DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Simulación Lógica Temporal de Altas Prestaciones y aplicación a la estimación del consumo de Potencia y Corriente en circuitos integrados CMOS-VLSI

Memoria presentada para optar al grado de Doctor por

Paulino Ruiz de Clavijo Vázquez

Ingeniero en Informática

Sevilla, Junio de 2007

Simulación Lógica Temporal de Altas Prestaciones y aplicación a la estimación del consumo de Potencia y Corriente en circuitos integrados CMOS-VLSI

Memoria presentada por

Paulino Ruiz de Clavijo Vázquez
para optar al grado de Doctor

Los Directores

Manuel Jesús Bellido Díaz
Profesor Titular de Universidad

Jorge Juan Chico
Profesor Contratado Doctor

Departamento de Tecnología Electrónica
Universidad de Sevilla

Índice de contenido

Prefacio

En las últimas dos décadas, en la sociedad ha acontecido una revolución digital que ha afectado principalmente al campo de las comunicaciones, abarcando áreas muy diversas, como sistemas de control, navegación, electrónica de consumo, etc. Esta revolución ha sido propiciada por el incesante incremento en la complejidad de los sistemas digitales y soportada por la paralela evolución en la tecnología VLSI, donde, al incrementarse el número total de componentes que caben en un único *chip*, supone una reducción del tamaño y el coste de los circuitos integrados.

El aumento en la complejidad de los sistemas digitales desemboca en la necesidad de utilizar metodologías de desarrollo jerárquicas y, además, disponer de herramientas de diseño y verificación en los sucesivos estratos de la jerarquía. Las herramientas de verificación son fundamentales, no sólo para chequear un diseño, sino también para poder evaluar comparativamente diferentes alternativas y seleccionar la de mejor rendimiento para una aplicación concreta. Así, adquieren especial relevancia la concepción de algoritmos de simulación que permitan emular el comportamiento de los diseños con la máxima precisión posible.

La evolución tecnológica permite un aumento muy significativo de la complejidad de los sistemas y afecta tanto a la precisión como a la capacidad de utilización de las herramientas CAD, surgiendo la necesidad de mejorarlas o incluso de limitar su uso a determinadas partes del sistema. Concretamente, algunas herramientas de verificación, por ejemplo, los simuladores eléctricos, son inviables aplicados a un sistema digital en su conjunto. Como alternativa nos encontramos la simulación lógica-temporal, que no es mas que una simulación guiada por eventos (*event-driven*).

La simulación discreta guiada por eventos consiste en una colección de técnicas que, aplicadas al estudio de un sistema dinámico discreto, genera secuencias que caracterizan su comportamiento. El objetivo de estas técnicas es reducir el coste de tiempo en la obtención de resultados en aquellos sistemas donde su complejidad raramente permite aplicar soluciones analíticas en tiempos razonables. La clave para el éxito en el uso de estas técnicas reside en el grado de conocimiento sobre el sistema a modelar, adquiriendo vital importancia tareas de investigación sobre el comportamiento del sistema real, en nuestro caso los circuitos digitales VLSI.

Aplicada esta técnica a la simulación lógica-temporal de circuitos VLSI, el

modelo de comportamiento empleado en estas herramientas determina la calidad de la misma en el sentido de la precisión de los resultados obtenidos. Precisamente el antecedente del que surge este trabajo de Tesis Doctoral está en el desarrollo de un modelo de comportamiento temporal de puertas lógicas denominado IDDM (*Inertial Degradation Delay Model*) [BELL01, JUAN00a] que incluye el denominado efecto de degradación del retraso de propagación y un nuevo algoritmo para el tratamiento del efecto inercial.

Así pues, con este antecedente el primer objetivo planteado para esta Tesis es desarrollar un simulador lógico que incluya el modelo IDDM y que permita simular circuitos digitales a nivel de puertas lógicas. Este simulador debe ser compatible con formatos estándar de entrada y salida siendo así fácilmente integrable en entornos profesionales de diseño de circuitos integrados. En este trabajo no sólo se desarrolla el simulador lógico sino que se aprovechan las características del modelo en cuanto a la precisión en las formas de onda de las señales y en la propagación de los *glitches* para buscar nuevas aplicaciones a la propia herramienta de simulación. En concreto, una gran precisión en el tratamiento de los *glitches* da lugar a una medida muy exacta de la actividad de conmutación de los circuitos. Este parámetro, la actividad de conmutación, es utilizado por los diseñadores para realizar estimaciones del consumo de potencia y poder comparar diversos diseños entre si en cuanto a la energía consumida. Con esta idea, en este trabajo se plantea aplicar la simulación lógica temporal no sólo al análisis funcional y temporal de los circuitos, sino también a la estimación del consumo tanto de potencia como de intensidad, convirtiéndose esta idea en el segundo objetivo principal de esta Tesis. Para ello, se desarrolla un modelo de consumo de intensidad fácilmente integrable con la herramienta de simulación y que, junto a las características del modelo IDDM, obtiene resultados muy precisos en la medida de la intensidad y potencia.

El desarrollo de estos objetivos ha dado lugar a la herramienta que denominamos HALOTIS (*High Accuracy LOgic TIming Simulator*) [RUIZ01a, RUIZ05b]. HALOTIS incluye IDDM y el nuevo modelo de intensidad y, aunque las siglas HALOTIS hacen referencia a un simulador, a lo largo del documento se comprenderá que HALOTIS es algo mas que un simulador lógico, realmente es un entorno de simulación formando por diversas herramientas, entre las cuales, están el motor de simulación, componente principal de HALOTIS que incluye los nuevos algoritmos necesarios para aplicar correctamente el modelo IDDM, la librería de celdas para poder adaptar *kits* tecnológicos con facilidad a la herramienta, *parsers* para el código HDL de entrada como VERILOG, visualizadores de formas de onda y otras.

Con este planteamiento general del trabajo desarrollado, la Tesis se ha organizado en seis capítulos con el siguiente contenido:

- El primer capítulo está dedicado a la simulación y verificación temporal de circuitos VLSI en general, así como los fundamentos en los que se basan los simuladores lógico-temporales.
- En el segundo capítulo se realiza una revisión de los modelos de retraso, donde se recogen el efecto de degradación y el efecto inercial, proponiéndose un modelo para ser implementado en HALOTIS.
- El tercer capítulo presenta un nuevo modelo de intensidad que, apoyado en el modelo IDDM, es capaz de obtener la evolución temporal de la intensidad durante una simulación digital.
- En el cuarto capítulo se hace un recorrido completo del entorno de simulación lógico-temporal HALOTIS. De forma exhaustiva, se presentan todos los componentes utilizando la metodología UML y detallándose los aspectos más relevantes de cada uno.
- El quinto capítulo presenta resultados significativos obtenidos con HALOTIS. Los resultados son de tres tipos: lógicos, de intensidad y del propio rendimiento del entorno de simulación.
- En el sexto y último capítulo se han destacado las conclusiones más significativas de este trabajo de Tesis.

Capítulo 1. Introducción a la simulación lógica temporal de Circuitos Integrados Digitales

La evolución en la tecnología de circuitos VLSI da lugar a un aumento en la escala de integración por dos factores: la reducción de las dimensiones mínimas de los transistores y el aumento en el área máxima del chip. Este factor permite a los diseñadores aumentar la complejidad de los sistemas, aumentando a su vez la complejidad de las tareas de diseño y verificación. En este capítulo se hace una introducción al tema al que está dedicada esta Tesis Doctoral, la simulación lógica de circuitos integrados digitales VLSI.

El capítulo se basa en diversos trabajos [ACOS00, JUAN00a y BELL06], en los que se presentan los conceptos propios de la simulación lógica-temporal y donde se cubren desde el modelado del comportamiento de los componentes de los circuitos integrados digitales, hasta las propias técnicas informáticas de

implementación de esas herramientas denominadas simuladores lógico-temporales.

La organización del capítulo es el siguiente: comienza con una introducción al proceso de diseño de sistemas digitales para, en la segunda sección, centrarse en la verificación temporal y, sus tipos. La tercera sección hace un análisis de la simulación lógica en general y, más detalladamente, en la simulación lógica temporal. Para finalizar, la última sección se dedica a estudiar las principales características de la simulación guiada por eventos.

1.1. Introducción

Para manejar eficientemente la complejidad de un diseño integrado actualmente es necesario una buena metodología [WOLF94]. Por metodología se entiende el conjunto ordenado de acciones que deben llevarse a cabo para realizar con éxito una tarea compleja. En el caso de circuitos VLSI las metodologías se basan en el empleo de una jerarquía en el proceso de diseño. Las técnicas jerárquicas permiten manejar problemas de alta complejidad tanto técnicos como de organización. Básicamente, la jerarquización consiste en establecer un conjunto de niveles estratificados, de forma que los problemas de un nivel se descomponen en problemas de menor complejidad en el nivel inferior, que son mas manejables. A su vez, cada uno de estos problemas puede ser tratado independientemente del resto y, además, subdividido en problemas aún más simples, pero aumentando el nivel de detalle en la descripción del circuito. La subdivisión continúa hasta que se alcanza el nivel más bajo de la jerarquía.

Se conoce con el nombre de método *top-down* (arriba-abajo) a la forma de trabajar que, partiendo de las especificaciones del sistema, va bajando a sucesivos niveles siguiendo la jerarquía establecida en la metodología. En este método, el diseño realizado en cada nivel sirve de especificaciones para el nivel inferior. Usando estas especificaciones se procede a diseñar en el siguiente nivel.

En la figura 1.1 se muestra el método *top-down* típico aplicado al diseño de circuitos integrados digitales. En él se parte de las especificaciones del sistema y se recorren los niveles de transferencia entre registros y de conmutación, hasta alcanzar el nivel físico o geométrico que proporciona el *layout* del circuito integrado digital que implementa el sistema. Pasar de un nivel al siguiente implica la realización de dos tareas básicas: el diseño o síntesis de un circuito, y su verificación.

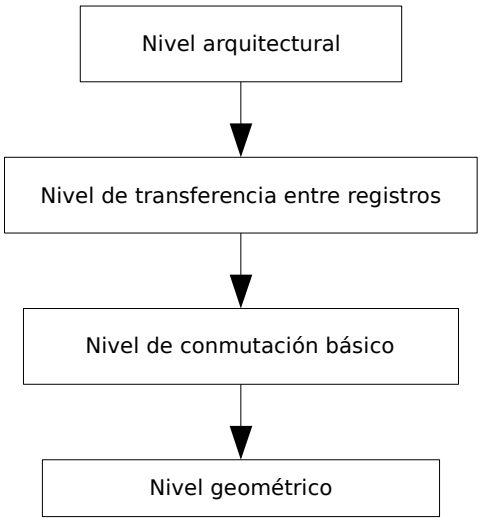


Figura 1.1. Método top-down de diseño de circuitos integrados digitales.

Para desarrollar adecuadamente el método *top-down* es necesario utilizar un entorno de diseño que consta, fundamentalmente, de herramientas CAD de dos tipos: (1) de síntesis, para llevar a cabo los diseños en los diferentes niveles y; (2) de verificación para comprobar la validez y eficiencia del diseño realizado.

Con respecto a la verificación, en cada uno de los niveles se realiza con técnicas y herramientas diferentes, como son, simuladores eléctricos para el nivel eléctrico, simuladores lógicos ó lógico-temporales en el nivel de puertas, llegando al nivel de comportamiento, donde se utilizan herramientas de análisis matemático de alto nivel.

Como ya se mencionó en el prefacio de la Tesis, este trabajo está dedicado a la simulación lógica temporal a nivel de puertas. Este tipo de herramientas sirve para realizar una verificación, no sólo funcional, sino también del comportamiento temporal del diseño, por tanto, el resto del capítulo está dedicado presentar cuales son las características principales de estas herramientas.

1.2. Introducción a la verificación temporal

El desarrollo de un circuito integrado digital VLSI está dividido en dos fases claramente diferenciadas que son el proceso de diseño y el de fabricación. Debido al alto coste de la fabricación, generalmente no se va a realizar una modificación en el diseño para mejorar las prestaciones a partir de los resultados del testado del *chip* ya implementado. De esta forma a la finalización del proceso de diseño se

debe garantizar, en la medida de lo posible, que el producto final va a cumplir los requisitos mínimos exigidos. Por ello resulta sumamente importante realizar una verificación del diseño lo más exhaustiva y precisa posible.

En las fases iniciales del proceso de diseño (nivel arquitectural y nivel de transferencia entre registros) la verificación debe ser, fundamentalmente, de carácter funcional. Esto es, debe servir para comprobar cómo el diseño realizado opera de acuerdo con las especificaciones iniciales de su comportamiento. En cambio, en las fases finales (nivel de puertas y geométrico) además de la verificación funcional hay que realizar una verificación temporal. Esto es, se debe realizar un análisis del diseño teniendo en cuenta el comportamiento dinámico de los distintos componentes que constituyen el circuito diseñado. Este análisis sirve para evaluar el rendimiento final del *chip,* en cuanto a velocidad máxima de operación y, además, se pueden detectar errores en el comportamiento del circuito causados por los retrasos de propagación.

La verificación se realiza empleando herramientas software que permitan estudiar el comportamiento del diseño. En estas herramientas, el comportamiento temporal se contempla a través de lo que se denominan modelos de retraso de los componentes del circuito. Sólo a partir del nivel de puertas es posible disponer de modelos suficientemente precisos como para poder considerar a los resultados del software suficientemente semejantes a los de la realidad.

Dentro de estas herramientas de análisis del comportamiento temporal de un circuito, la fundamental es el simulador. El simulador es un programa informático que, recibiendo como entrada el *netlist* del circuito y un conjunto de estímulos de las señales de entrada al circuito, es capaz de imitar el comportamiento del mismo y obtener el valor de las señales internas y de salidas correspondientes a los estímulos dados. En los niveles de puertas y transistores la simulación que se realiza sobre un circuito es, además de funcional, temporal mediante el uso de simuladores temporales. Estos simuladores permiten analizar distintos aspectos del comportamiento de un sistema como son [ABRA90]:

- Si es sensible a variaciones en los retrasos de los componentes.
- Si está libre de carreras críticas, oscilaciones, condiciones de entrada no permitidas, o estados de bloqueo.
- Permite evaluar comparativamente varios diseños.
- Permite comenzar la simulación en cualquier estado.
- Permite un análisis de las consecuencias de entradas asíncronas en circuitos síncronos.

Existen otras herramientas de verificación temporal que son las llamadas herramientas de análisis temporal. Fundamentalmente el análisis temporal tiene como objetivo básico determinar lo que se denominan los caminos críticos (*critical paths*) en un circuito, generalmente combinacional, esto es, establece los caminos que van desde las entradas hasta las salidas del circuito, que tienen un valor máximo y mínimo de retraso de propagación [MIZCO86]. A veces ocurre que los caminos críticos que se obtienen son caminos que en realidad no se activan nunca [MCGE91]. Es importante saber eliminar estos caminos en el análisis temporal, sobre todo cuando se trata de diseñar un sistema de alta velocidad. A este problema se le denomina comúnmente el problema del falso camino crítico. El análisis temporal persigue asegurar que los retrasos de propagación de un circuito no van a afectar al comportamiento del mismo. La utilidad fundamental de esta herramienta es hacer una estimación, a priori, de la máxima frecuencia de operación de los circuitos secuenciales síncronos.

Las siguientes secciones se centran fundamentalmente, en la simulación temporal y sus principales características.

1.3. La simulación temporal

Para llevar a cabo, no sólo un análisis funcional de un diseño, sino también un análisis del comportamiento dinámico es necesario realizar simulaciones temporales del mismo. Esta simulación podría realizarse con diferentes tipos de herramientas, cada una de las cuales, es idónea para un determinado tipo de circuito, en función de las siguientes características:

- El tamaño máximo del circuito que puede ser simulado con los recursos existentes.
- La velocidad de obtención de resultados.
- La precisión de los resultados obtenidos.

Un simulador ideal sería aquel que permitiera simular circuitos muy grandes, en un tiempo de simulación muy pequeño y con resultados de la más alta precisión. No obstante, mientras más se acerca algún tipo de simulador actual a alguna de estas características más se aleja de las otras. Por ello es clave en todo el proceso de simulación que un diseñador conozca las limitaciones de cada uno de los diferentes tipos de simuladores y, así, ser sea capaz de elegir el idóneo para la verificación del circuito que se está diseñando.

1.3.1. Tipos de simulación temporal

La mayoría de los simuladores temporales que se utilizan en circuitos digitales pueden clasificarse en simuladores analógicos (también llamados simuladores de nivel eléctrico) y digitales (también llamados simuladores de nivel lógico).

La simulación analógica (tipo HSPICE [HSPICE00]) trata a los circuitos como sistemas dinámicos cuyas variables (tensiones e intensidades) varían de forma continua. Este tipo de simulación se caracteriza por su alta precisión, relativa a los otros tipos de simuladores. En cambio, necesita elevados tiempos de computación para generar los resultados. Este tiempo se incrementa enormemente con el tamaño de los circuitos. Por ello se utiliza, generalmente, para verificar con alta precisión circuitos de pequeño tamaño o módulos independientes que pertenecen a circuitos más grandes. Otra utilidad que presenta es servir de referencia para verificar los resultados de otros simuladores más eficientes (menor tiempo de computación) pero, también, más imprecisos. En cambio, no es realista usar estos simuladores analógicos para la simulación temporal de circuitos digitales VLSI.

Por otra parte, los simuladores temporales digitales tratan al circuito como una red lógica cuyos nodos pueden tomar sólo valores discretos (0's ó 1's). Esto trae consigo una menor precisión en los resultados, pero, en cambio, el coste de tiempo que se necesita para generar los resultados se reduce en varios órdenes de magnitud, lo que permite analizar circuitos digitales VLSI.

Centrándonos en la simulación digital, las distintas herramientas existentes actualmente pueden dividirse en dos bloques diferentes: los simuladores a nivel de puertas lógicas y los simuladores a nivel de transistores (también llamados a nivel de conmutación, *switch-level*).

En los simuladores a nivel de puertas lógicas, un circuito consiste en un conjunto de puertas lógicas unidireccionales conectadas entre sí. Por su parte, los simuladores a nivel de transistores consideran a un circuito como una red de transistores MOS. Este tipo de simulación es específica de la tecnología MOS y surge como consecuencia de las técnicas de diseño de circuitos dinámicos implementados con estos tipos de transistores, ya que existen partes de estos circuitos que no se ajustan bien al modelo de puerta lógica (como, por ejemplo, la llave de paso) [RAO89]. Con respecto al rendimiento, en general, la simulación a nivel de transistores es más precisa que la simulación a nivel de puertas. En cambio, esta última es más rápida en la obtención de resultados.

En la tabla 1.1 se muestra un resumen de las características propias de los

diferentes tipos de simulación temporal existentes, atendiendo al tamaño máximo del circuito, a la velocidad de obtención de resultados y la precisión de los mismos. De ella se pueden obtener dos conclusiones básicas. La primera es que mientras más rápida es la simulación menor precisión tienen los resultados y a la inversa; y la segunda, es que la simulación analógica, en la práctica, es irrealizable a la hora de simular el comportamiento de circuitos digitales medianos o grandes (tipo LSI o VLSI). No obstante, debido a la alta precisión que proporciona se considera que sus resultados son equivalentes a resultados experimentales. Es por ello que este tipo de simulación tiene utilidad en dos aspectos relacionados con la simulación lógica temporal. Por una parte, es la herramienta con la que, generalmente, se realiza la caracterización de celdas que, básicamente, consiste en determinar el valor exacto de los parámetros del modelo de retraso empleado en la simulación lógica para cada una de las celdas que componen un diseño; por otra, para medir el grado de precisión que tienen los resultados de simulación lógica temporal al compararlos con los de simulación analógica.

Simulación	**Tamaño máximo del circuito**	**Velocidad**	**Precisión**	**Observaciones**
Analógica	10-100 puertas	baja	alta	No pueden simularse circuitos LSI/VLSI
Lógica a nivel de transistores	10^5 puertas	media	media	Se emplea en el diseño *full-custom*
Lógica a nivel de puertas	10^6 puertas	alta	baja	Se emplea en el diseño *semi-custom*

Tabla 1.1. Tipos de simulación temporal y características de rendimiento.

El avance tecnológico está dando lugar a circuitos integrados digitales cada vez más complejos en el sentido de que un mismo *chip* es capaz de albergar más componentes e, incluso, puede llegar a implementar un sistema completo (SOC). Esta enorme complejidad de los diseños hace los procesos de simulación de los sistemas completos cada vez más costosos en tiempo de ejecución y recursos de memoria. De hecho, es inviable emplear la simulación eléctrica e, incluso, la simulación a nivel de transistores, y el problema sólo se puede afrontar buscando niveles más altos de abstracción digital que simplifiquen la complejidad de los diseños. El problema estriba en que a mayor nivel de abstracción, menor precisión se obtiene en los resultados de simulación. Así, la precisión temporal se pierde casi completamente en niveles superiores al nivel de puertas. Por tanto, se concluye que la mejor forma de afrontar la verificación temporal es mediante simulación temporal a nivel de puertas lógicas.

