975ns 4ns
Z29 11 48 0 EFET03 w=12
Z30 1 11 11 DFET03 w=6
Z31 48 47 0 EFET03 w=12
Z32 1 48 48 DFET03 w=6
Z33 55 53 0 EFET03 w=12
Z34 1 55 55 DFET03 w=6
Z35 58 12 0 EFET03 w=10
Z36 1 58 58 DFET03 w=5
Z37 62 10 0 EFET03 w=10
Z38 1 62 62 DFET03 w=5
Z39 61 11 0 EFET03 w=12
Z40 1 61 61 DFET03 w=6
Z41 1 73 13 EFET03 w=10
Z42 1 72 13 EFET03 w=10
Z43 73 62 0 EFET03 w=11
Z44 72 55 0 EFET03 w=11
Z45 73 58 0 EFET03 w=11
Z46 72 61 0 EFET03 w=11
Z47 13 0 0 DFET03 w=7
Z48 1 72 72 DFET03 w=8
Z49 1 73 73 DFET03 w=8
Z50 1 80 9 EFET03 w=10
Z51 1 79 9 EFET03 w=10
Z52 80 11 0 EFET03 w=11
Z53 79 53 0 EFET03 w=11
Z54 80 55 0 EFET03 w=11
Z55 79 61 0 EFET03 w=11
Z56 9 0 0 DFET03 w=7
Z57 1 79 79 DFET03 w=8
Z58 1 80 80 DFET03 w=8
Z59 1 87 53 EFET03 w=10
Z60 1 86 53 EFET03 w=10
Z61 87 12 0 EFET03 w=11
Z62 86 10 0 EFET03 w=11
Z63 87 62 0 EFET03 w=11
Z64 86 58 0 EFET03 w=11
Z65 53 0 0 DFET03 w=7
Z66 1 86 86 DFET03 w=8
Z67 1 87 87 DFET03 w=8

.end
```

B.4 Fichero Correspondiente al Análisis en ac

```
ANALISIS AC DE UN HEMT

VDD 1 0 3
RD 1 2 1k
Z1 2 3 0  MOD1 w=25
VGS 3 0 0 AC 1
VMEAS 4 2
CSHUNT 4 0 0.1u

.MODEL MOD1 NMF LEVEL=2
+         GEXP=0.075
```

```
+       AFACT=4.0
+       VC=0.65
+       VSB=5.0
+       LAMBDA=0.05
+       RD=600.0
+       RS=600.0
+       RG=0.2
+       MFACT=1.0
+       GAMMA=4.0
+       IS=1.0E-14
+       DXL=10.0
+       VS=0.4
+       CDVC=0.23E-3
+       CDVSB=0.0
+       DELTA=0.0
+       NP=1.5
+       BETA=2.8
+       ALPHA=5.0
+       FC=1.5
+       VST=1.0
+       CGS0=2.5E-16
+       CGS1=9.0E-16
+       VAT=-0.3
+       VBT=0.3
+       QC1=0.0
+       CDS=2.0E-16
+       RGS=800.0
+       RGD=800.0
+       TD=3.0E-12

.AC DEC 20 0.05G 25G
.END
```

Apéndice C

Presupuesto

C.1 Costes de Amortización de Equipos y Herramientas

Los equipos y paquetes software utilizados para la elaboración del Proyecto que se presenta conllevan unos costes de amortización y mantenimiento que deben ser incluidos en el presupuesto en función del periodo de uso.

Descripción	*Periodo de uso*	*Valor/Año*	*Total*
Sistema operativo SunOS Release 4.1.3, sistema de ventanas Open Windows y Librerías X11.	1 año	43.333 pts.	43.333 pts.
Simulador IAFSPICE (Curso Eurochip Design Course)	1 año	190.000 pts.	190.000 pts.
SOFTWARE		TOTAL	233.333 pts.

Descripción	*Periodo de uso*	*Valor/Año*	*Total*
Estación de trabajo Sun SPARC modelo SPARCstation 10. Con periodo de amortización de 3 años			
Amortización	1 año	870.000 pts.	870.000 pts.
Mantenimiento	1 año	262.000 pts.	262.000 pts.