1.4. La simulación temporal a nivel de puertas

Como ya se ha comentado, en la simulación a nivel de puertas lógicas, un circuito está constituido por un conjunto de componentes lógicos (puerta lógicas) que son unidireccionales y realizan funciones lógicas con los datos que llegan a sus entradas. Para considerar el comportamiento dinámico es necesario realizar un modelado a nivel lógico de dicho comportamiento. Entremos en detalle sobre cómo se puede modelar esta conducta dinámica de las puertas.

Las puertas lógicas que se usan para implementar un circuito digital son, inherentemente, dispositivos analógicos que operan con señales de tensión e intensidad con formas de onda continua (figura 1.2a). Para poder especificar y diseñar sistemas digitales es necesario hacer una abstracción de estas formas de onda continuas y transformarlas en forma de onda con valores lógicos (0's y 1's) (figura 1.2b). A nivel funcional tal abstracción facilita el uso del *Álgebra de Boole* para el tratamiento matemático de los circuitos, lo que permite especificar formalmente, y en consecuencia diseñar, los sistemas digitales. Además, tal abstracción establece una relación temporal entre las entradas y la salida de una puerta que es representada por la noción intuitiva de retraso de propagación. Por ejemplo, la salida del inversor de la figura 1.2b cambia en el instante $t_1+\Delta$ en respuesta al cambio en la entrada ocurrido en t_1, de aquí que se puede decir que el inversor tiene un retraso de propagación de valor Δ.

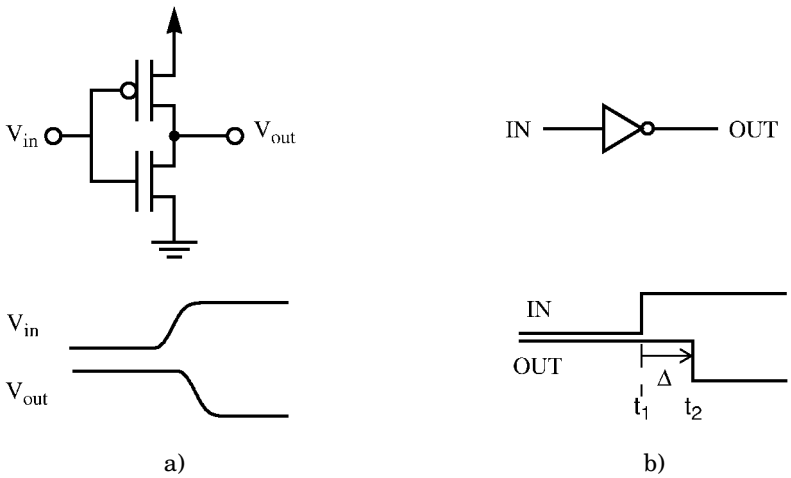


Figura 1.2. a) Inversor CMOS; b) Inversor lógico.

Estos dos niveles que se obtienen de la abstracción funcional y temporal son separados para hacer un tratamiento distinto con cada uno de ellos. La forma de hacer dicha separación consiste en, dada una entidad o puerta real, modelarlo mediante dos bloques o conjunto de bloques distintos. Uno, en el que se tiene en cuenta el nivel funcional de la puerta y otro en el que se consideraran los efectos dinámicos, esto es, el retraso de la señal. Este segundo bloque recibe el nombre de elemento de retraso. Como se observa en la figura 1.3 existen dos posibles configuraciones, en principio no excluyentes, para estos bloques. En la configuración de la figura 1.3b, los bloques dinámicos o elementos de retraso se colocan entre la señal de entrada y el bloque funcional. De esta forma, hay tantos bloques como entradas. En cambio, en la configuración de la figura 1.3c se coloca un único bloque dinámico entre el funcional y la señal de salida. A causa de que usa muchos más elementos, el primer modelo da mucha mayor flexibilidad que el segundo. Sin embargo, la mayoría de los simuladores y analizadores temporales usan el segundo ya que requiere mucho menos almacenamiento de datos.

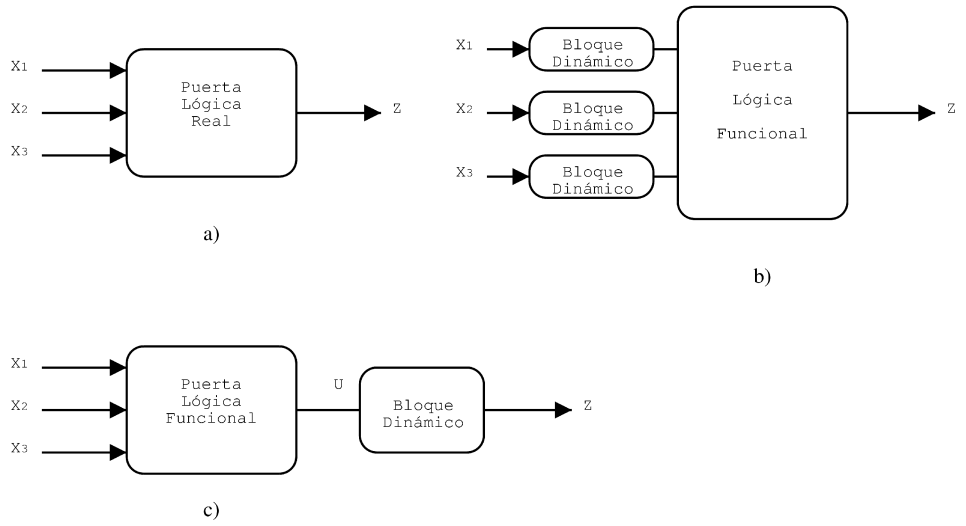


Figura 1.3. División del bloque lógico real en dos bloques: funcional y dinámico; a) Puerta lógica real; b) Modelo de retraso anterior a la puerta; c) Modelo de retraso posterior a la puerta.

El elemento de retraso queda completamente caracterizado cuando se le asocia un modelo de retraso (MR). Básicamente un modelo de esta naturaleza está constituido por un conjunto de parámetros y un conjunto de reglas que determinan cómo debe responder la señal de salida ante un cambio en la entrada. Para establecer un modelo de retraso deben tenerse en cuenta los siguientes aspectos que van a caracterizar el modelo:

- Por una parte, los efectos que se consideran en el modelo. Básicamente estos efectos se pueden agrupar en dos tipos: estáticos y dinámicos. Los estáticos

son aquéllos invariantes en el tiempo, mientras que los dinámicos van evolucionando en el tiempo. Los efectos estáticos más típicos que se consideran son: el retraso puro, la dependencia con la carga de salida y la dependencia con la geometría de los transistores. Los efectos dinámicos de mayor importancia son la dependencia con la pendiente de las transiciones de entrada y la dependencia con la proximidad entre transiciones, esto es, efectos como las colisiones y *glitches*. La tendencia actual es desarrollar modelos que incluyan el mayor número de efectos, pues de esta forma se mejora la precisión de los resultados obtenidos.

- Método de obtención de las relaciones que describen el comportamiento temporal. Existen dos formas de obtener estas relaciones que son mediante un método analítico y mediante un método heurístico. En el método analítico se parte de modelos simplificados para los transistores MOS y se deduce el comportamiento de la salida en función de los parámetros de dichos modelos simplificados (geometrías, cargas, etc) y de la forma de onda de la entrada (pendiente, etc.). El método heurístico se basa en la extracción de una cantidad importante de información de la simulación eléctrica para buscar las expresiones y/o métodos heurísticos que mejor describan dichos datos. Existen diversos métodos de ajuste, destacando el lineal a tramos (*Piece-Wise Linear* -PWL-) y el polinómico.

En la práctica, la mayoría de los modelos de retraso desarrollados llevan al menos una componente de ambos métodos. Lo mas habitual es realizar una justificación teórica-analítica de las relaciones o reglas del modelo y, en cambio, una caracterización de los parámetros del modelo a partir de los resultados de la simulación analógica. Todos los modelos tienen su propio conjunto de parámetros, cuyo valor es característico de cada una de las puertas que componen un diseño. Así mismo, todo MR debe establecer los criterios de medida de dichos parámetros y el método para su obtención a partir de, por ejemplo, la simulación analógica. Es destacable el gran esfuerzo invertido por la comunidad científica en las últimas dos décadas en el modelado y la caracterización de los circuitos integrados digitales. Resultado de estos esfuerzos es la gran cantidad de artículos publicados en este área.

En general, el modelo de retraso se asocia a cada elemento básico del circuito, existiendo una relación directa de la precisión de los resultados de simulación con lo preciso que sea el modelos de retraso empleado. Los modelos de retraso pueden clasificarse en dos categorías diferentes [JUAN99, BELL06]: deterministas y no deterministas. Un MR determinista calcula un valor concreto del retraso de propagación, mientras un MR no determinista acota el valor del retraso de propagación dentro de un rango obteniendo generalmente resultados

pesimistas, incluso, puede dar lugar a resultados falsos en determinados circuitos [MIZCO86].

El objeto de estudio en esta Tesis son los modelos de retraso deterministas, principalmente motivado por la existencia de ventajas frente a los no deterministas. Los MR deterministas permiten simular aspectos importantes de los circuitos digitales como, por ejemplo, la propagación de pulsos pequeños o estados de oscilación, en cambio, con los no deterministas no es posible y, además, aumenta considerablemente el coste de tiempo de computación de la simulación [JUAN99].

1.5. Simulación guiada por eventos

Para completar la introducción al campo de la simulación lógica a nivel de puertas, es necesario presentar la técnica de simulación empleada, esto es, cómo se desarrolla el algoritmo de simulación, cuales son las características básicas de dicha técnica de simulación denominada guiada por eventos y, las herramientas software que se utilizan.

En la práctica, el tiempo de ejecución y el consumo de memoria de las simulaciones se incrementa de forma dramática al incrementarse el tamaño del sistema modelado. Este incremento está fuertemente ligado, no solo a la naturaleza el sistema modelado, sino a la técnica de simulación implementada en la herramienta software. La simulación discreta ó también denominada, guiada por eventos, es una técnica que se legitima cuando el uso de métodos analíticos no permite obtener resultados en tiempos razonables. El éxito en la utilización de esta técnica radica en el conocimiento acumulado sobre el sistema a modelar, siendo clave establecer un buen modelo discreto que incluya suficientes variables para establecer tanto el estado del sistema, como la evolución del mismo mediante un conjunto de relaciones que caracterizan el sistema.

En el campo de la simulación de circuitos digitales integrados, la figura 1.4, muestra una visión de alto nivel de los diferentes elementos englobados en el proceso de simulación. El elemento clave de este proceso es el programa de simulación, caracterizado por incluir modelos internos del comportamiento para los componentes del circuito. El programa de simulación recibe, por una parte, una descripción del circuito a simular o *netlist* y por otra parte, el conjunto de estímulos y los parámetros de control que estime el diseñador. Tras finalizar la simulación se generan unos resultados con el comportamiento del diseño que se está analizando.

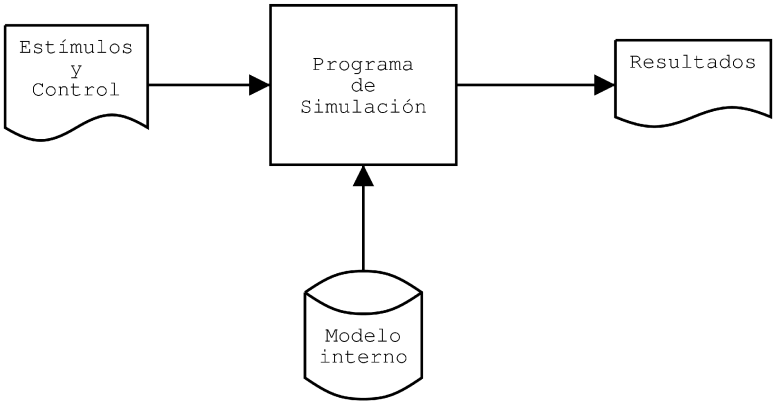


Figura 1.4. Proceso de simulación digital.

Las herramientas de simulación lógica-temporal están basadas en la técnica de simulación controlada por eventos/sucesos (*event-driven*). Aplicando esta técnica, el circuito digital queda modelado de forma discreta en el tiempo, analizándose exclusivamente los sucesos y su propagación a través de los diferentes componentes del circuito. En simulación lógica-temporal, un suceso es el cambio en el valor de una señal de 0 a 1 o a la inversa. Esto implica un cambio sustancial con respecto a la simulación a nivel eléctrica de circuitos, en la que se realiza un análisis del valor de las señales considerándolas señales reales (V o I) que varían de forma continua, calculando el valor de las señales en cada instante de tiempo. Esta diferencia fundamental motiva que la simulación lógica, controlada por sucesos, sea varios órdenes de magnitud más rápida que la simulación eléctrica. A continuación se va a describir en mayor profundidad las características básicas de la simulación controlada por eventos y el modelo de simulación establecido para los circuitos digitales.

1.5.1. Características básicas de la simulación *event-driven*

En todo sistema discreto por eventos, la propiedad en estudio sólo cambia de valor o estado en instantes discretos de tiempo y determinados por el estado, las entradas y las relaciones que caracterizan el modelo de sistema. En la simulación digital el parámetro en estudio es la evolución temporal de los valores lógicos en los nodos del circuito, y estos valores sólo cambian en aquellos instantes en los que conmuten determinadas puertas, por lo que la simulación temporal queda discretizada a estos instantes. Durante los intervalos discretos de tiempo establecidos, el estado del sistema, en su conjunto, queda representado por los denominados descriptores, pudiendo ser éstos tanto discretos como continuos. Para el caso de la

simulación lógica estos descriptores podrán ser estados de las conexiones, valores lógicos en entradas de puertas, etc. No obstante, de manera general, se puede establecer que un sistema discreto controlado por eventos, engloba los siguientes conceptos:

- **Evento**: Valor discreto de un descriptor que indica un posible cambio de estado del sistema en conjunto. Para la simulación lógica el instante de tiempo en que se produce una conmutación lógica.
- **Trabajo a realizar**: Denota el mecanismo de procesado de los elementos, en nuestro caso el algoritmo de simulación.
- **Camino**: Es un conjunto de reglas que nos indica la ruta que siguen los elementos procesados. En el caso de la simulación lógica será la descripción del circuito, ya que éste nos indica los lugares en los que pueden aparecer cambios lógicos al conmutar una determinada puerta.
- **Cola o buffer:** Es el lugar de espera en el que residen los elementos que todavía no han sido procesados.
- **Orden de procesado o secuencia**: Son las reglas que indican cual de los elementos que se encuentra pendiente de procesar debe ser procesado en primer lugar.

Otro concepto que aparece en los sistemas discretos por eventos es el concepto de bucle abierto o cerrado. Éste hace referencia al lugar donde terminan los elementos procesados, de forma que, si al procesarse algún elemento, éste reaparece en la cola de procesado se dice que el sistema está en bucle cerrado, siendo en el caso contrario en bucle abierto. Esta propiedad, en el caso de la simulación lógica, estará referida a los circuitos con realimentación, ya que en éstos, tras analizar un estímulo en una determinada puerta se puede producir otro de forma que acabe afectando a la misma puerta. Por tanto, el concepto de bucle cerrado en el caso de simulación lógica lo haremos extensible a circuitos con realimentación. Si se concretan los conceptos expuestos para el problema de la simulación lógica se obtiene:

- Los eventos son aquellos instantes de tiempo en los que se activa alguna de las entradas de una celda lógica.
- El trabajo a realizar es la simulación lógica, mostrándose en la figura 1.5 de manera general cual es el algoritmo de simulación. Básicamente, éste consiste en el procesado de un conjunto de estímulos de forma ordenada en el tiempo, sabiendo que, cada estímulo o evento, tras su procesado, puede producir otros eventos que habrá que añadirlos al conjunto de eventos aún no procesados y procesarlos de forma ordenada en el tiempo, finalizando la

simulación cuando todos los eventos estén procesados.

- El camino queda determinado por el *netlist* del circuito, por tanto, un evento sólo puede producir otro en las salidas de una determinada puerta.
- La cola de procesado contendrá aquellos eventos producidos en las salidas de las celdas.
- La cola de eventos seguirá un orden de procesado creciente en el tiempo

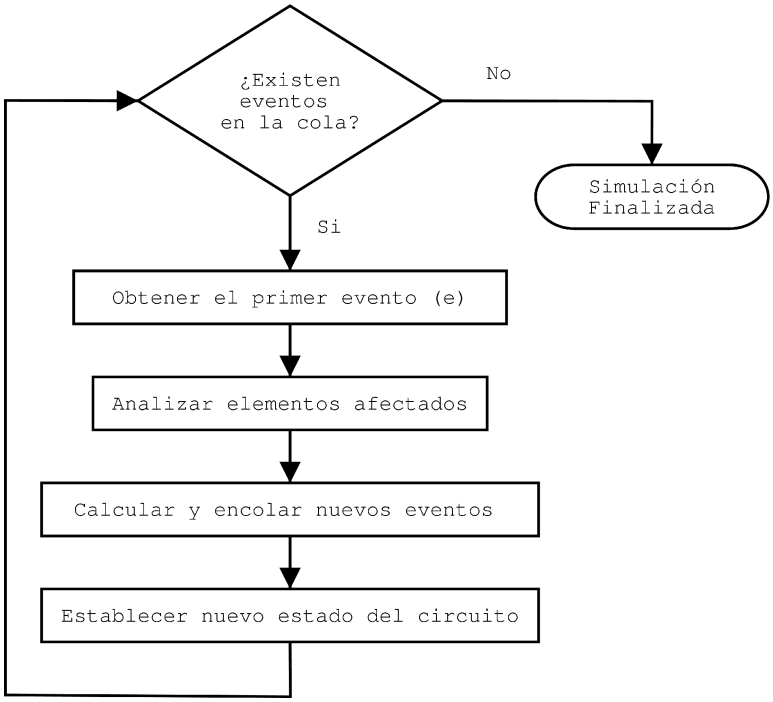


Figura 1.5. Visión general de un algoritmo de simulación lógica.

Para terminar el capítulo se comentan una serie de parámetros descritos en la bibliografía [FISH01], que caracterizan los sistemas discretos por eventos, que serán objeto de estudio en la implementación de HALOTIS, presentada posteriormente en el capítulo 4. Estos parámetros son:

- **Tamaño de la cola u ocupación de buffer**: Indica el número de elementos en el sistema en espera de ser procesados.
- **Retraso o tiempo de espera**: Indica el tiempo que permanece un elemento en la cola de espera hasta ser procesado.
- **Productividad**: Número de elementos procesados por unidad de tiempo, parámetro utilizado para medir el rendimiento del software que implementa un simulador discreto.

Capítulo 2. Modelos de retraso

En la simulación lógica el modelo de retraso predice el comportamiento temporal de cada uno de los componentes del circuito. La precisión de los resultados de simulación dependen de la precisión del modelo de retraso empleado. Nuestro interés se centra en modelos de retraso deterministas para tecnologías CMOS, motivado por la ventajas, mencionadas en el capítulo anterior, que presentan frente a los no-deterministas.

El capítulo, en su primera sección, hace un recorrido de diferentes modelos de retraso deterministas destacando los efectos contemplados con cada modelo. La segunda y tercera sección tratan el efecto de degradación del retraso de propagación y el efecto inercial respectivamente, presentando finalmente el denominado modelo IDDM (*Inertial and Degradation Delay Model*). Tras presentar el modelo IDDM se incluye un trabajo original de aplicación de dicho modelo a celdas CBL (*Current Balanced Logic*). Por último, se presenta la adaptación concreta de los modelos en el simulador lógico temporal HALOTIS.

2.1. Modelos de retraso deterministas

Los modelos deterministas calculan valores específicos para el retraso de propagación en cada componente del circuito, donde el valor del retraso depende de diversos factores. Así, se clasifican en dos grandes grupos: (1) retraso unitario, en el que cada componente tiene siempre el mismo retraso y, (2) asignable, donde el retraso en cada componente difiere. La figura 2.1 muestra esta clasificación más detallada.

Los modelos asignables consideran efectos para calcular el retraso de cada componente, como puede ser la capacidad en los nodos y la forma de onda de las señales. Se contemplan dos tipos de modelos de retrasos asignables: estáticos y dinámicos. Mientras en los modelos estáticos el retraso sólo depende del circuito y de las características de la puerta, en los modelos dinámicos, el retraso también depende de la forma de onda en la entrada de la puerta.

Los modelos estáticos son menos precisos que los dinámicos al no tener en cuenta los estados anteriores de la puerta. Hay diversos factores en la señal de entrada que pueden cambiar el retraso de propagación, por ejemplo, la pendiente de entrada o la proximidad temporal de transiciones en una entrada de la puerta (colisiones de entrada). Modelar estos efectos hace posible tratar en la simulación lógica la propagación de *glitches* en los circuitos o detectar y controlar estados oscilatorios [ABRA90].

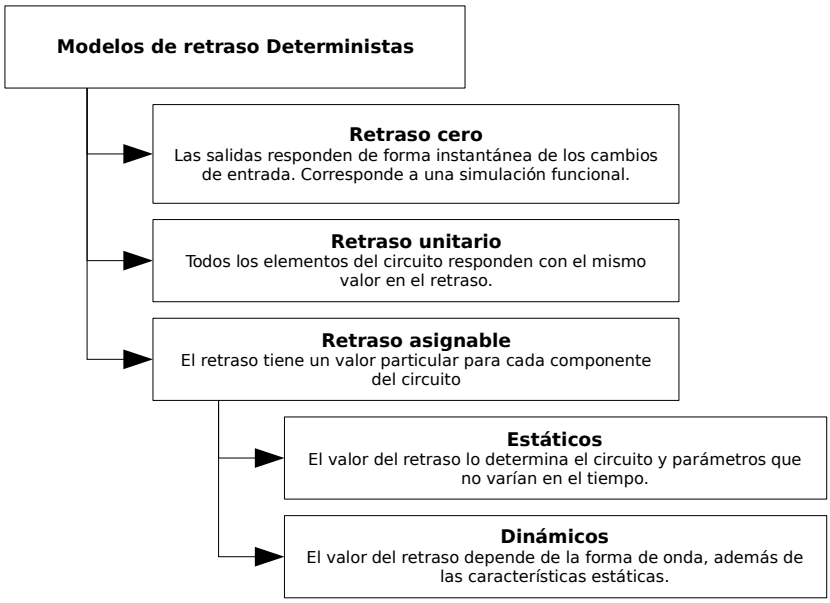


Figura 2.1. Clasificación de los modelos de retraso deterministas.

En las siguientes secciones se estudian un conjunto de MR muy utilizados en los simuladores a nivel de puertas, presentando sus relaciones básicas. Siguiendo un orden creciente de complejidad y, consecuentemente, de precisión en los resultados obtenidos, se parte de aquellos que son útiles fundamentalmente para la verificación funcional y se continua con aquellos que sirven para la verificación temporal. Dentro de estos últimos, en primer lugar se presentan los modelos estáticos y seguidamente los que incluyen efectos dinámicos.

2.1.1. Modelos de retraso cero y unitario

Los de menor complejidad son los denominados MR Cero y MR Unitario que, como ya se ha dicho, fundamentalmente son útiles para realizar la verificación funcional del circuito sin tener en cuenta el aspecto temporal, al menos con un grado de precisión mínimo para considerarla como tal.

El MR Cero no considera retrasos de propagación. Esto es, en la abstracción de la puerta real sólo se considera el nivel funcional eliminando el temporal (figura 2.2).

Figura 2.2. Modelo de retraso cero.

El MR Unitario es el más sencillo que ya implica el uso de un elemento de retraso en la abstracción hecha sobre la puerta real. Como se observa en la figura 2.3, el modelo consta de un único parámetro de valor unidad. Este es el valor del retraso de propagación de la señal de salida de una puerta respecto al cambio en la entrada. Es importante destacar que, con este modelo, todas las puertas que componen el diseño de un circuito tienen el mismo comportamiento temporal, lo cual es una idealización de la realidad que no siempre resulta aceptable.

Figura 2.3. Modelo de retraso unitario.

Aunque los resultados que proporciona el uso de este modelo siguen siendo poco precisos como para considerar que se ha realizado una verificación temporal, sí es posible detectar algunos fenómenos temporales como por ejemplo los azares (figura 2.4).

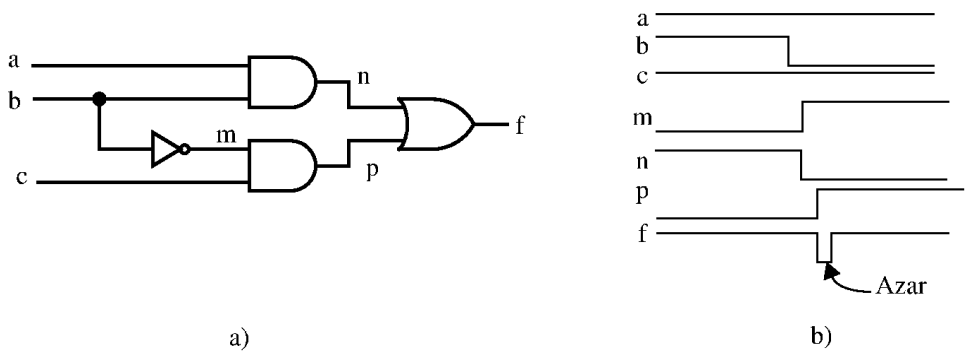


Figura 2.4. a) Circuito digital; b) Simulación con el modelo unitario.

2.1.2. Modelos de retraso estáticos

El primer modelo que empieza a aportar un grado suficiente de precisión en los resultados de la simulación lógica es el denominado Modelo de Retraso Asignable (MRA). Se caracteriza porque a cada puerta lógica se le asocia un elemento de retraso propio, esto es, el valor de los parámetros del modelo es propio de cada puerta. Teniendo en cuenta esta definición, realmente todos los modelos que se presentan en lo que sigue son MRA.

La versión más simple consta de un único parámetro, D, llamado retraso puro, para cada elemento de retraso. La regla que determina el funcionamiento del bloque es la siguiente: cualquier cambio en la entrada del elemento de retraso aparece en la salida con un tiempo de retraso igual a D, tal como se aprecia en la figura 2.5a. La diferencia entre esta versión y el Modelo de Retraso Unitario está en que el valor de D puede variar de una puerta a otra.

La segunda versión considera dos valores distintos para el retraso puro del mismo elemento, según sean las transiciones de 0 a 1 (D_H) o de 1 a 0, (D_L) y opera como se muestra en la figura 2.5b. De esta forma se contempla el efecto de la diferencia en los retrasos de propagación según el tipo de cambio en la señal de salida. En la misma puerta, D_H y D_L son, en general, distintos y los valores de D_H (y de D_L) pueden variar de una puerta a otra.

Otra versión surge al analizar el valor del retraso de propagación en función de la carga en la salida de una puerta. Se obtiene una relación lineal entre ambos, que da lugar a la siguiente expresión:

$$D = D_0 + m_c \times C_L \tag{2.1}$$

siendo C_L la capacidad de la carga en la salida y D_0 y m_c los parámetros característicos de la puerta. Un elemento de retraso con esta versión del modelo de retraso

opera tal y como se muestra en la figura 2.5c. En esta versión se incorporan al modelo aspectos sobre la situación en la que se encuentra la puerta dentro de un circuito (el *fanout*, por ejemplo).

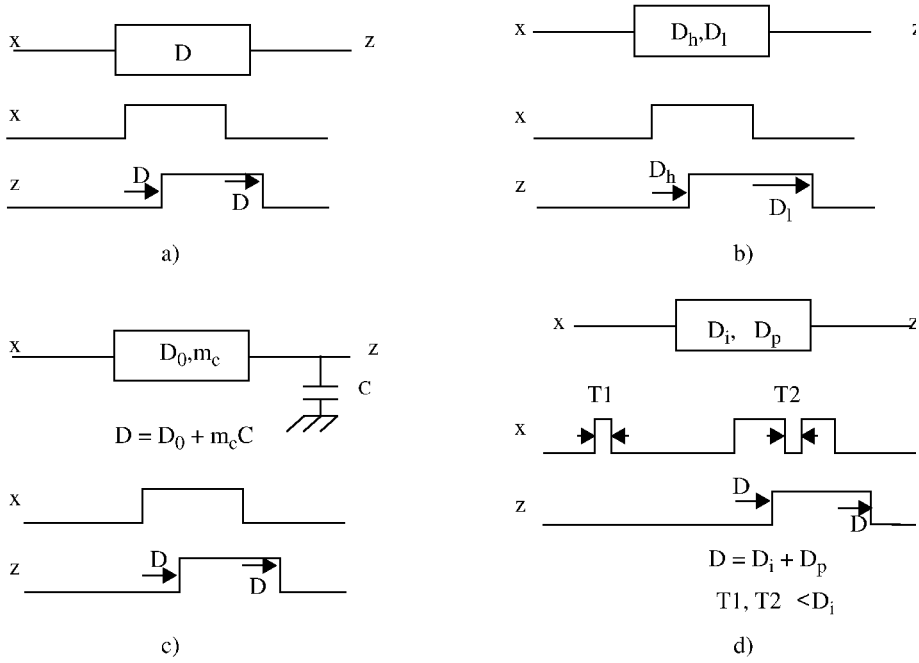


Figura 2.5. Modelos de retraso asignables: a) Retraso puro; b) Retraso puro con distinción entre subida y bajada; c) Retraso puro considerando el efecto de carga en la salida; d) Retraso puro e inercial.

Una versión más compleja de este modelo asignable introduce un efecto observado en las puertas reales, que consiste en la no propagación a través de las puertas de pulsos pequeños. Este efecto se denomina efecto inercial [UNGE69]. En esta versión cada elemento de retraso consta de dos parámetros distintos: D_p o componente pura y D_i o componente inercial. Las reglas de funcionamiento son las siguientes (figura 2.5d): un pulso de entrada al elemento de retraso de anchura T es propagado a la salida con un retraso total $D=D_p+D_i$ si $T>D_i$ y, en cambio, es eliminado (esto es, la salida no cambia) si $T \leq D_i$. El efecto inercial se estudia detalladamente en la sección 2.3.

A modo de resumen, en las distintas versiones del Modelo de Retraso Asignable que se han visto, se han considerado los siguientes efectos del comportamiento real de las puertas:

- Efecto de retraso de propagación puro, parámetro D.
- Distinción entre retrasos de propagación según el tipo de transición:

parámetros D_H y D_L.

- Efecto de variación del retraso en función de la carga en la salida, parámetros D_0 y m_c para la puerta y C para el circuito.
- Efecto inercial o de eliminación de pulsos pequeños: parámetros D_p y D_i.

Estos modelos tienen un proceso de caracterización del valor de sus parámetros relativamente sencillo y rápido sin tener que desarrollar modelos analíticos para obtener expresiones de los mismos.

2.1.3. Efectos dinámicos

El amplio uso de los modelos estáticos ha permitido observar cómo los resultados que se obtienen en la simulación lógica no son lo suficientemente precisos. El motivo fundamental reside en el hecho de que los retrasos de propagación de una puerta varían según sea la forma de onda de la señal de entrada (i.e., manteniendo la puerta en las mismas condiciones de carga tanto en la entrada como en la salida). Esto ha llevado a desarrollar nuevos MRA en los que se tengan en cuenta estos efectos, genéricamente denominados MR dinámicos.

Actualmente los modelos dinámicos existentes tienen en cuenta fundamentalmente dos efectos de la forma de onda: la pendiente de la señal de entrada y la proximidad entre cambios consecutivos, denominado este último efecto de colisiones [MELC92]. En la figura 2.6 se muestran ejemplos de estos efectos.

La pendiente de entrada se refiere al hecho de que las señales reales son continuas en el tiempo y tardan un cierto tiempo en conmutar de un valor lógico al otro. Este tiempo es lo que se denomina tiempo de transición y se caracteriza por el valor de la pendiente de entrada. Como se observa en la figura 2.6a, si varía este tiempo de transición, el valor del retraso de propagación variará significativamente.

Con respecto a las colisiones, existen dos tipos distintas. El primero (figura 2.6b) está causado por cambios próximos en las entradas que producen cada uno de ellos cambios en la salida en sentido opuesto. De esta forma, este tipo de cambio de entrada da lugar a pulsos pequeños o *glitches* en la salida de la puerta y que pueden propagarse a través del resto del circuito. El segundo tipo de colisión es aquel en que existen cambios próximos en las entradas que tienden a producir el mismo cambio en la salida de la puerta (figura 2.6c). Generalmente ocurre que el cambio en la salida se acelera, como consecuencia, el retraso de propagación disminuye.

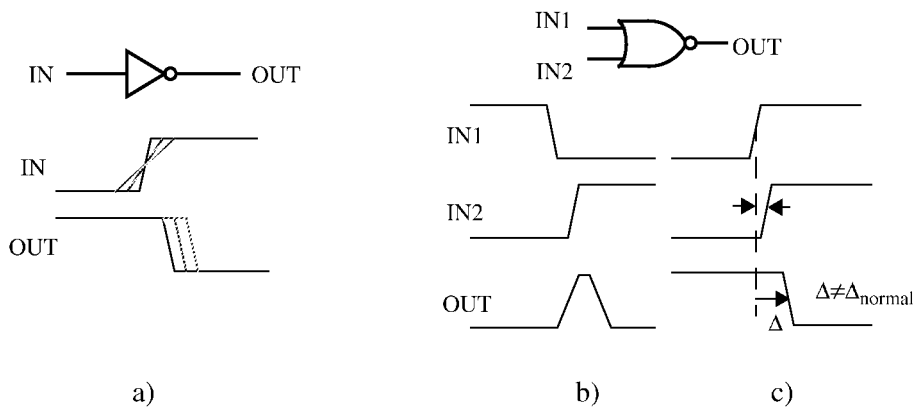


Figura 2.6. Efectos dinámicos:a) Pendiente de la señal de entrada; b) Colisión de las entradas que producen cambios distintos en la salida; c) Colisión en las entradas con el mismo efecto en la salida.

Existen un gran número de trabajos reportados que presentan MRA en los que se incluye alguno de estos efectos dinámicos para diferentes tipos de tecnologías, alguno de los cuales son los siguientes: [VALE86, DENG88, DESC88, CALV91, JEPP94, BELL94, JUAN97]. Lo más destacable de estos trabajos es que la metodología de obtención del MRA es analítica o heurística, según sea el efecto considerado. Así, los trabajos que incluyen el efecto de la pendiente de entrada parten de un modelo analítico y dan lugar a expresiones que, eso sí, se ajustan por métodos heurísticos. En cambio, los trabajos que incluyen el efecto de colisiones están basados en métodos heurísticos, generalmente en resultados de simulación analógica. Como ejemplo de desarrollo de estos modelos vamos a presentar dos, uno que incluye el efecto de pendiente de entrada y otro que incluye el efecto de colisiones, en concreto las colisiones que dan lugar a pulsos pequeños.

2.1.4. Modelo de retraso considerando el efecto de la pendiente de entrada

Como hemos mencionado, es fácil observar que el retraso de propagación varía apreciablemente al cambiar la pendiente de la forma de onda de la transición de entrada. El estudio minucioso de estas variaciones es un problema complejo ya que, a medida que va cambiando la señal de entrada, los dispositivos que componen las puertas lógicas (transistores MOS) atraviesan diferentes regiones de operación, haciendo difícil calcular la forma de onda de la transición de salida y, por tanto, el retraso de propagación. Este problema ha sido ampliamente estudiado en la bibliografía [DENG88, DESC88, JEPP94].

Pese a esta complejidad, este problema puede ser tratado analíticamente en una primera aproximación. En efecto, los dispositivos MOS disponen de una tensión umbral que separa las zonas de corte y conducción del dispositivo. Si consideramos, dentro de esta primera aproximación, que en cierto instante durante la transición de entrada los dispositivos MOS pasan de estar completamente cortados a un estado de máxima conducción o viceversa, es posible hacer un estudio simplificado y obtener expresiones que reflejen la influencia de la pendiente de la señal de entrada.

Para realizar el estudio empleamos un inversor (figura 2.7) sin que por ello exista pérdida de generalidad, ya que es el caso al que se reduce el estudio de puertas multientrada cuando una sola de dichas entradas está activa. Nos centraremos además en el estudio de una transición de entrada bajo-alto (transición de salida alto-bajo) siendo el otro caso completamente análogo.

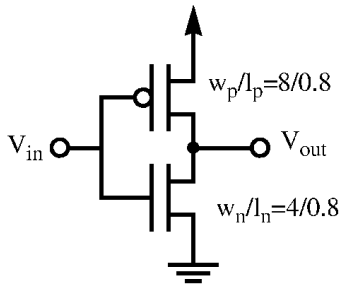


Figura 2.7. Inversor CMOS.

Las hipótesis de partida son: (1) el transistor NMOS posee una tensión V_{GS} umbral (V_{TN}) de modo que si $V_{GS}<V_{TN}$ el dispositivo no conduce mientras que si $V_{GS}>V_{TN}$ el dispositivo conduce plenamente (lo mismo se aplica al PMOS); (2) para una transición de entrada de tipo bajo-alto, el nodo de salida resultará descargado, si bien dicha descarga no comenzará hasta que el transistor PMOS haya sido cortado; (3) a partir del instante del corte, puede considerarse que el NMOS se encuentra en un estado de conducción plena y la salida se descargará en estas condiciones. En resumen, y para el caso de una entrada bajo-alto, la situación es la que se muestra en la figura 2.8: al principio el PMOS está activado y el NMOS cortado, por lo que la salida permanece en estado alto. Cuando V_i alcanza V_{TN}, el NMOS pasa a conducir, pero la salida permanece cargada ya que el PMOS sigue suministrando carga. Cuando el PMOS pasa a estar cortado ($V_i=V_{DD}-|V_{TP}|$) sólo el NMOS permanece conduciendo y comenzará la descarga. La descarga en estas condiciones corresponde al retraso de propagación que sufriría una transición de entrada de tipo escalón (pendiente infinita ó tiempo de transición cero).

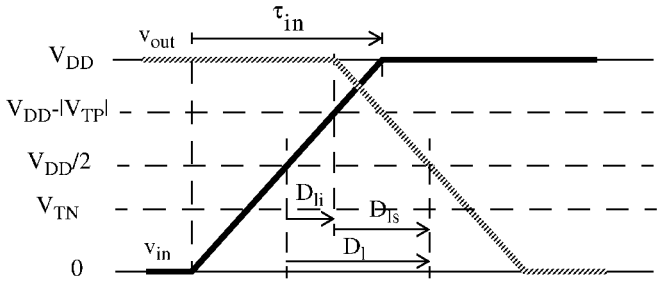


Figura 2.8. Retraso de propagación de un inversor CMOS ante una entrada de pendiente finita.

En el retraso de propagación aparecen por tanto dos componentes:

$$D_l = D_{li} + D_{ls} \tag{2.2}$$

donde D_{ls} es el retraso ante entrada escalón, característico de la puerta y D_{li} es un parámetro geométrico que depende de la pendiente de entrada y que puede obtenerse fácilmente de la figura:

$$\frac{V_{DD} - |V_{TP}| - \frac{V_{DD}}{2}}{D_{li}} = \frac{V_{DD}}{\tau_{in}} \qquad D_{li} = \tau_{in}\left(\frac{1}{2} - \frac{|V_{TP}|}{V_D}\right) \tag{2.3}$$

donde τ_{in} es el tiempo de subida de la transición de entrada linealizada. Se tiene por tanto una dependencia lineal entre el retraso global D_l y el tiempo de subida. Generalizando para ambos tipos de transición:

$$D_l = A_l \times \tau_{in} + D_{ls} \qquad D_h = A_h \times \tau_{in} + D_{hs} \tag{2.4}$$

donde A_l y A_h son parámetros tecnológicos cuyo valor está en torno a 0.2. Por otra parte, D_{ls} y D_{hs}, por definición, son independientes de la pendiente de entrada, pero mostrarán la dependencia lineal esperada con la carga:

$$D_{ls} = m_{ls} \times C_L + D_{ls0} \qquad D_{hs} = m_{hs} \times C_L + D_{hs0} \tag{2.5}$$

quedando finalmente cada puerta caracterizada por un total de seis parámetros.

2.2. Efecto de degradación y modelo DDM

Cuando el retraso de propagación en un circuito es mucho más pequeño que la frecuencia de operación, las señales y las puertas se pueden considerar ideales. En esta situación cada transición no es afectada por la previa. No obstante, cuando la frecuencia de operación aumenta aparecen efectos no ideales que comienzan a ser relevantes, teniéndose que incluir en los modelos de retraso.

El efecto de degradación, presentado en [JUAN97], proporciona una importante mejora en la precisión de los resultados de la simulación lógica, debido al tratamiento de efectos dinámicos como las transiciones simultáneas en las entradas de las puertas y la propagación de pulsos pequeños o *glitches*, que no eran considerados anteriormente. Estos efectos se engloban en el llamado efecto de degradación del retraso, de donde toma su nombre el modelo de retraso: *Degradation Delay Model* - DDM (Modelo de Degradación del Retraso).

El modelo DDM es un modelo del retraso en puertas lógicas que, además de los efectos clásicos, introduce el efecto de degradación en el cálculo del retraso de propagación de transiciones a través de una puerta. Este efecto ocurre en circuitos digitales sometidos a altas frecuencias de operación y en situaciones de excitaciones críticas y puede interpretarse como una generalización del efecto inercial.

Básicamente el efecto de degradación hace decrecer la anchura de un pulso en la salida de una puerta lógica cuando el pulso en la entrada que lo generó es pequeño. Al crecer la anchura del pulso de entrada, el retraso de propagación aumenta hasta alcanzar su máximo, correspondiente al retraso de propagación normal. Por otro lado, si el ancho del pulso decrece, el retraso de propagación disminuye hasta alcanzar un mínimo en el ancho del pulso a partir del cual no es propagado. Este fenómeno ha sido incluido en los modelos de retraso como efecto inercial [UNGE69].