HARDWARE		TOTAL	1.132.000 pts.

C.2 Costes de Personal

El Proyecto se ha realizado en 12 meses con la participación de un Ingeniero Junior. El presupuesto final, en lo que respecta a personal, queda reflejado en la siguiente tabla:

Descripción de tarea	*Duración*	*hombres/mes*	*Total*
Especificación	2 meses	250.000 pts.	500.000 pts.
Realización de Software	7 meses	250.000 pts.	1.750.000 pts.
Preparación de documento	3 meses	250.000 pts.	750.000 pts.
MANO DE OBRA		TOTAL	3.000.000 pts.

C.3 Gastos en Bibliografía

En este apartado se incluye los gastos derivados de la adquisición de la bibliografía necesaria para la realización de este Proyecto.

Referencia	*Unidades*	*Precio/Unidad*	*Total*
[M89/42]	1	2.042 pts.	2.042 pts.
[M89/43]	1	780 pts.	780 pts.
[M89/44]	1	1.832 pts.	1.832 pts.
[M89/42]	1	901 pts.	901 pts
Gastos de tramitación y envío	1	3.755 pts.	3.755 pts.
[VLADI94]	1	9.850 pts.	12.857 pts.
BIBLIOGRAFÍA		TOTAL	22.167 pts.

C.4 Otros

En este apartado se incluyen los gastos de material fungible durante los 12 meses de duración del Proyecto.

Descripción	*Unidades*	*Precio/Unidad*	*Total*
Papel	15	500 pts.	7.500 pts.
Accesorios Impresora láser	15	1.000 pts.	15.000 pts.
Cintas de back-up y diskettes	2	5.000 pts.	10.000 pts.
MATERIAL FUNGIBLE		TOTAL	32.500 pts.

C.5 Gastos Totales

El presupuesto total del Proyecto asciende a un total de CUATRO MILLONES CUATROCIENTAS VEINTE MIL pesetas, distribuidas tal y como se muestra en la siguiente tabla:

Apartado	*Coste*
▷Software	233.333 pts.
▷Hardware	1.132.000 pts.
▷Costes de Personal	3.000.000 pts.
▷Bibliografía	22.167 pts.
▷Material Fungible	32.500 pts.
TOTAL	4.420.000 pts.

REFERENCIAS BIBLIOGRÁFICAS

Referencias Bibliográficas

Capítulo 1

[JQNPS91] B. Johnson, T. Quarles, A.R. Newton, D.O. Pederson, A. Sangiovanni-Vicentelli, "SPICE3 Version 3e User's Manual", Departament of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, Abril 1991.

Capítulo 2

[LONBU90] Stephen L. Long, Steven E. Butner, "Galium Arsenide Digital Circuit Design", McGraw-Hill. 1990.

[AFKHM84] Arnold,D.J.,Fischer,R.,Kopp,W.F.,Henderson,T.S.,y H. Morkoc: "Microwave Characterization of (AL,Ga)As/GaAs Modulation-Doped FETs: Bias Dependence of Small-Signal Parameters", *IEEE Trans. Elect. Dev.*, ED-31: 1399-1402, Octubre 1984.

[SMSZE81] S.M.Sze, "Physics of Semiconductor Devices (2nd. Edition)", John Wiley & Sons. 1981.

[SMSZE90] S.M.Sze, "High-Speed Semiconductor Devices", John Wiley & Sons. 1990.

[MAEAS96] Martel S., Esper-Chaín R., de Armas V., Sarmiento R., "Implementación de un conmutador 2X2 para redes de ATM hasta 1Gb/s con tecnología HEMT", *DCIS 96*, p. 35, Noviembre 1996.

[SKOLN88] Merril I. Skolnik, "Introduction to Radar Systems", McGraw-Hill International Editions, 1988.

Capítulo 3

[SCNRR94] D. Schreurs, B. Nauwelaers, W. De Raedt y M.Van Rossum, "Requirements of a large-signal HEMT model with regard to non-linear MMIC design", *Procedings of the European Gallium Arsenide and Related III-V Compounds Applications Symposium*, GAAS 94, IEEE, Abril 1994.