Un parámetro apropiado para describir el modelo DDM es el tiempo transcurrido desde la última transición en la salida de la puerta (parámetro T) mostrado en la figura 2.9.

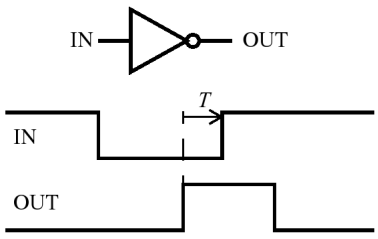


Figura 2.9. Parámetro T.

Según el valor del parámetro T se distinguen tres regiones de comportamiento mostradas en la figura 2.10:

1. En la primera región se considera que el valor T es suficientemente grande para que la salida de la puerta alcance un valor estable, V_{DD} en el ejemplo mostrado en la figura 2.10a. En esta situación cuando llega una nueva transición, el retraso no depende del instante en que ocurrió la transición anterior, por tanto, es propagada con el valor del retraso de propagación

normal, independientemente de T.

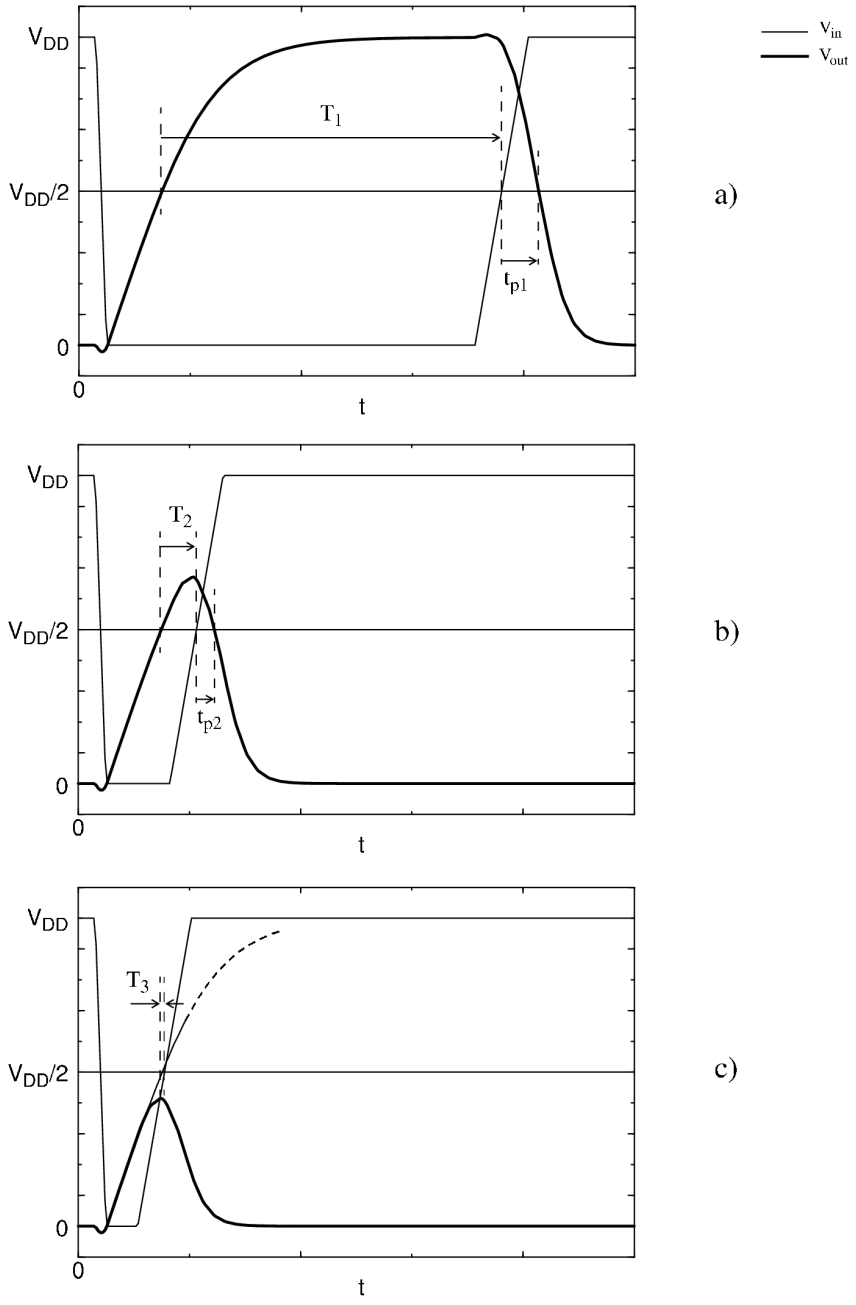


Figura 2.10. Regiones según el valor de T: a) Propagación normal; b) Efecto de degradación; c) Efecto inercial.

2. La segunda región, correspondiente a la figura 2.10b, ocurre cuando la llega una nueva transición en la entrada y la salida todavía está cambiando desde el valor $V_{DD}/2$ a V_{DD}. El retraso de propagación de esta nueva transición, t_{p2}, es menor que el retraso de propagación normal ($t_{p2} < t_{p1}$), propagándose la transición con degradación.
3. En la última región T es pequeño (figura 2.10c) y tras llegar la segunda transición la salida no alcanza el umbral de conmutación lógica $V_{DD}/2$, filtrándose en este caso, es decir, ocurre el efecto inercial.

En la última situación parece haber un conflicto con la interpretación del parámetro T como medida desde la ultima transición en la salida, puesto que, la transición de salida no ha llegado a suceder. La solución consiste en precisar la definición del parámetro T como: el tiempo medido desde el instante de la última transición de salida predecida, ocurra esta última o no, hasta el instante donde la siguiente transición de salida sucede.

Con la definición dada al parámetro T, en la figura 2.11, se representa el retraso de propagación normal frente a T, donde se distinguen tres regiones mencionadas: (1) zona sin degradación, el retraso de propagación normal no es afectado; (2) la zona de degradación donde el retraso de propagación es menor que el retraso de propagación normal; (3) la zona de filtrado.

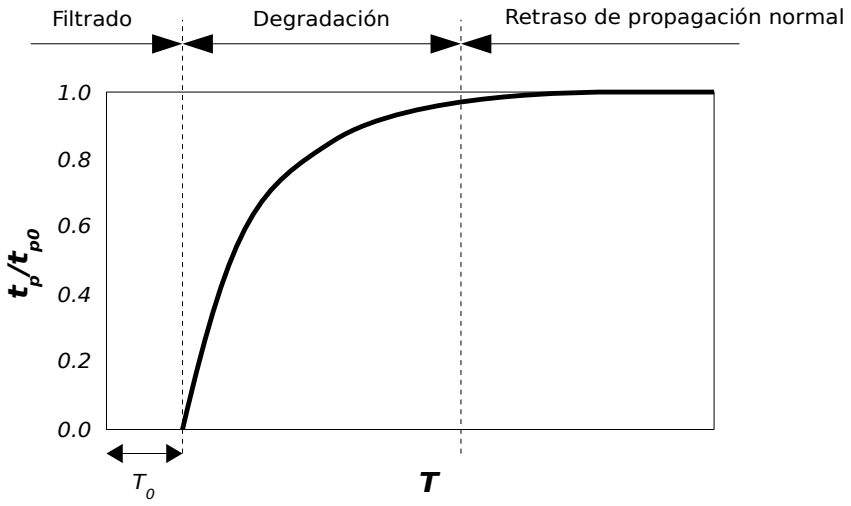


Figura 2.11. Diferentes regiones según el tiempo de propagación (t_p) frente al parámetro T.

2.2.1. Modelado de la degradación

El parámetro T se interpreta como una variable de estado de la puerta que determina el estado cuando llega una nueva transición. En la figura 2.12 se observa la respuesta de una puerta lógica a dos transiciones de entrada sucesivas. En estas condiciones, el retraso de propagación t_p depende del tiempo transcurrido desde la última transición de salida T como se muestra en la figura 2.11. Esta dependencia constituye el efecto de degradación y puede expresarse como:

$$t_p = t_{p0}\left(1 - e^{\frac{T - T_0}{\tau}}\right) \tag{2.6}$$

donde t_{p0} es el retraso para valores muy grandes de T denominado retraso de propagación normal, y τ y T_0 son los parámetros de degradación. El retraso de propagación normal ha sido ampliamente estudiado en la bibliografía [DAGA98, BISD98].

Los parámetros de degradación τ y T_0 son particulares para cada combinación de entradas de la puerta, así como para cada transición (alto-bajo ó bajo-alto). Además, dependen de:

- La capacidad de carga de la salida C_L.
- La duración de la transición de entrada τ_{in}.
- La tensión de polarización V_{DD}.
- La estructura interna de la puerta, caracterizada por los parámetros geométricos de los transistores (W y L).

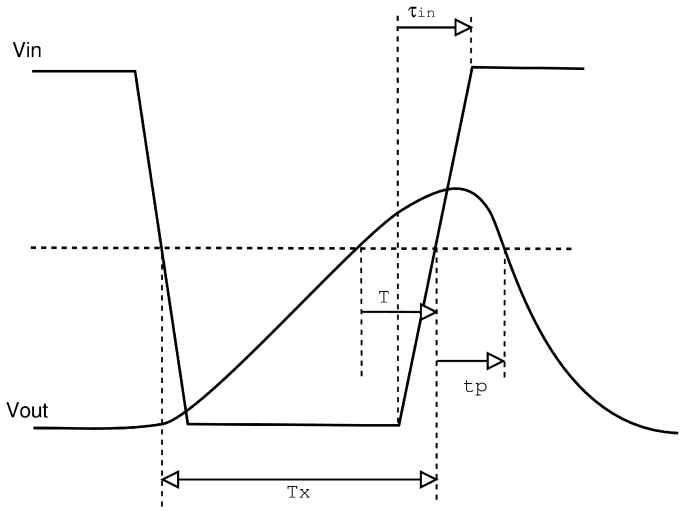


Figura 2.12. Modelo de retraso.

Los parámetros de degradación se encuentran caracterizados en [JUAN00a] a dos niveles: a nivel de celda lógica y a nivel de parámetros tecnológicos para el caso del inversor CMOS. En el primer caso, los parámetros de degradación τ y T_0 se obtienen a partir de las variables fijadas externamente a la puerta (τ_{in}, C_L, V_{DD}) junto con unos parámetros propios y característicos de la puerta. En el segundo caso, τ y T_0 se obtienen de las mismas variables externas, junto con las variables que describen la estructura interna del inversor (W_N y W_P) y un conjunto de parámetros característicos del proceso de fabricación.

Los parámetros τ y T_0 se obtienen mediante simulación eléctrica por regresión lineal en la ecuación 2.6, reescribiéndola como:

$$T = T_0 + \tau\left(-\ln(1-\frac{t_p}{t_{p0}})\right) \tag{2.7}$$

2.3. Efecto inercial

El modelo clásico del efecto inercial se basa en la definición de retraso inercial D_i [UNGE89]. Según este modelo, todo pulso en la entrada de una puerta de duración menor a la magnitud D_i es filtrado. La mayoría de los simuladores lógicos utilizan el retraso inercial para implementar el efecto inercial.

Para calcular el valor del retraso inercial de una puerta se establece el criterio universal, para el valor del umbral, en el valor $V_{DD}/2$ y, además éste mismo valor se utiliza para medir los cambios de la señal a nivel lógico. Un pulso de entrada en una puerta es filtrado si el pulso generado en la salida no alcanza el umbral $V_{DD}/2$, estableciéndose que un pulso de menor ancho que D_i genera un pulso en la salida de amplitud menor de $V_{DD}/2$. Este modelo simple no es preciso en muchas situaciones, por ejemplo, la figura 2.13a contiene un circuito de test con un primer inversor (g_0) conectado en su salida a otros dos inversores (g_1, g_2). Los inversores g_1 y g_2 tienen curvas DC (figura 2.13b) con diferentes tensiones umbrales: $V_{T1}=1.32V$ y, $V_{T2}=3.41V$. Estos umbrales se definen como el valor de tensión de entrada que hace $V_{out}=V_{DD}/2$, considerado buen procedimiento para obtener el valor lógico de una puerta dado un valor de tensión de entrada.

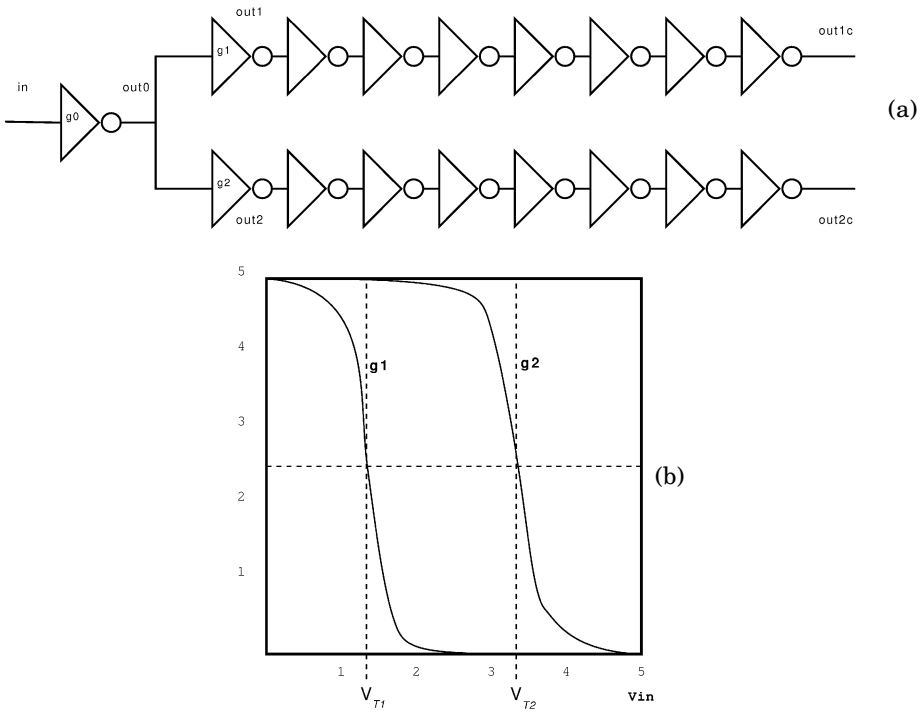


Figura 2.13. Ejemplo de fallo del modelo inercial: a) Circuito de ejemplo; b) Curvas DC de los inversores.

Las cadenas de inversores conectadas a las salidas de g_1 y g_2 tienen la propiedad de poder regenerar los pulsos que reciban en sus entradas. El circuito ha sido simulado con HSPICE, introduciendo un pulso de anchura menor que el retraso inercial (D_i). Puesto que el pulso es de menor anchura que la magnitud D_i el pulso generado en la salida out_0 no alcanza el valor umbral $V_{DD}/2$. Desde el punto de vista del modelo inercial, el pulso es filtrado en la puerta g_0 y no se contempla más actividad de conmutación en el resto del circuito. En cambio, con la precisión del simulador eléctrico se comprueba en la figura 2.14 que el pulso generado en out_0 si es propagado por el inversor g_2 y es regenerado en la cadena de inversores conectada a out_2, mientras que sí es filtrado por g_1. En cambio, a nivel lógico con el modelo de retraso inercial los resultados son los mostrados en la figura 2.15. Este comportamiento es debido a que el efecto inercial ocurre en g_1 pero no en g_2, obteniéndose resultados erróneos con el modelo de retraso inercial.

El motivo de esta imprecisión en el retraso inercial es debida al uso de un único valor de tensión umbral para todas las puertas del circuito ($V_{DD}/2$). Al definir valores umbrales individuales para cada puerta, un pulso en un nodo como out_0, puede ser propagado a través de una puerta, siempre que alcance su valor umbral como ocurre con g_1, mientras que en otra puerta si no alcanza su valor umbral no es propagado, como ocurre en g_2.

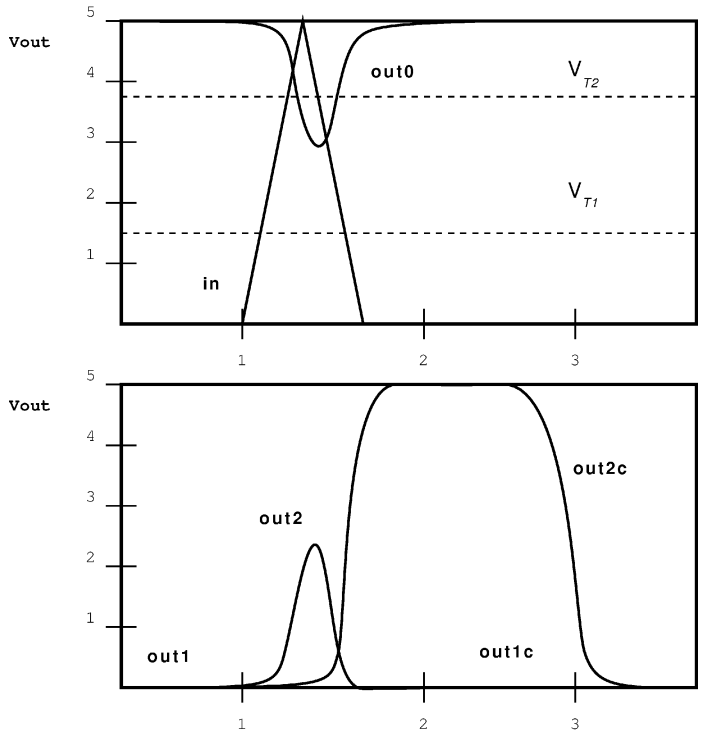


Figura 2.14. Simulación con HSPICE del circuito de la figura 2.13a.

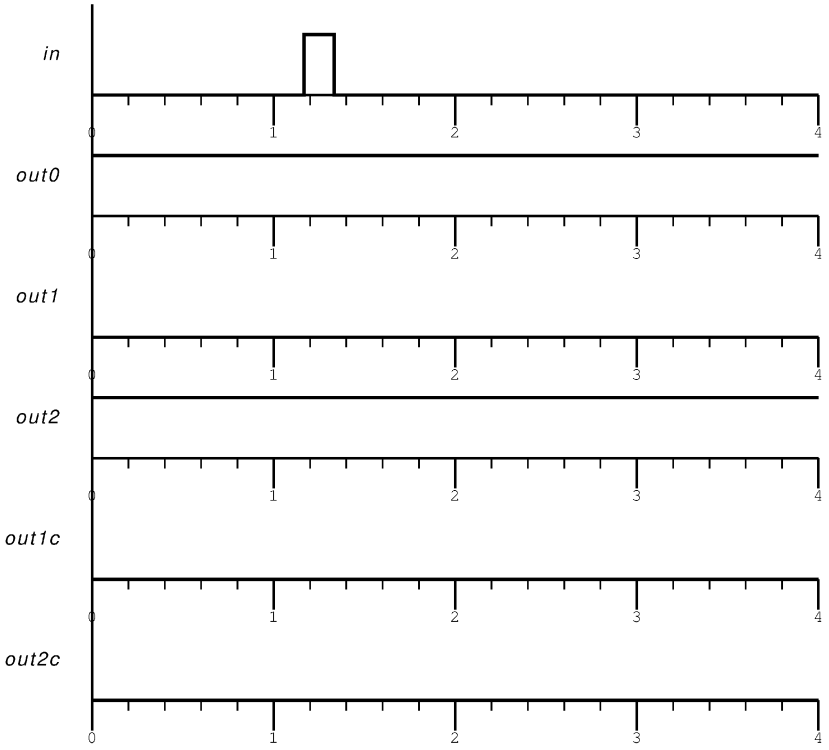


Figura 2.15. Simulación lógica con el modelo de retraso inercial.

Con este criterio para el efecto inercial, presentado en [JUAN00b], se incluye un nuevo parámetro V_T individual para cada entrada de cada puerta, el cual representa el umbral de voltaje asociado a la entrada. Con este nuevo tratamiento del efecto inercial es necesario manejar señales temporales y señales de voltaje. Por este motivo no es posible incluir este modelo en los actuales simuladores lógicos temporales y como consecuencia, existe considerable disminución en la precisión de los resultados obtenidos mediante simulación.

Este nuevo tratamiento del efecto inercial se combina con el modelo de degradación mostrado en la sección anterior, obteniendo como resultado el modelo que denominamos IDDM: *Inertial and Degradation Delay Model.*

2.4. Modelo IDDM, implementación en HALOTIS

HALOTIS, el entorno de simulación desarrollado en esta Tesis, implementa un motor de simulación a nivel de puertas, en decir, trata cada puerta como una caja negra y asigna valores específicos de los parámetros del modelo para cada entrada de una puerta dada. Los modelos de retraso implementados en HALOTIS a nivel de puertas incluyen: el efecto de la pendiente de entrada, la carga en los nodos, el efecto inercial y el efecto de degradación.

En este sentido, el modelo de retraso implementado en el simulador lógico es IDDM. Con este modelo, para una evaluación completa del retraso se precisan expresiones concretas para los componentes de la ecuación 2.6, esto es, t_{p0}, T_0 y τ, así como para el cálculo del tiempo de transición de las transición de salida (τ_{out}) indispensable para la simulación de etapas sucesivas del circuito y la aplicación del modelo inercial, siendo necesario disponer también del valor de tensión umbral, V_T, para cada entrada de cada puerta del circuito.

En una primera versión de HALOTIS las expresiones para el retraso de propagación normal (t_{p0}) eran la simplificación de las ecuaciones a nivel de puertas propuestas en [JUAN00a], que a su vez se obtenían de [DAGA98], al igual que las expresiones para τ_{out} :

$$t_{p0}=\frac{V_T}{2\,V_{DD}}\tau_{in}+\left(1+\frac{2C_M}{C_{out}+C_L}\right)t_{ps} \tag{2.8}$$

$$t_{ps}=k_1\frac{C_{out}+C_L}{C_{in}} \tag{2.9}$$

$$\tau_{out} = \begin{cases} 2t_{ps} & \tau_{in} \le t_{p0} \\ 2_{tps}\dfrac{1-\dfrac{V_T}{V_{DD}}}{\dfrac{1}{2}-\dfrac{V_T}{V_{DD}}+\dfrac{t_p}{\tau_{in}}} & \tau_{in} > 2t_{p0} \end{cases} \quad (2.10)$$

donde los parámetros característicos para cada entrada de una puerta y tipo de transición, son V_T, C_{in}, C_{out}, C_M y K_1. Este modelo con base analítica es difícil de caracterizar presentando escasas ventajas frente a otros heurísticos, utilizados finalmente en la versión actual y detallados en las dos secciones posteriores.

Por su parte, en [JUAN00a, JUAN00c] se extendía el modelo IDDM a puertas CMOS de más de una entrada, resultando, expresiones válidas para T_0 y τ:

$$\tau \times V_{DD} = A_x + B_x \times C_L \quad (2.11)$$

$$T_0 = (\frac{1}{2} - \frac{C_x}{V_{DD}}) \times \tau_{in} \quad (2.12)$$

donde A_x, B_x y C_x son característicos para cada entrada de la puerta y tipo de transición, y el subíndice x valdrá r o f según la transición sea ascendente o descendente respectivamente. El proceso de caracterización empleado para obtener el valor de estos parámetros puede consultarse en [JUAN00a], para el cual existe una herramienta que automatiza el proceso llamada AUTODDM [JUAN01b, JUAN01c].

2.4.1. Modelado heurístico del retraso de propagación normal

La implementación del modelo IDDM implica la caracterización del retraso de propagación normal (t_{p0}). El valor de t_{p0} depende principalmente del valor de la carga en el nodo de salida de la puerta (C_L) y, del valor de la pendiente de entrada de la transición (τ_{in}), tal y como se estudia analíticamente en [BISD98] y [DAGA99]. El modelo heurístico propuesto es simple y rápido en tiempo de computación y, además, suficientemente preciso en el rango de variación de C_L y τ_{in}.

Establecer un adecuado rango de variación ayuda tanto a la simplificación del modelo, como al proceso de caracterización. Respecto a C_L, el rango depende de la capacidad de las entradas de las puertas conectadas a la salida, se establecerá una variación desde $2 \times C_{IN}$ hasta $10 \times C_{IN}$, siendo C_{IN} la capacidad típica de entrada de un inversor en la tecnología utilizada. El rango de τ_{in} se calcula en función del retraso de propagación normal cuando el tiempo de la transición de entrada es

cero, llamado t_{ps}. El rango utilizado comienza en $0.1t_{ps}$ y termina en $10t_{ps}$ correspondientes a transiciones suficientemente rápidas y transiciones suficientemente lentas respectivamente.

Utilizando estos rangos en [MILLAN02] se obtienen conjuntos de puntos de t_{p0} mediante simulación con HSPICE en diferentes valores del rango, diferenciando transiciones de subida y transiciones de bajada. La figura 2.16 muestra un ejemplo de los puntos obtenidos con HSPICE en un inversor para transiciones de subida. Los puntos de la figura 2.16 se encuentran aproximadamente en un plano, volviéndose a repetir esta situación en todas las entradas para el conjunto de puertas lógicas analizadas. En [MILLAN02] se aproxima el plano por una ecuación lineal:

$$t_{p0}=D_{xi}\times C_L+E_{xi}\times\tau_{in}+F_{xi} \tag{2.13}$$

donde D_{xi}, E_{xi} y F_{xi} son los parámetros del modelo. Cada parámetro es obtenido para cada tipo de transición en la salida (indicado en el subíndice x, siendo este subíndice r para transiciones de subida y f para las de bajada) y para cada una de las entradas (indicado en el subíndice i).

La tabla 2.1 muestra resultados de diversas puertas en las que se han realizado ajustes mediante regresión lineal múltiple. Todas estas puertas son de tecnología CMOS 0.35μm y la tabla también muestra el error y la desviación de los diferentes ajustes. Consideramos el modelo suficientemente preciso, por ser el error del ajuste en todos los casos inferior al 4%.

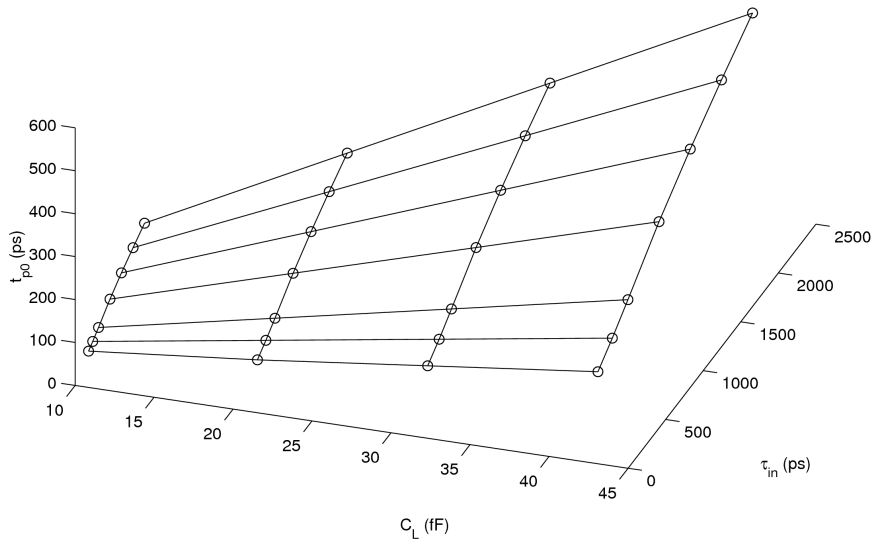


Figura 2.16. Puntos obtenidos con HSPICE para un inversor y transiciones de subida.

Puerta	Entrada	Caso	$D(ps/fF)$	E	$F(ps)$	$\overline{err}(ps)$	$\overline{dev}(ps)$
INV	1	Subida	4,27	0,180	30,0	4,6	1,74
INV	1	Bajada	3,57	0,100	31,5	6,8	3,75
NAND2	1	Subida	4,21	0,202	55,4	2,2	1,03
NAND2	2	Subida	4,16	0,213	93,1	3,0	1,20
NAND2	1	Bajada	2,77	0,083	42,4	3,9	3,27
NAND2	2	Bajada	2,75	0,019	61,1	3,8	3,34
NOR2	1	Subida	4,07	0,087	99,7	3,8	1,63
NOR2	2	Subida	3,95	0,160	43,9	2,6	1,24
NOR2	1	Bajada	3,61	0,135	114,9	4,8	1,58
NOR2	2	Bajada	3,47	0,125	59,7	4,6	2,29

Tabla 2.1. Caracterización del retraso de propagación normal usando HSPICE.

Una vez establecido el modelo lineal de t_{p0}, el proceso de caracterización ha sido incluido en AUTODDM, reduciéndose así en gran medida, el tiempo del proceso caracterización. Para minimizar el proceso se han reutilizado los datos generados por AUTODDM en la obtención de los parámetros D_{xi}, E_{xi} y F_{xi}. Esto ha sido posible ya que AUTODDM realiza dos grupos de simulaciones: uno para un conjunto de valores de C_L fijando el valor de τ_{in} y el segundo, para un conjunto de valores de τ_{in} fijando el valor de C_L. Estos conjuntos de puntos corresponden a dos líneas del plano de la figura 2.16, y por tanto es posible calcular el plano y extraer los parámetros. En [MILLAN02] se comprueba que el error en esta aproximación tampoco supera el 4% en ninguna de las puertas estudiadas.

2.4.2. Modelado heurístico de la pendiente de salida

En las puertas CMOS estáticas el factor principal que afecta a la variación de la pendiente de salida, en el rango típico de operación, es el valor de la carga en el nodo de salida C_L. El modelo heurístico para la pendiente de salida incluido en HALOTIS es:

$$\tau_{out} = G_x \times C_L + H_x \tag{2.14}$$

donde G_X y H_X son los parámetros a caracterizar mediante regresión lineal y el subíndice x indica si las transiciones son de subida o bajada (r ó f respectivamente). El proceso de caracterización es simple y se realiza en paralelo durante la ejecución de la caracterización del retraso de propagación normal expuesto en la sección anterior.

Puerta	Entrada	G_R(s/F)	H_R(s)	$Error_R$	G_F(s/F)	H_F(s)	$Error_F$
BUFFER	1	9974.26	$8.6876 \cdot 10^{11}$	0.999966	7011.94	$6.9397 \cdot 10^{11}$	0.999985
INV	1	9916.13	$8.86367 \cdot 10^{11}$	0.999957	7005.88	$7.01589 \cdot 10^{11}$	0.999978
EXOR	1	9861.7	$2.77883 \cdot 10^{10}$	1	7060.74	$1.40968 \cdot 10^{10}$	0.999994
	2	10072.3	$1.81501 \cdot 10^{10}$	0.999992	7064.72	$1.40092 \cdot 10^{10}$	0.999994
NAND2	1	10340.8	$1.33747 \cdot 10^{10}$	1	6054.05	$8.64447 \cdot 10^{11}$	0.999939
	2	10319.5	$2.15881 \cdot 10^{10}$	1	6012.03	$8.8294 \cdot 10^{11}$	0.999977
AND2	1	9820.06	$1.06063 \cdot 10^{10}$	0.999993	7468.03	$9.40487 \cdot 10^{11}$	0.999851
	2	9816.81	$1.06125 \cdot 10^{10}$	0.999991	7333.44	$1.12689 \cdot 10^{10}$	0.999736
NOR2	1	9612.79	$1.80261 \cdot 10^{10}$	0.999975	13265.9	$4.20325 \cdot 10^{10}$	0.999612
	2	10216	$1.56534 \cdot 10^{10}$	0.999992	14838.6	$1.78293 \cdot 10^{10}$	1
AND3	1	10006.6	$1.16891 \cdot 10^{10}$	0.99997	7544.86	$1.06065 \cdot 10^{10}$	0.999822
	2	10000.8	$1.17299 \cdot 10^{10}$	0.999969	7415	$1.32687 \cdot 10^{10}$	0.999514
	3	9993.7	$1.17387 \cdot 10^{10}$	0.999971	7566.72	$1.57151 \cdot 10^{10}$	0.999068
NAND3	1	10350.6	$4.24161 \cdot 10^{10}$	1	12423	$3.45625 \cdot 10^{10}$	0.999991
	2	10337.3	$3.07808 \cdot 10^{10}$	1	12426	$3.45645 \cdot 10^{10}$	0.999988
	3	10343.8	$1.90983 \cdot 10^{10}$	1	12661.3	$3.34477 \cdot 10^{10}$	0.999988
NAND4	1	9388.37	$1.2003 \cdot 10^{10}$	0.999996	14786.2	$1.53698 \cdot 10^{10}$	0.999972
	2	9401.13	$1.19501 \cdot 10^{10}$	0.999998	14793.8	$1.53721 \cdot 10^{10}$	0.999967
	3	9537.48	$1.02115 \cdot 10^{10}$	0.999992	14871	$1.48975 \cdot 10^{10}$	0.999968
	4	9567.24	$1.00845 \cdot 10^{10}$	0.999995	14880.9	$1.48919 \cdot 10^{10}$	0.999957
NAND8	1	9636.93	$1.50529 \cdot 10^{10}$	0.999987	15277.6	$1.89781 \cdot 10^{10}$	0.999994
	2	9626.87	$1.50611 \cdot 10^{10}$	0.999989	15281.3	$1.89761 \cdot 10^{10}$	0.999993
	3	9605.95	$1.51123 \cdot 10^{10}$	0.999989	15276.1	$1.89827 \cdot 10^{10}$	0.999994
	4	9380.29	$1.29299 \cdot 10^{10}$	0.999991	15341.9	$1.86333 \cdot 10^{10}$	0.999993
	5	9366.13	$1.29799 \cdot 10^{10}$	0.999992	15343.4	$1.86294 \cdot 10^{10}$	0.999991
	6	9561.44	$1.07488 \cdot 10^{10}$	0.999998	15657.6	$1.66659 \cdot 10^{10}$	0.999971
	7	9525.49	$1.09161 \cdot 10^{10}$	0.999997	15648	$1.66777 \cdot 10^{10}$	0.999973
	8	9492.42	$1.11168 \cdot 10^{10}$	0.999997	15661.3	$1.66625 \cdot 10^{10}$	0.999972

Tabla 2.2. Caracterización de la pendiente de salida para diversas puertas.

La caracterización con este modelo de un conjunto de puertas en tecnología CMOS 0.35μm se incluye en la tabla 2.2, donde el ajuste en el rango estudiado (usando el mismo rango establecido que para t_{p0}) lo consideramos suficiente.

2.5. Aplicación del modelo de DDM a celdas CBL

Las celdas CBL (*Current Balanced Logic*) se proponen como una familia de celdas lógicas de bajo ruido de conmutación [ALBU99b]. Estas celdas reducen los picos de intensidad respecto a las celdas convencionales CMOS [ALBU99a], siendo una alternativa para el diseño de circuitos de señal mixta (analógico/digital) donde

el ruido de conmutación es un factor decisivo y debe ser reducido.

La figura 2.17a muestra la estructura de una celda CBL genérica, donde la celda es obtenida añadiendo el transistor M_3 a la parte lógica pseudo-NMOS. Esta familia lógica presenta las siguientes mejoras respecto a otras de bajo ruido de conmutación [ALBU99b]:

- Construcción simple de celdas, celdas con *N* entradas tienen *N+2* transistores.
- Tensión baja de alimentación, por tanto, reduce en consumo de potencia y el ruido en los nodos internos.
- Compatibilidad con las celdas convencionales CMOS, tanto estáticas como dinámicas.

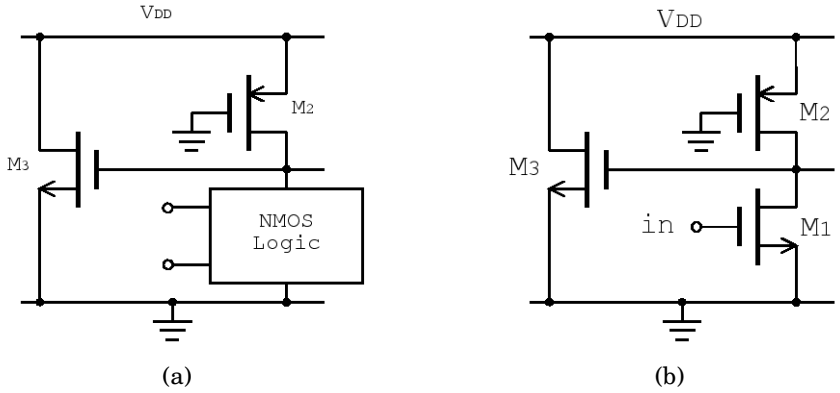


Figura 2.17. Estructura de las celdas CBL: a) Estructura genérica, b) Inversor.

Se utilizará el inversor CBL de la figura 2.17b para realizar la caracterización con el modelo DDM y la posterior simulación en el capítulo 5. Respecto a la caracterización con AUTODDM, en la tabla 2.3 se muestran los parámetros de degradación obtenidos, los errores relativos y los coeficientes de correlación. Tras la caracterización se concluye que el inversor CBL se ajusta con bastante precisión a las ecuaciones de modelado de degradación incluidas en HALOTIS. También para el modelo de retraso de propagación normal (ecuación 2.13) estudiado en [MILLAN02], se obtienen un buen ajuste en los resultados de caracterización con AUTOTPN y, mostrados en la tabla 2.4.

Transiciones de bajada	**Transiciones de subida**
A = 548,128 ps/v	A = 200,075 ps/v
B = 27,3973 f fps/v	B= 12,32 f fps / v
C = 2,24878 vFf	C = 0,2349 vFf
Corr = 0,999412	Corr = 0,99991

Tabla 2.3. Resultados de caracterización del inversor CBL con AUTODDM.

Transiciones de bajada	Transiciones de subida
D = 11748,2 sF	D = 20485,6 sF
E = 0,316018 f fps	E = 12,32 f fps/v
F = 2,24878 vFf	F = 0,2349 vFf
Corr = 0,999994	Corr = 0,999988

Tabla 2.4. Resultados de la caracterización de t_{p0} en el inversor CBL.

2.6. Algoritmo de simulación para IDDM.

Una vez estudiados los efectos a considerar, hay que plantearse la necesidad de utilizar un algoritmo de simulación capaz de incluirlos. Previamente se comentó la necesidad del tratar con el conjunto tiempo-voltaje para la inclusión del modelo IDDM en un simulador. Así nace una nueva forma de tratar con estímulos en la herramienta de simulación, distinguiendo por tanto dos conceptos: transición y evento [RUIZ01a].

Una transición será una señal cambiando de 0 a 1 ó de 1 a 0. Será aproximada por un segmento, quedando ésta determinada por el tiempo de bajada o subida τ_x y, el instante en el que la transición comienza t_0. El valor de x, según el tipo de transición, valdrá f ó r resultando: τ_r para una transición de subida y τ_f en el caso de una transición de bajada, tal y como se muestra en la figura 2.18.

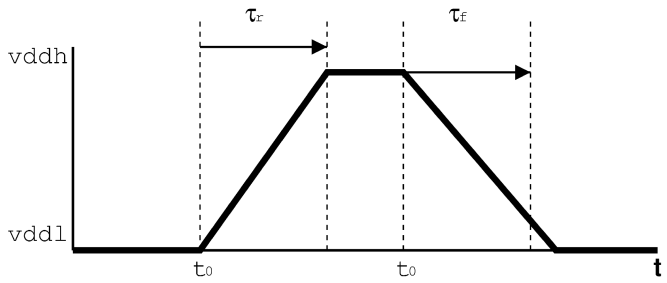


Figura 2.18. Ejemplo de transición.

El concepto de transición permite contemplar el hecho de tener umbrales de activación individuales en cada una de las entradas de las diferentes puertas. La primera consecuencia de la inclusión de este nuevo parámetro es la obtención de eventos a partir de una transición en diferentes instantes de tiempo. Con este modelo, un evento se genera únicamente cuando una transición cruza el umbral de tensión de una determinada entrada, tal y como se muestra en la figura 2.19. Ésta figura presenta el tratamiento que se realiza con las transiciones. Se puede concluir que a partir de una transición se calculan los correspondientes eventos generados en las diferentes entradas de las distintas puertas.

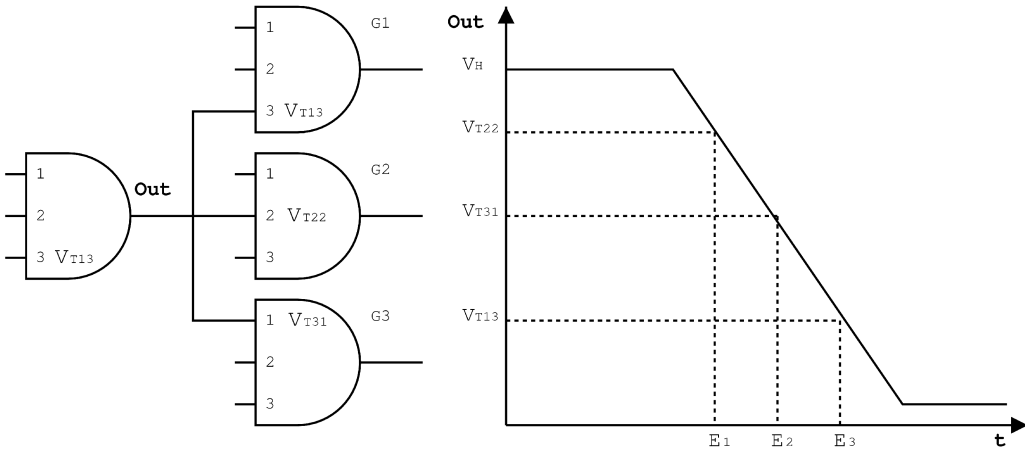


Figura 2.19. Generación de eventos a partir de transiciones.