[CURTI80] W. Curtice, "A MESFET model for use in the design of GaAs integrated circuits", *IEEE Trans. Microwave Theory Tech.*, MTT-28: 448-455, 1980.

[CURET85] W. Curtice y M. Ettenberg, " A nonlinear GaAs FET model for use in the design of output circuits for power amplifiers", *IEEE Trans. Microwave Theory Tech.*, MTT-33: 1383, 1985.

[MATKA85] A. Materka y T. Kacprzak, "Computer calculation of large-signal GaAs FET amplifier characteristics", *IEEE Trans. Microwave Theory Tech.*, MTT-33: 129-134, 1985.

[ANGEL92] I. Angelov *et al.*, "A new empirical nonlinear model for HEMT and MESFET devices", *IEEE Trans. Microwave Theory Tech.*, MTT-40: 2258-2266, 1992.

[STANE87] H. Statz, P. Newman *et al.*, "GaAs FET device and circuit simulation in SPICE", *IEEE Trans. Electron Devices*, ED-34: 160-166, 1987.

[TAWRM81] Y. Tajima, B. Wrona, y K. Mishima, "GaAs FET large-signal model and its application to circuit design", *IEEE Trans. Electron Devices*, ED-28: 171, 1981.

[MAANE90] S. Maas y D. Neilson, "Modeling of MESFETs for intermodulation analysis of mixers amplifiers", in 1990 *IEEE MTT-S Microwave Symp. Dig.*, pp. 1291-1294, 1990.

[HOBLC86] R. Holmstrom, W. Bloss, y J.Chi, "A gate probe method of determining parasitic resistance in MESFETs", *IEEE Electron Devices Lett.*, EDL-7: 410-412, 1986.

[ROKAM90] P. Roblin, S. Kang, y H. Morkoc, "Analytic solution of the velocity-saturated MOSFET/MODFET wave equation and its application to the prediction of the microwave characteristics of MODFETs", *IEEE Trans. Electron Devices*, ED-37, 1990.

[AKFKM84] D. Arnold, W. Kopp, R. Fisher, J. Klem, y H. Morkoc, "Bias dependence of capacitances in MODFET at 4 GHz", *IEEE Electron Devices Lett.*, EDL-5: 123, 1984.

[EUROC94] "EUROCHIP Design Course", *FhGIAF*, IAF Design Manual Versión 3.0, 1994.

[M89/42] T. L. Quarles, "Analysis of Perfomance and Convergence Issues for Circuits Simulation", Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, Memo No. UCB/ERL M89/42, 1989.

Capítulo 4

[MCCAL88] W. J. McCalla, "Fundamentals of Computer-Aided Circuit Simulation", Boston: Kluwer Academic,1988.

[NAGEL75] L. W. Nagel, "SPTCE2: A computer program to simulate semiconductor circuits", Univ. of California, Berkeley, ERL Memo No. ERL M520, Mayo 1975.

[MCCPE71] W. J. McCalla, y D. O. Pederson, "Elements of computer-aided circuit analysis", *IEEE Transactions on Circuit Theoq* CT-18, Enero 1971.

[HACSA81] G. Hachtel, y A. L. Sangiovanni-Vincentelli, "A survey of third-generation simulation techniques", *IEEE Proceedings* 69,Octubre 1981.

[DORF89] R. C. Dorf, "Introduction to Electric Circuits", New York: John Wiley & Sons, 1989.

[NILSS90] J. W. Nilsson, "Electric Circuits", 3d ed. Reading, MA: Addison-Wesley, 1990.

[PAUL89] Paul C. R. "Analysis of Linear Circuits", New York: McGraw-Hill, 1989.

[NAGRO71] L. W. Nagel, y R. Rohrer, "Computer analysis of nonlinear circuits, excluding radiation (CANCER)", *IEEE Journal of Solid-State Circuits,* SC-6: 166-182, Agosto 1971.

[FORMO67] G.E. Forsythe, y C. B. Moler, "Computer Solution of Linear Algebraic Systems", Englewood Cliffs, NJ: Prentice Rail, 1967.