Los elementos obtenidos durante el proceso de simulación de la figura 2.19 se incluyen en la tabla 2.5, como un conjunto de eventos generados a partir a una única transición. Los eventos y las transiciones, durante el proceso de simulación, serán almacenadas en diferentes estructuras de datos debido a su diferente naturaleza. Para los eventos, el tratamiento es similar al realizado en los algoritmos de simulación tradicionales, es decir, formarán parte de una cola de eventos ordenada en el tiempo. En el caso de las transiciones, son elementos temporales durante el proceso de simulación, por ello, tras su procesado pueden ser eliminadas. Podrían ser almacenadas para alguna determinada línea de interconexión consiguiendo así representar la forma de onda existente. Es necesario en este caso usar una estructura de datos de tipo lista en la que se almacenen estas transiciones en el orden de creación.

Transición	Evento	Puerta	Umbral
τ_f y t_0	E1	G2	V_{T22}
	E2	G3	V_{T31}
	E3	G1	V_{T13}

Tabla 2.5. Eventos generados tras una transición.

Una vez visto el tratamiento de estímulos, ya es posible establecer el algoritmo de simulación para el modelo IDDM [BELL00]. Los pasos de este algoritmo, mostrados de forma gráfica en figura 2.20, se tratan desde un punto de vista de alto nivel. Para una mejor compresión se procederá a realizar una descripción a continuación.

En primer, lugar el proceso de simulación debe obtener el primer evento de la denominada cola de eventos. Para realizar el correcto procesado del mismo, es necesario aportar datos sobre el evento como son: la entrada de la puerta en el que se produce y la transición que lo generó. Tras esto se comprueban todas las entra-

das de celdas afectadas por la transición. En esta comprobación se determina para cada entrada de puerta afectada:

1. Si produce un evento en la entrada.
2. En caso de producir evento, se evalúa la posibilidad de la existencia de efecto inercial.

Para una determinada entrada de una puerta, se comprueba si el evento generado es anterior al último encolado en esta puerta. En este caso, el nuevo evento no es encolado y el existente es eliminado. El proceso vuelve a comenzar mientras se tengan eventos en la cola de eventos.

El mecanismo de filtrado descrito en el apartado 2.3 tratado se muestra en la figura 2.21. En la figura se observa como tras la aparición de la primera transición se producen dos eventos: uno en la puerta 2 y otro en la puerta 1.

Con estos datos se tiene la información necesaria para calcular la transición de salida generada por este evento (parámetros de la puerta, capacidad de salida, tiempo de transición de la entrada, etc.). Una vez obtenida la correspondiente transición de salida, se evalúan posibles eventos generados por la misma puerta conectada al nodo donde se produce la transición generada, de la forma que se describió en la figura 2.19. Ahora se entra en un bucle para cada evento generado encaminado a determinar si se produce efecto inercial de forma individualizada en cada una de las puertas afectadas por la transición. Para ello se compara el instante del evento actual (E_j) con el instante del último evento encolado para esa entrada (E_{j-1}).

Como puede verse en la figura 2.21, si $E_j > E_{j-1}$ significa que la transición actual ha cruzado el umbral lógico de la puerta (V_{T2}) y por tanto puede producir una activación de la misma, por tanto, este evento se encola en la cola de eventos para su procesamiento posterior. En cambio, si $E_j < E_{j-1}$ (caso de la puerta 1 en la figura 2.21) significa que la transición actual, combinada con la transición anterior en ese nodo, produce un pulso de amplitud insuficiente para alcanzar el umbral de la puerta (V_{T1}) y por tanto no la activa. En este caso, el nuevo evento no es encolado y el evento anterior, E_{j-1}, es eliminado de la cola de eventos, produciéndose efecto inercial para esa puerta. Cuando se han analizado todos los posibles eventos generados por la transición actual, se comprueba si quedan eventos pendientes que procesar en la cola de eventos y en caso negativo se da por finalizada la simulación.

Una descripción más detallada sobre este nuevo algoritmo para el tratamiento del efecto inercial puede consultarse en [JUAN99].

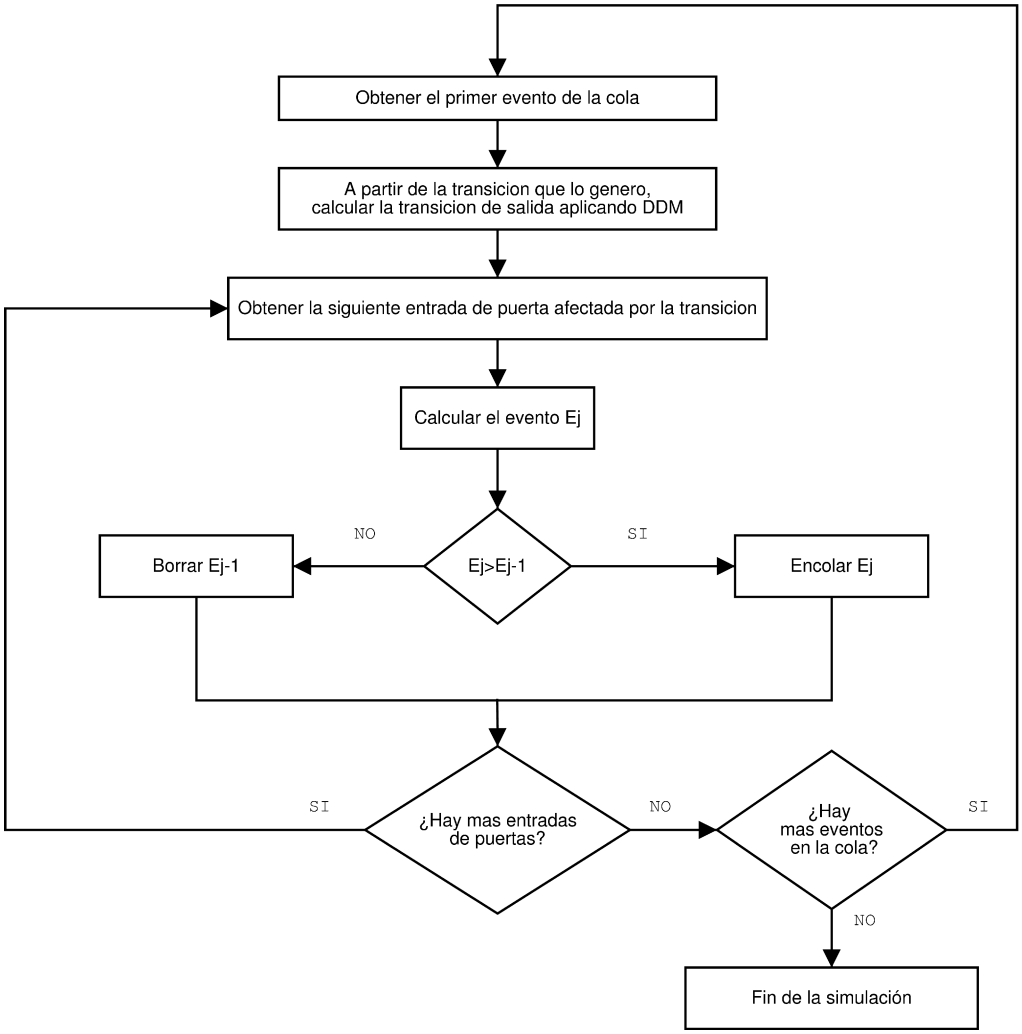


Figura 2.20. Algoritmo de simulación.

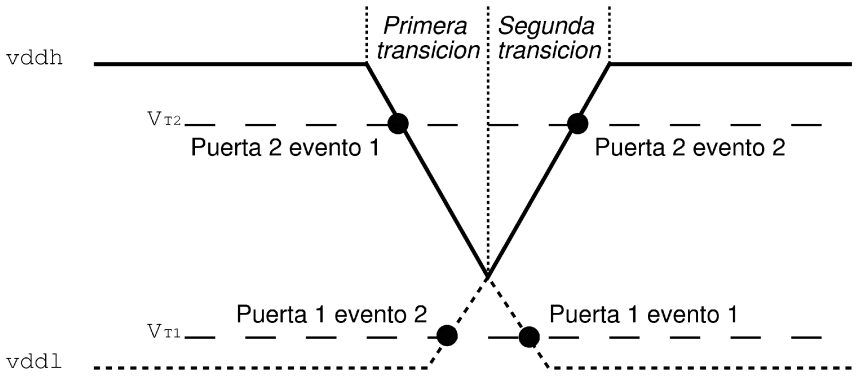


Figura 2.21. Filtrado de eventos

2.7. Conjunto de parámetros necesarios para la simulación lógica temporal en HALOTIS

Finalmente, esta sección presenta un resumen de los modelos utilizados en el motor de simulación lógico-temporal. El modelo IDDM implementado en HALOTIS contempla:

- Retraso de propagación normal implementado con la ecuación 2.13.
- Nuevo modelo inercial basado en tensiones umbrales.
- Modelo de degradación basado en las ecuaciones 2.11 y 2.12.
- Modelo de pendiente de salida, ecuación 2.14.

La tabla 2.6 presenta en modo de resumen los parámetros que HALOTIS debe disponer para cada combinación entrada/salida de cada puerta del circuito digital, así como la ecuación y el modelo que en el que son utilizados.

Parámetros	**Modelo**	**Ecuación**	**Página**
V_T	Inercial	-	40
A_r, A_f, B_r, B_f	Degradación	2.11	44
C_r, C_f	Degradación	2.12	44
D_r, D_f, E_r, E_f, F_r, F_f	Propagación normal	2.13	45
G_r, G_f, H_r, H_f	Pendiente de salida	2.14	46
C_{IN}	Propagación normal, pendiente de salida y, degradación.	2.11, 2.13 y 2.14	

Tabla 2.6. Resumen de parámetros para la simulación lógica temporal en HALOTIS.

A estos modelos y conjuntos de parámetros hay que añadir los provenientes del modelado de la intensidad desarrollado en el siguiente capítulo.

Capítulo 3. Modelo de estimación de intensidad y potencia

Hoy en día adquiere gran importancia el diseño de circuitos de bajo consumo de potencia haciendo relevante el desarrollo de herramientas CAD de estimación precisas, tanto del consumo de potencia como de intensidad.

Los simuladores lógico-temporales vienen usándose como herramientas para proporcionar un parámetro básico como es la actividad de conmutación y así realizar una estimación precisa de potencia e intensidad. Una de las principales aplicaciones del modelo DDM es la posibilidad de aportar una alta precisión en el cálculo de la actividad de conmutación.

Por ello, se pretende aprovechar esta capacidad del modelo para convertir al simulador HALOTIS en una herramienta no sólo de simulación temporal sino también de estimación de potencia e intensidad. Para ello se ha desarrollado un

modelo de intensidad que, unido a las capacidades del modelo DDM, da lugar a una herramienta de gran precisión en la estimación de consumos. Este capítulo describe este modelo.

Así, el capítulo comienza con una introducción para, posteriormente, presentar el modelo de intensidad aplicado al inversor CMOS. A continuación se presenta un método de caracterización utilizado para validarlo en otras puertas lógicas. Finalmente, se detallan algunos aspectos relevantes de la implementación del modelo en un simulador lógico temporal.

3.1. Introducción

La estimación de intensidad y potencia en la verificación de circuitos VLSI juega cada vez un papel más importante, motivado principalmente por la estimación del ruido de conmutación y el consumo de potencia. Al trabajar en niveles altos de diseño (lógico, RT, arquitectural) la pérdida de precisión en estas medidas es considerablemente alta. Solventar este problema requiere, además de un buen modelo de estimación de intensidad, un buen modelo de estimación de actividad de conmutación.

La verificación digital en los circuitos integrados VLSI tiene lugar a diferentes niveles de diseño: layout, nivel lógico, arquitectural y nivel de sistemas. Debido al incremento en la escala de integración de las últimas dos décadas, se tiende a procesos de verificación en los niveles más altos, nivel de sistemas o incluso nivel del software [ARTS03, SEEN04, LATTA04]. Actualmente adquieren gran importancia estimaciones de corriente, energía, potencia y actividad de conmutación, aplicados en áreas de diseño de circuitos de señal mixta [JIME02], procesadores de alto rendimiento y sistemas de bajo consumo para operar con baterías [ABBO04]. En los circuitos de señal mixta el ruido de conmutación es un factor clave en el diseño, mientras que en el diseño de procesadores de alto rendimiento la disipación de energía presenta dificultades. En este escenario la simulación eléctrica se limita a la optimización de bloques básicos y/o críticos, jugando la verificación a nivel lógica un papel decisivo hasta el nivel de sistemas.

Para afrontar los nuevos desafíos citados, los entornos EDA (*Electronic Design Automation*) han incorporado herramientas para la estimación de corriente y potencia a sus simuladores de nivel lógico (PRIMEPOWER [SYNOPSYS], XPOWER [XILINXb], POWERTOOL [VERI]). Estas herramientas realizan la estimación de corriente/potencia a nivel lógico obteniendo los resultados en dos fases: (1) mediante la simulación lógica se procesa la actividad de conmutación en cada nodo del circuito; (2) la potencia en cada nodo es calculada utilizando modelos de carga en el nodo. En cambio, hay trabajos [BOGLI97, NIKO99] en los que se

consigue generar la curva de intensidad durante la simulación lógica utilizando modelos que pueden ser incluidos en la simulación guiada por eventos.

Este capítulo presenta un modelo de estimación de la curva de intensidad a nivel de puertas lógicas, basado en la obtención del cálculo de la intensidad para cada transición y aplicable durante la simulación lógica. Un primer modelo fue presentado en [RUIZ02] donde se aproximaba la intensidad en cada transición por un escalón de altura proporcional a la carga del nodo y ancho igual al tiempo de transición. A pesar de la simplicidad, los resultados obtenidos concluían que la inclusión un modelo de corriente en un simulador lógico era posible obteniéndose resultados con un error por debajo del 20%.

En [BOGLI97] se propone un modelo de corriente y potencia a nivel lógico, el cual aproxima la curva de intensidad por una forma de onda triangular cuyo vértice superior (pico de intensidad) se aproxima de forma heurística. El modelo aquí presentado aproxima también la curva intensidad en cada celda lógica por un triángulo. Las diferencias con [BOGLI97] radican en la simplificación de la caracterización, de la simulación y de la precisión del cálculo de los puntos del triángulo. La aproximación de los vértices del triángulo se realiza en base a los trabajos [RABE98], [TURGIS98], [BISD99], [DAGA99] y [MAUR01] que realizan estudios analíticos de la intensidad de cortocircuito en las celdas CMOS estáticas, consiguiéndose modelar la curva de intensidad con alta precisión y con un reducido número de parámetros.

A continuación se presenta el modelo de intensidad utilizando un inversor CMOS como ejemplo.

3.2. Modelo de intensidad

La estimación del consumo de potencia y/o intensidad con un simulador lógico se basa en conocer la actividad de conmutación en los diferentes nodos y, para dicha actividad de conmutación, asociarle de alguna forma un valor de potencia o intensidad consumida.

El planteamiento realizado en HALOTIS consiste en ser capaz de obtener directamente como resultados de la simulación del comportamiento del circuito, curvas de intensidad consumida frente al tiempo de operación. El cálculo global de intensidad se obtiene como una suma de las intensidades en cada uno de los nodos del circuito. Por tanto, para obtener esta curva de intensidad se necesita un modelo de cálculo de intensidad para cada transición de salida de una puerta lógica.

El estudio del modelo de intensidad propuesto se realiza utilizando un inver-

sor, no perdiendo generalidad, ya que es el caso al que se reduce el estudio de puertas de más entradas cuando sólo una de ellas está activa. Un primer análisis en continua del inversor CMOS con HSPICE se presenta en la figura 3.1, el cual representa la tensión de salida frente a la tensión de entrada y, además, tiene añadida la intensidad de la fuente con los valores escalados para poder incluirla en la misma figura.

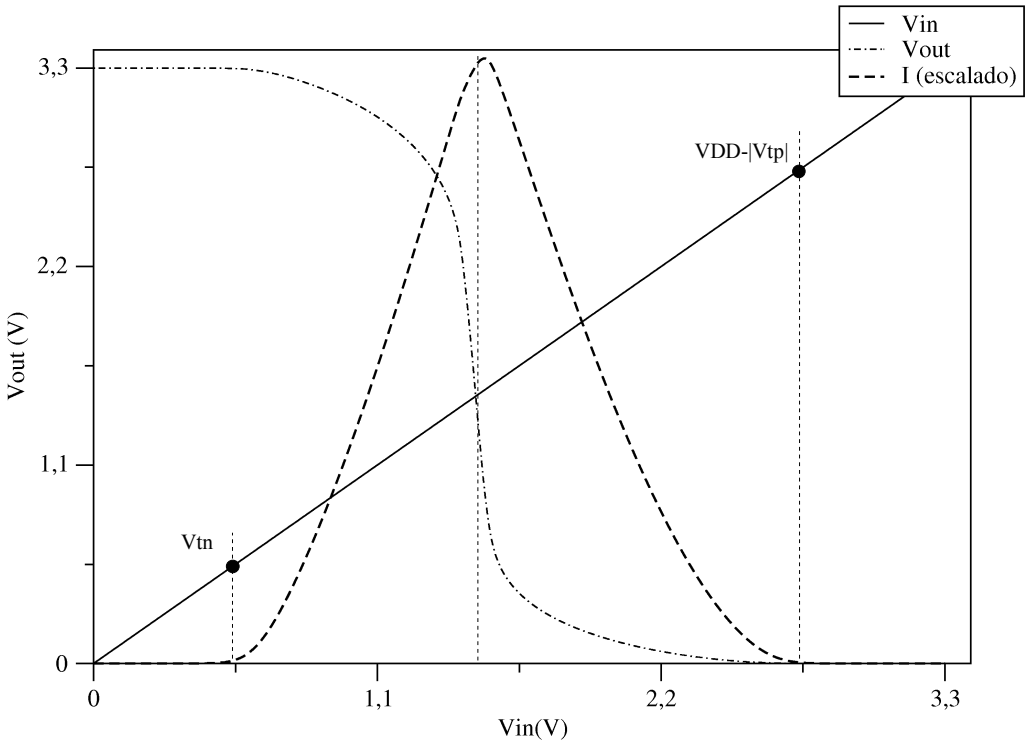


Figura 3.1. Análisis DC de un inversor CMOS.

Tal y como se ha planteado anteriormente, el consumo de intensidad está ligado a la existencia de conmutaciones en los nodos de salida de las puertas. En la figura 3.1 se observa que la circulación de intensidad comienza cuando la tensión en el nodo de entrada alcanza los valores umbrales de los transistores NMOS o PMOS (V_{TP} y V_{TN} respectivamente) según la transición sea de subida o bajada. La intensidad estará circulando hasta que el nodo de salida haya sido completamente descargado (o cargado en el caso de entradas alto-bajo). De hecho se pueden establecer las siguientes condiciones generales de circulación de intensidad en un inversor CMOS como el de la figura 3.2a:

1. En transiciones de entrada bajo-alto la intensidad empezará a circular cuando $V_{in}=V_{GSN}>V_{TN}$ y, en transiciones alto-bajo cuando $V_{in}<V_{DD}-|V_{TP}|$.

2. Durante el intervalo de tiempo donde que $V_{in}>V_{TN}$ y $V_{in}<V_{DD}-|V_{TP}|$, circulará tanto intensidad de corto circuito (I_{SC}), como intensidad de carga/descarga.

3. En transiciones bajo-alto, a partir del instante de tiempo donde V_{in} alcanza $V_{DD}-|V_{TP}|$ se corta el transistor PMOS circulando exclusivamente intensidad de descarga. En transiciones de entrada alto-bajo se cortará el transistor NMOS cuando $V_{in}=V_{TN}$, quedando sólo intensidad de carga (I_{CL}) (figura 3.2a).

Este comportamiento básico del inversor CMOS es generalizable a puertas CMOS de más de una entrada, puesto que la estructura básica de estas puertas está basada en el par de transistores complementarios (NMOS-PMOS) para cada una de las entradas (figura 3.2b).

Una vez establecido el comportamiento básico, se establece un modelo cuantitativo de la intensidad. El modelo desarrollado en este trabajo está basado en el modelo analítico de [TURGIS98] y [DAGA99] pero, para alcanzar cotas de precisión, se ha establecido una aproximación heurística de dicho modelo.

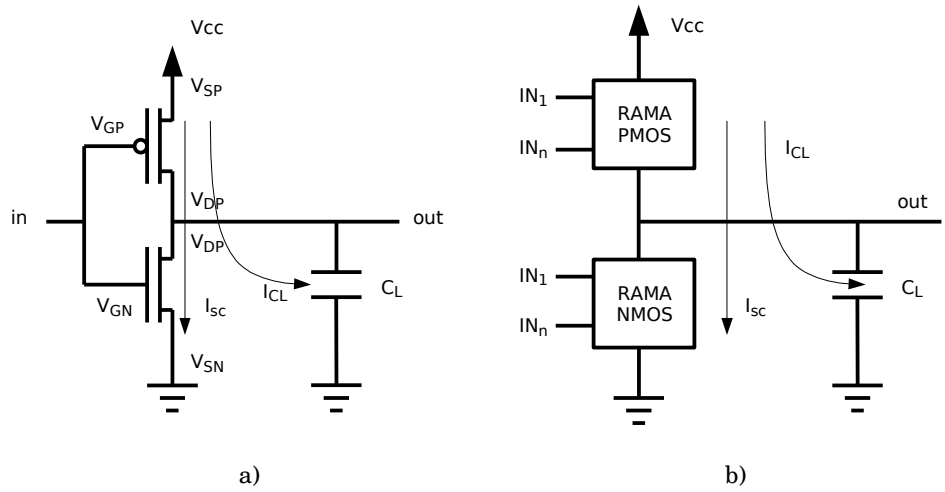


Figura 3.2. a) Intensidades en el inversor CMOS durante la conmutación lógica. b) Estructura genérica para puertas CMOS de n entradas.

3.2.1. Análisis heurístico del consumo de intensidad en el inversor CMOS

En el modelo analítico de [TURGIS98] y [DAGA99] se distinguen tres zonas: zona sobredisparo, zona de cortocircuito y zona de carga/descarga.

La mayor parte de la aportación de corriente desde la fuente de alimentación tiene lugar en dos zonas: (1) zona de cortocircuito y (2) zona de carga/descarga. En la figura 3.2 se propone un modelo simplificado con las intensidades

principales cuando una puerta CMOS conmuta hacia un valor alto en la salida, estas son: (1) la intensidad que carga el nodo de salida I_{CL} y, (2) la intensidad de cortocircuito I_{SC} que circula directamente desde V_{CC} a *GND* mientras la rama NMOS esté conduciendo. En esta simplificación, el valor de intensidad aportada por la fuente se obtiene como la suma de I_{CL} e I_{SC}. Para el caso de transiciones de bajada en la salida, la capacidad C_L se descarga y la única intensidad que circula desde la fuente es I_{SC}, quedando este valor como la intensidad total aportada desde la fuente. La intensidad total que circula desde la fuente depende fundamentalmente de la pendiente de la transición de entrada (τ_{in}) y de la capacidad de la salida (C_L) [BOGLI97, NIKO99].

En las figuras 3.3, 3.4, 3.5 y 3.6 se representan las curvas de intensidad de un inversor CMOS para diferentes valores de C_L (figuras 3.3 y 3.5), diferentes valores de τ_{in} (figuras 3.4 y 3.6), para transiciones de subida (figuras 3.3 y 3.4) y, transiciones de bajada (figuras 3.5 y 3.6).

Respecto a la variación con C_L, se han obtenido curvas de cuatro valores de C_L múltiplos de la capacidad de entrada del inversor, calculado con la siguiente ecuación:

$$C_{in}=(W_P+W_N)\times C_{ox}\times L_{min} \tag{3.1}$$

donde W_P y W_N son los anchos de los transistores PMOS y NMOS respectivamente, L es la longitud del transistor MOS y C_{OX} es la capacidad del óxido por unidad de área. Concretamente, en la tecnología empleada (0.35μm de AMS) C_{in} es 4.035*fF*. Este valor se utilizará como valor característico de esta tecnología.

Respecto a la variación de la pendiente de entrada, se han considerado cuatro casos correspondientes a *fan-in* 1 hasta *fan-in* 4 donde cada *fan-in* es la capacidad equivalente de un inversor CMOS y la intensidad de carga del nodo está proporcionada otro inversor CMOS. Esto da lugar a variaciones en la pendiente de entrada desde 200ps hasta 800ps en la mencionada tecnología.

En todas las gráficas se observa una forma de onda muy semejante a un triángulo, esto es, a partir del momento en que empieza a circular intensidad (V_{in}=V_{TN}) va aumentando prácticamente de forma lineal hasta alcanzar un máximo, a partir del cual comienza a descender también de forma lineal.

Sin embargo, se observan diferencias significativas en las dependencias con C_L y τ_{in} para las transiciones de salida de subida (figuras 3.3 y 3.4) y de bajada (figuras 3.5 y 3.6). En el caso de las primeras, respecto a la dependencias con C_L se observa como la pendiente durante el aumento de intensidad y la pendiente de la bajada de intensidad son prácticamente las mismas, cambiando el valor de intensidad máximo que se alcanza en los diferentes casos, el cual, aumenta con el valor de C_L. Respecto a la dependencia con τ_{in}, tanto el instante donde comienza a

circular intensidad como la pendiente durante el aumento varían con τ_{in}. Además, mientras mayor es el valor de τ_{in} se alcanza un valor menor de intensidad máxima. En cambio, la pendiente de disminución de la intensidad es prácticamente la misma.

En el caso de las transiciones de bajada hay que destacar como la magnitud de la intensidad es mucho menor que en el otro tipo de transiciones, debido a que sólo existe I_{SC}. Con C_L, la variación no es muy grande ni en el valor de intensidad máxima alcanzada, ni en el valor en que la intensidad deja de circular.

Respecto a τ_{in} es significativa la disminución del pico de intensidad máxima y, sobre todo, un desplazamiento de la pendiente de bajada de la intensidad. Esto último se justifica, ya que al ser τ_{in} mayor se tarda más tiempo en alcanzar el instante de corte del transistor NMOS ($I_{SC}=0$).

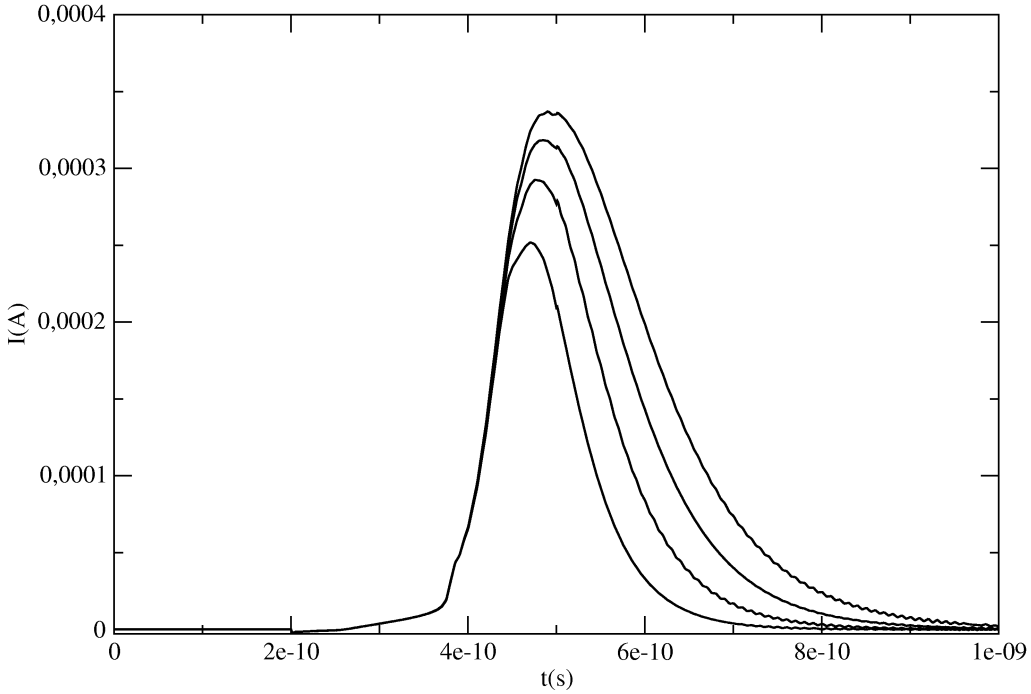


Figura 3.3. Intensidades en el inversor CMOS variando C_L desde C_{in}×1 hasta C_{in}×4 cuando el nodo se salida se carga.

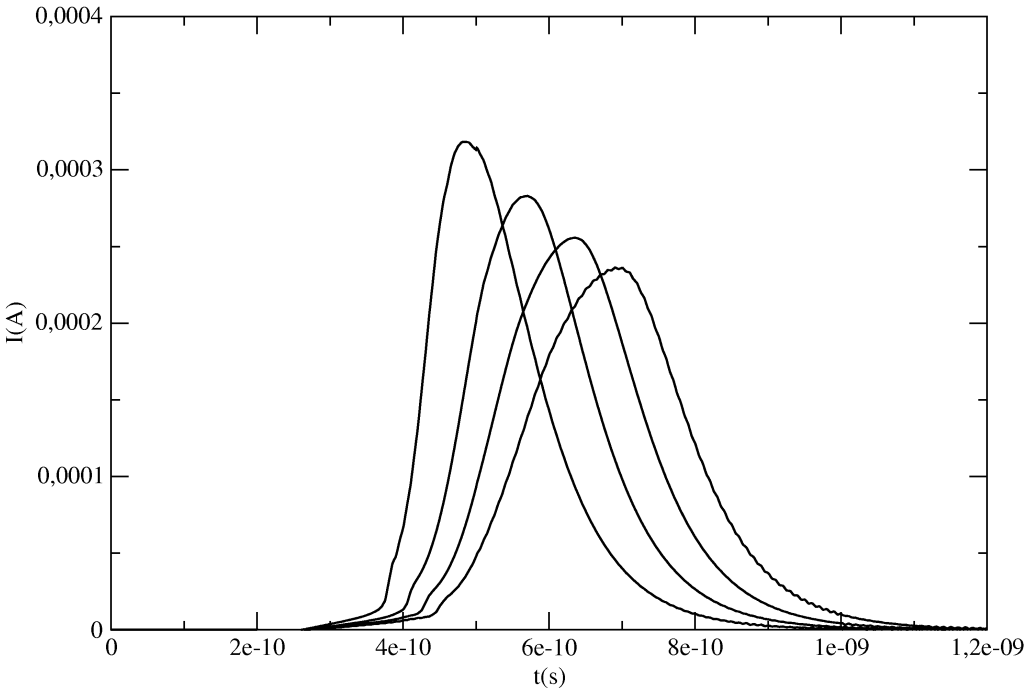


Figura 3.4. Intensidades en el inversor CMOS con variaciones de τ_{in} cuando el nodo de salida se carga.

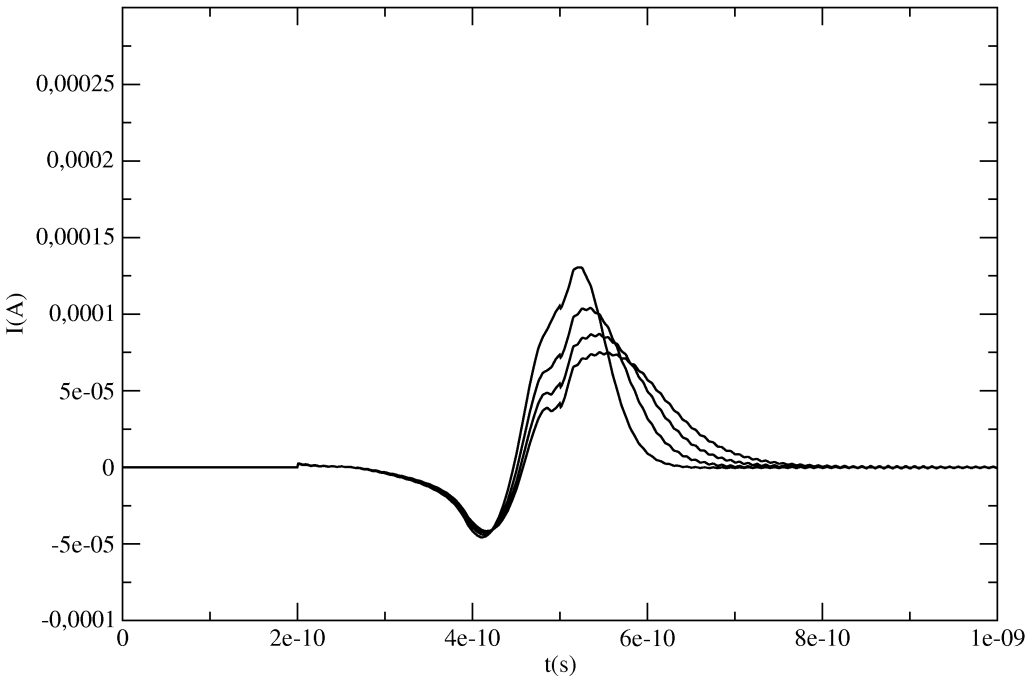


Figura 3.5. Intensidades en el inversor CMOS variando C_L desde C_{in}×1 hasta C_{in}×4 cuando el nodo de salida se descarga.

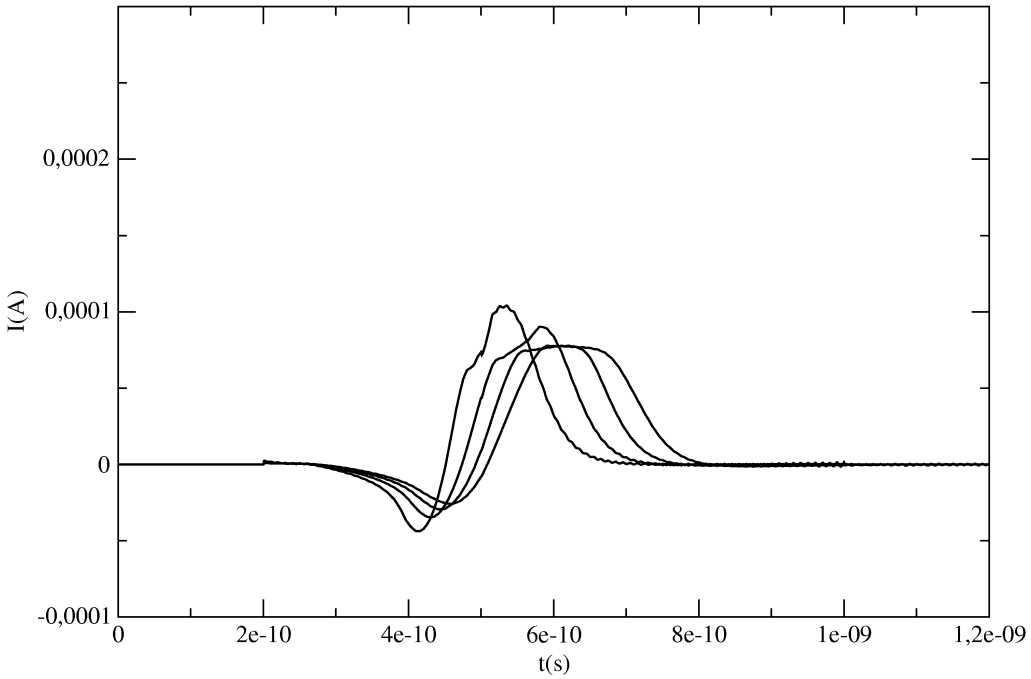


Figura 3.6. Intensidades en el inversor CMOS con variaciones de τ_{in} cuando el nodo de salida se descarga.

3.2.2. Modelo de intensidad para simulación lógica

El objetivo en esta sección es desarrollar un modelo cuantitativo de obtención de la curva de intensidad capaz de ser aplicado a la simulación lógica temporal. El procedimiento seguido es el siguiente:

- Establecer la forma característica de la curva de intensidad para cada transición de salida de una puerta lógica.
- Determinar los parámetros que caracterizan la forma de onda de intensidad.
- A partir de modelos analíticos propuestos por otros autores junto con el análisis cualitativo de la sección anterior, establecer las condiciones y ecuaciones que determinan el valor de los parámetros característicos de la curva de intensidad.

Comenzando por la forma de onda característica de la curva de intensidad, tanto por el análisis de las curvas reales como por los resultados de trabajos de otros autores como [BOGLI97], existe una solución idónea que consiste en modelar la curva de intensidad por una curva triangular. Esta curva triangular se adapta perfectamente a las características de la simulación lógica *event-driven*, tal y como

se observa en las figuras 3.7 y 3.8.

En estas figuras se representa, para un inversor CMOS, como se modela una transición real que provoca un cambio en la salida (figuras 3.7a y 3.8a) para adaptarlos a la simulación lógica. El criterio consiste fundamentalmente en linealizar las diferentes formas de onda (transición de entrada, transición de salida y curva de intensidad). De esta forma, sólo es necesario determinar unos pocos parámetros característicos de cada forma de onda a través de unas ecuaciones que surgen del modelo de comportamiento. Además, en las figuras 3.7 y 3.8 se observa como la curva triangular es una aproximación muy buena de la curva de intensidad.

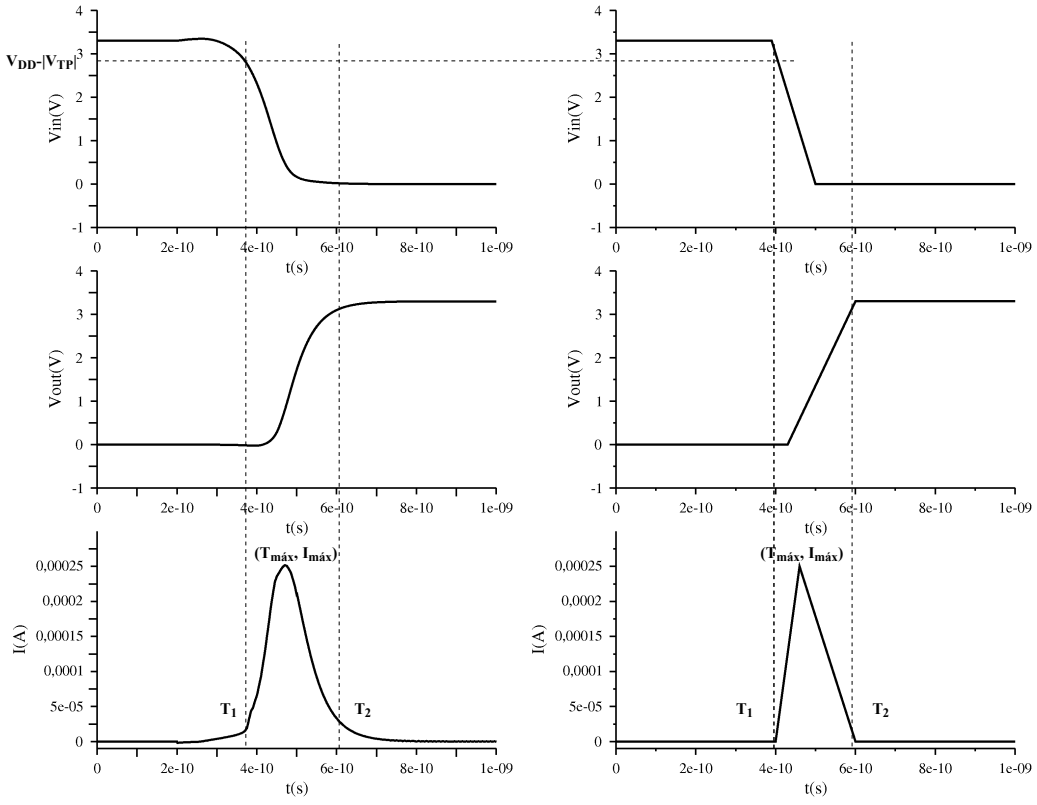


Figura 3.7. Curvas de intensidad durante la conmutación de un inversor a nivel alto: a) Simulación con HSPICE, b) Modelo triangular.

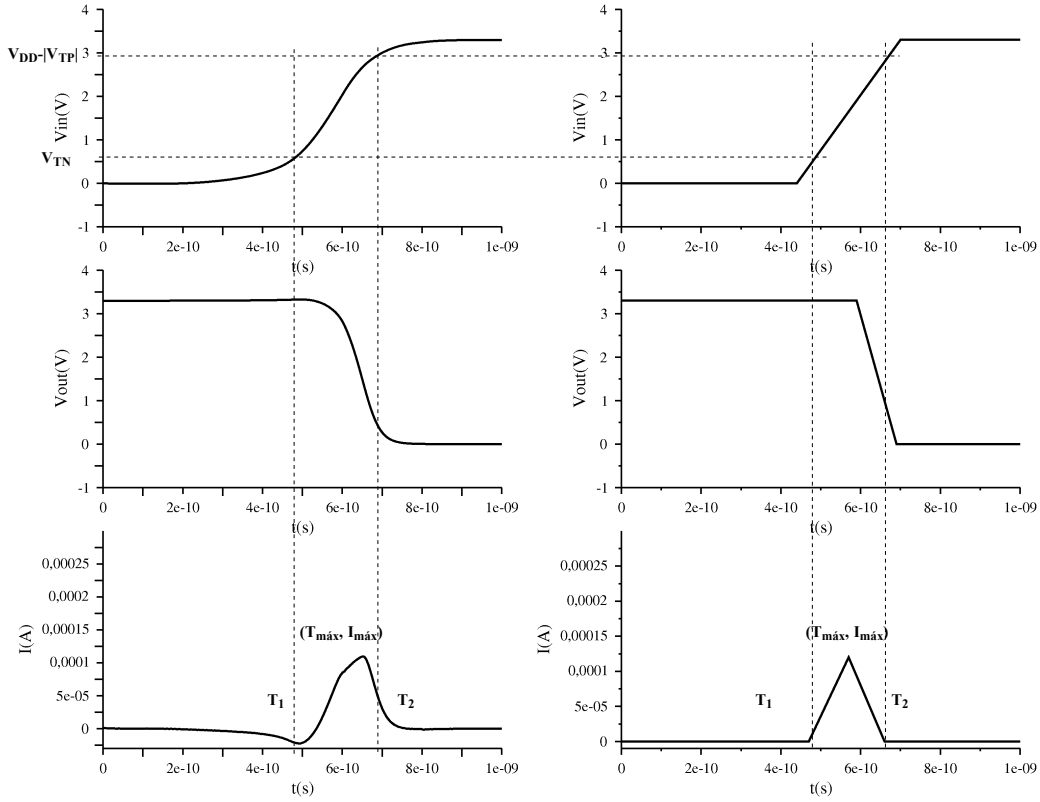


Figura 3.8. Curvas de intensidad durante la conmutación de un inversor a nivel bajo: a) Simulación con HSPICE, b) Modelo triangular.