[HORUB75] C. W. Ho, A. E. Ruehli, y P. A. Brennan, "The modified nodal approach to network analysis", *IEEE Transactions on Circuits and Systems,* CAS-22: 504-509, Junio 1975.

[COHEN81] E. Cohen, "Performance limits of integrated circuits simulation on a dedicated minicomputer system", Univ. Of California, Berkeley, ERL Memo No. UCB/ERL M81/29, Mayo 1981.

[HAYAT81] I. N. Hajj, P. Yang, y T. N. Trick, "Avoiding zero pivots in the modified nodal approach", *IEEE Transactions on Circuits and Systems,* CAS-28: 271-278, Abril 1981.

[FRERE76] J. P. Freret, "Minicomputer Calculation of the DC Operating Point of Bipolar Circuits", Technical Report No. 5015-l, Stanford Electronics Labs., Stanford Univ., Stanford, CA., Mayo 1976.

[VLADI94] A. Vladimirescu, "The SPICE Book", New York: John Wiley & Sons, 1994.

[ORTRE70] J. M. Ortega, y W. R. Rheinholdt, "Alterative Solution of Non-Linear Equations in Several Variables", New York: Academic Press, 1970.

[CALAH72] D. A. Calahan, "Computer-Aided Network Design", New York: McGraw-Hill, 1972.

[CHULI75] L. O. Chua, y P. M Lin, "Computer Aided Analysis of Electronic Circuits: Algorithms and Computational Technianes", Englewood Cliffs, NJ: 1975.

[GEAR67] C. W. Gear, "Numerical integration of stiff ordinary equations", Report 221, Dept. of Computer Science, Univ. of Illinois, Urbana, 1967.

Capítulo 5

[M89/44] T. L. Quarles, "The SPICE3 Implementation Guide", Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, Memo No. UCB/ERL M89/44, 1989.

[GUMPO70] H. K. Gummel, y H. C. Poon, "An Integral Charge Control Model of Bipolar Transistors", *Bell Sys. Tech. J.*, 49: 827-852, Mayo 1970.

[EBEMO54] J.J. Ebers y J.L. Moll, "Large-Signal Behavior of Junction Transistors", *Proc. IRE*, 42: 1761, Diciembre 1954.

[SHIHO68] H. Shichman, y D. A. Hodges, "Modeling and simulation of insulated-gate field effect transistor switching circuits", *IEEE Journal of Solid-State Circuits,* SC-3: 285-289, Septiembre 1968.

[STATZ87] H.Statz et al., "GaAs FET Device and Circuit Simulation in SPICE", *IEEE Transactions on Electron Devices,* ED-34: 160-169, Febrero 1987.

[VLALI80] A. Vladimirescu, y S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2", ERL Memo No. ERL M80/7, Electronics Research Laboratory, University of California, Berkeley, Octubre 1980.

[SAKNE90] T. Sakurai, y A. R. Newton, "A Simple MOSFET Model for Circuit Analysis and its application to CMOS gate delay analysis and series-connected MOSFET Structure", ERL Memo No. ERL M90/19, Electronics Research Laboratory, University of California, Berkeley, Marzo 1990.

[SHSCK85] B. J. Sheu, D. L. Scharfetter, y P. K. Ko, "SPICE2 Implementation of BSIM", ERL Memo No. ERL M85/42, Electronics Research Laboratory University of California. Berkeley, Mayo 1985.

[MINCH90] Min-Chie Jeng, " Design and Modeling of Deep-Submicrometer MOSFETSs" ERL Memo No. ERL M90/90, Electronics Research Laboratory University of California, Berkeley, Octubre 1990.

[ROYPE91] J.S. Roychowdhury, y D.O. Pederson, "Efficient Transient Simulation of Lossy Interconnect", ACM/IEEE Design Automation Conference, San Francisco, Junio 1991.

Capítulo 6

[PSPIC90] "PSPICE Circuit Analysis", Versión 4.04, MicroSim Corporation, California, 1990.

[M89/45] T. L. Quarles, "Adding Devices to SPICE3", Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, Memo No. UCB/ERL M89/45, 1989.