Una vez establecida la forma de onda, hay que determinar los parámetros que la caracterizan. En las figuras 3.7 y 3.8 se han establecido cuatro parámetros que caracterizan completamente la curva triangular:

- T_1: Instante de comienzo de la circulación de la intensidad.
- $T_{máx}$: Intensidad máxima.
- T_2: Instante en que finaliza la circulación de intensidad.

Establecidos estos parámetros característicos, se determinarán las condiciones y ecuaciones para calcularlos en las transiciones de salida de las puertas. Del análisis realizado en la sección anterior se concluye que hay tratar por separado las transiciones de salida de subida y de bajada.

Por tanto, comenzando por las transiciones de salida de subida las condiciones y/o ecuaciones del cálculo de los parámetros característicos son las siguientes:

- T_1: Es el instante de cruce de la señal de entrada a la puerta por la tensión

umbral del transistor PMOS (V_{DD}-$|V_{TP}|$) (figura 3.7).

- T_2: En las transiciones de salida de subida, existen dos componentes de la intensidad: I_{SC} e I_{CL}. La importancia de cada componente depende del valor de C_L, estableciéndose dos casos diferentes:
  - **Caso 1**: Valor de C_L pequeño, por tanto, I_{SC} domina a I_{CL}. En este caso el instante T_2 es el instante en que $I_{SC}=0$, esto es, cuando la señal de entrada cruza por la tensión umbral del transistor NMOS (V_{TN}).
  - **Caso 2**: Valores grandes de C_L, por tanto, I_{CL} domina a I_{SC}. En este caso el instante T_2 es cuando la transición de salida ha alcanzado su máximo valor (figura 3.7).

  Para diferenciar entre el caso 1 y el caso 2 se emplea el modelo de retraso, el cual, determina la forma de onda de salida. Así, si el instante de finalización de la transición de salida es anterior al cruce de la señal de entrada por V_{TN}, se considera que se está en el caso 1, en cambio, si es posterior se considera que se está en el caso 2. En este caso 2, el instante T_2 se obtiene directamente de la forma de onda de la transición de salida (figura 3.7b).

- $T_{máx}$ e $I_{máx}$: Para el cálculo de la intensidad máxima y el instante donde se alcanza, se emplean resultados de modelos analíticos propuestos por otros autores. En concreto [MAUR01] propone las siguientes ecuaciones:

$$I_{max}=\sqrt{K_p\times W_p\times {V_{DD}}^2\frac{(C_L+C_{SC}+C_{PAR})}{\tau_{in}}} \tag{3.2}$$

$$T_{max}=\frac{(C_L+C_{SC}+C_{PAR})\,V_{DD}}{I_{max}} \tag{3.3}$$

  donde K_P es el factor de transcundancia, W_P es el ancho del transistor PMOS, V_{DD} es la tensión de alimentación, τ_{in} es el tiempo de duración de transición de entrada, C_L es la carga activa en el nodo de salida, C_{PAR} es la capacidad parásita en la salida y, C_{SC} es una capacidad equivalente a la carga movida por I_{SC}, definida como capacidad de cortocircuito en [TURGIS98]. Esta capacidad en su definición tiene dependencias tanto con la pendiente de entrada como con la de salida haciendo compleja la caracterización.

  Analizando los trabajos [MAUR01] y [TURGIS98] se concluye la existencia de dependencias lineales con la carga en la salida y C_L y la inversa de la pendiente de entrada τ_{in}. De esta forma, para poder llevar a cabo un proceso de caracterización de parámetros adecuado, se propone la siguiente ecuación para la intensidad máxima:

$$I_{max}^2 = P + Q \times C_L + \frac{R}{\tau_{in}} \tag{3.4}$$

donde el valor de C_L engloba tanto la carga activa en el nodo de salida como la capacidad parásita de la propia puerta.

Respecto a $T_{máx}$, según la ecuación 3.3 depende linealmente de C_L. Por este motivo se propone aplicar el mismo criterio que con T_2 distinguiendo valores de C_L pequeños (caso 1) y grandes (caso 2). En el caso 1, se obtiene $T_{máx}$ como el instante de cruce de la señal de entrada por la tensión umbral de la puerta lógica; en el caso 2, $T_{máx}$ se calcula de acuerdo a la siguiente ecuación:

$$T_{max} = \frac{C_L \times V_{DD}}{I_{max}} \tag{3.5}$$

En el caso de transiciones de salida de bajada la principal diferencia es la no existencia de I_{CL}, quedando el cálculo de los parámetros característicos de la curva de intensidad simplificados del siguiente modo:

- T_1: Se calcula del mismo modo que en las transiciones de subida, por tanto, es el instante en que la transición de entrada cruza el umbral de transistor NMOS (V_{TN}).
- T_2: Al no existir I_{CL}, el instante T_2 se calcula siempre igual que en el caso 1 de las transiciones de subida (figura 3.8b).
- $I_{máx}$: Al no considerarse intensidad de carga C_L, la ecuación propuesta para $I_{máx}$ es el resultado de establecer $C_L=0$ en la ecuación 3.4:

$$I_{max} = \sqrt{P + \frac{R}{\tau_{in}}} \tag{3.6}$$

- $T_{máx}$: Se obtiene del mismo modo que en el caso 1 de las transiciones de subida (figura figura 3.8b).

3.3. Validación y ajustes finales al modelo de intensidad

En esta sección se pretende validar el modelo triangular para el ajuste de la curva de intensidad propuesto en la sección anterior. Comenzando con la validación de T_1 y T_2 los criterios de elección de estos instantes están basados en el análisis heurístico realizado previamente a la propuesta del modelo.

En cuanto al instante inicial (T_1) en todas las curvas de intensidad reales se

comprueba como la intensidad comienza a circular a partir del instante de cruce de la señal de entrada por la tensión umbral del transistor PMOS ó NMOS (según la transición sea de bajada o subida respectivamente) (figuras 3.7 y 3.8). Para el instante final, los criterios adoptados en los diferentes casos son completamente lógicos y, quedando por comprobar la exactitud del instante final elegido en el caso 2 para transiciones de salida ascendentes. Este punto comprobará mediante la precisión de los resultados que se obtendrán con el simulador lógico en el ajuste de las curvas de intensidad respecto a los resultados de HSPICE. Estos resultados se presentan en el capítulo 5 de este trabajo.

De esta forma, esta sección está dedicada fundamentalmente a validar el valor de $I_{máx}$ calculado en el modelo propuesto. Para ello, se realiza un proceso de caracterización de los parámetros de la ecuación 3.4: P, Q y R. Con este proceso se pretende comprobar que el comportamiento real de la intensidad máxima se ajusta a dicha ecuación. Por esto, en el análisis de los resultados va a ser importante asegurar que se obtiene un coeficiente de correlación suficientemente alto para validar la ecuación.

El proceso de caracterización va a consistir en obtener valores reales de la $I_{máx}$ mediante simulación eléctrica con HSPICE. Como, según la ecuación 3.4, $I^2_{máx}$ depende linealmente con C_L y $1/\tau_{in}$ se analizan un conjunto suficientemente amplio de resultados de simulación variando tanto C_L como τ_{in}. A partir de la representación de $I^2_{máx}$ frente a C_L y τ_{in} se obtendrán valores de ajuste para los parámetros P, Q y R.

Por otro lado, un factor importante antes de abordar el proceso de caracterización es la similitud entre el entorno de caracterización y el entorno real de operación. Concretamente, en este proceso de caracterización, el hecho de tratar con estímulos diferentes a los reales, como son entradas lineales en rampa, produce resultados erróneos y, no detectables durante el proceso de caracterización, teniendo como consecuencia directa un gran error en los resultados calculados por el modelo durante la simulación lógica.

En [MILLAN03] y [MILLAN04] se comprobó como el hecho de trabajar con señales lineales tipo rampa en las entradas de las puertas lógicas introducía un error en los parámetros obtenidos en el proceso de caracterización hasta del 10%. En la obtención de medidas de $I^2_{máx}$ también hay que considerar la influencia del tipo de señal en la entrada del inversor ya que se repite la situación. En caso de utilizar señales no reales, como son las típicas rampas, la variación en la medida del pico de intensidad llega a superar el 20%. La figura 3.9 muestra esta situación, donde se han superpuesto, en cuatro simulaciones de un inversor CMOS realizadas con HSPICE, las formas de onda de intensidad obtenidas con una entrada real y con una entrada de tipo rampa. En todos los casos la intensidad

obtenida supera a la real, y presenta un comportamiento abrupto originado por el error del cálculo numérico en el simulador eléctrico.

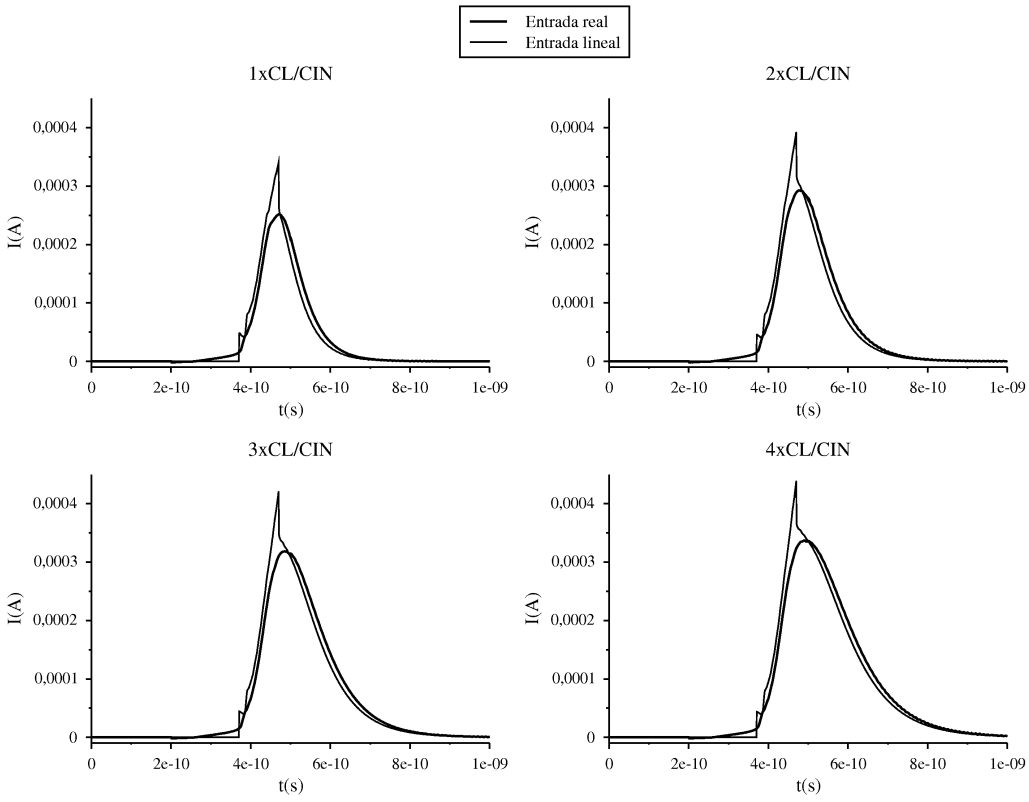


Figura 3.9. Error en la caracterización.

Para solventar este problema de caracterización se propone un circuito previo capaz de generar en su salida estímulos reales. Esta salida será la entrada del componente a caracterizar. El circuito propuesto es el de la figura 3.10 y tiene las siguientes características:

- El primero y segundo inversor son utilizados para suavizar las señales de entrada lineales de tipo rampa, obteniéndose señales reales en la salida del segundo inversor.
- El tercer inversor es el inversor bajo estudio.
- La carga C_T permite cambiar la pendiente de entrada de forma real. Generalmente se cambia este valor con múltiplos de la capacidad de entrada del inversor (C_{IN}), establecida como carga típica y, en unos rangos predeterminados y equivalentes a cargas en el nodo de 1 a 6 puertas lógicas.
- La carga C_L representa la carga en el nodo de salida. De manera similar a C_T, este valor de carga se fijará con diferentes valores múltiplos de la capacidad de entrada típica de la tecnología.

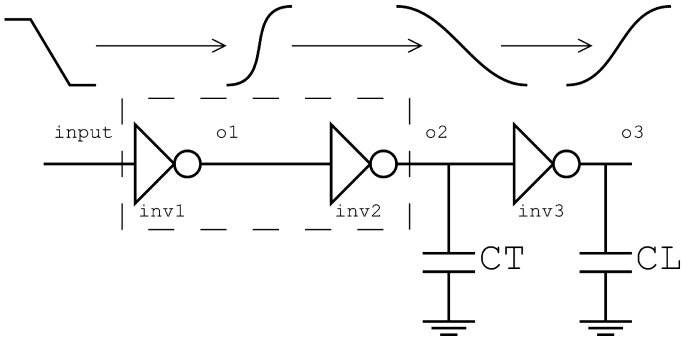


Figura 3.10. Circuito de caracterización

Otro aspecto importante es el procedimiento de medida del tiempo de transición, tanto para las transiciones de entrada (τ_{in}) como para las de salida (τ_{out}). Con formas de onda reales el procedimiento consiste en medir sólo el intervalo de tiempo que transcurre desde señal atraviesa el 30% de la tensión de alimentación hasta que atraviesa el 70% si la transición es de subida. En caso de transiciones de bajada, este intervalo será desde 70% al 30% del valor de tensión de alimentación. Basta con dividir este intervalo entre 0.4 y se obtendría el tiempo de transición completo.

Utilizando los procedimientos descritos, las simulaciones se han realizado en un rango de valores típicos de τ_{in} y C_L que cubren pendientes rápidas y lentas en las transiciones de entrada y cargas pequeñas y grandes en el nodo de salida. Los diferentes valores de C_L y C_T se han tomado utilizando como referencia el valor de la capacidad de entrada (C_{IN}) del inversor, considerada el valor de carga típica de la tecnología, valor obtenido a partir ecuación 3.1.

En el capítulo previo se mostró que la caracterización de celdas para los modelos de degradación y propagación normal se basa en la utilización de herramientas que facilitan y automatizan la mayor parte de este proceso (llamadas AUTODDM y AUTOTPN, respectivamente). Estas herramientas se apoyan en un software que facilita la interacción con el simulador eléctrico, llamada CIS [RUIZ01b]. CIS permite realizar simulaciones iterativas sobre un mismo circuito, alterando valores de varios parámetros en cada simulación individual, requiriendo un mínimo esfuerzo de programación. De manera similar a AUTODDM y AUTOTPN se ha implementado otra herramienta denominada AUTOTCM (*Auto Triangle Current Model*) que extrae los parámetros Q y R mediante reiteradas simulaciones sobre el circuito de la figura 3.10 utilizando CIS para interactuar con HSPICE.

Los resultados que a continuación se presentan consisten en la caracterización de dos celdas CMOS, un inversor y una puerta NAND ambas implementadas con tecnología de AMS CMOS 0,35μm y realizadas con AUTOTCM.

3.3.1. Caracterización del inversor CMOS

Con el proceso de caracterización descrito, las figuras 3.11 y 3.12 representan regresiones lineales de grupos de 11 puntos obtenidos con AUTOTCM y HSPICE.

En la primera serie de simulaciones se obtienen cuatro grupos de 11 valores de $I_{máx}$, simulando con HSPICE el circuito de la figura 3.10. Además, se representa en la figura 3.11 el cuadrado el pico de intensidad ($I^2_{máx}$) frente a la carga en el nodo de salida (C_L). Durante la obtención de los valores individuales de cada grupo, ha permanecido constante el valor de la pendiente de entrada (τ_{in}), esto es, la carga C_T permanece constante, alterándose en cada punto el valor de la carga en el nodo de salida C_L. En cambio, cada grupo de 11 puntos difiere la pendiente de entrada utilizada, alterándose C_T para este fin. Los diferentes valores de C_T son múltiplos de la carga de entrada típica (C_{IN}) logrando así pendientes equivalentes a diferentes *fan-in* en el nodo de entrada de la puerta a caracterizar. Así, los cuatro grupos se incluyen en la figura 3.11 junto con las rectas de regresión.

El objetivo de estas simulaciones es comprobar, en los rangos estudiados, las hipótesis de la ecuación 3.4, esto es, las dependencias lineales propuestas. Así, la primera hipótesis de partida es la existencia, mediante el parámetro de Q, de una dependencia lineal de $I^2_{máx}$ con C_L, y además, independiente de los valores de τ_{in}. El paralelismo entre las rectas en la figura 3.11, o de manera equivalente, la similitud en los valores de pendiente de las rectas de los ajustes lineales (tabla 3.1), justifican una de las dependencias lineales propuesta en las en las ecuación 3.4, concretamente con C_L.

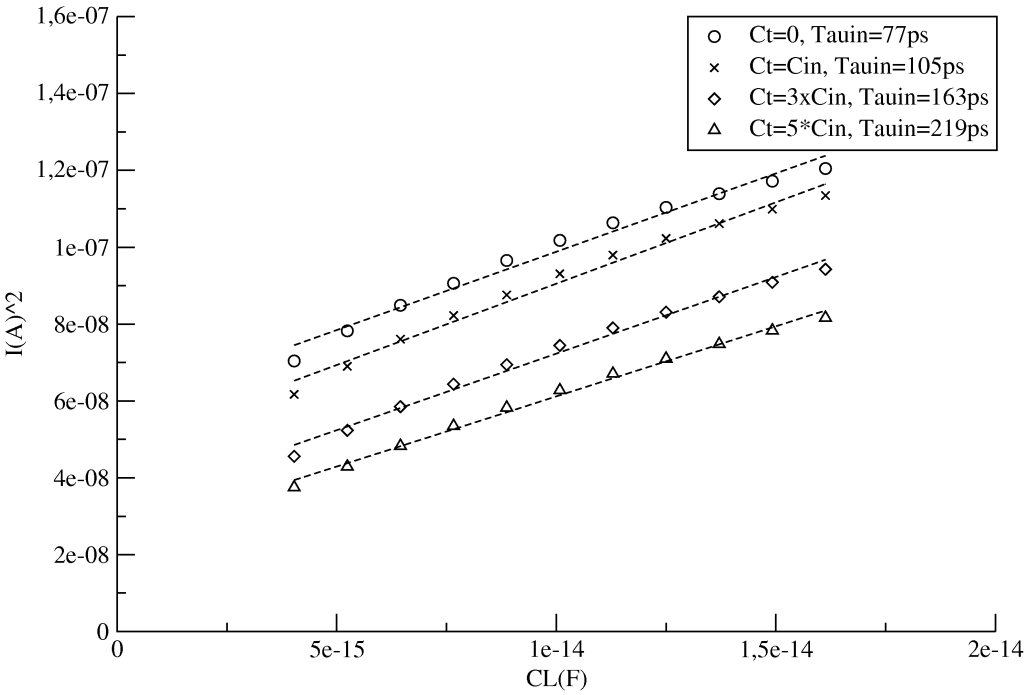


Figura 3.11. $I^2_{máx}$ frente a C_L.

τ_{in}	Pendiente	Coeficiente de correlación
77*ps*	$4.07 \cdot 10^6$ (A^2/F)	0.991636
105*ps*	$4.23 \cdot 10^6$ (A^2/F)	0.993805
163*ps*	$3.98 \cdot 10^6$ (A^2/F)	0.995442
219*ps*	$3.77 \cdot 10^6$ (A^2/F)	0.997317

Tabla 3.1. Pendientes obtenidos en las regresiones con valores de τ_{in} constantes.

Con una segunda serie de simulaciones se obtienen de nuevo otros cuatro grupos de 11 valores de $I^2_{máx}$. Así, la figura 3.12 representa $I^2_{máx}$ frente a $1/\tau_{in}$ incluyendo las regresiones lineales de cada grupo. En este caso, el criterio de obtención de cada grupo de puntos ha consistido en establecer como valor constante la carga en el nodo de salida (C_L), cambiando el valor de la pendiente de entrada 11 veces, alterando C_T en el circuito de caracterización. Como sucedía en el caso anterior, los resultados de pendiente justifican la dependencia lineal propuesta en las ecuación 3.4, en este caso, con $1/\tau_{in}$ (parámetro R de ambas ecuaciones). Ciertamente, los resultados de las regresiones tienen valores de pendiente muy similares, observados tanto en la figura 3.12 como en la tabla 3.2, tabla que además, incluye valores de los coeficientes de regresión de cada grupo donde todos son cercanos a uno.

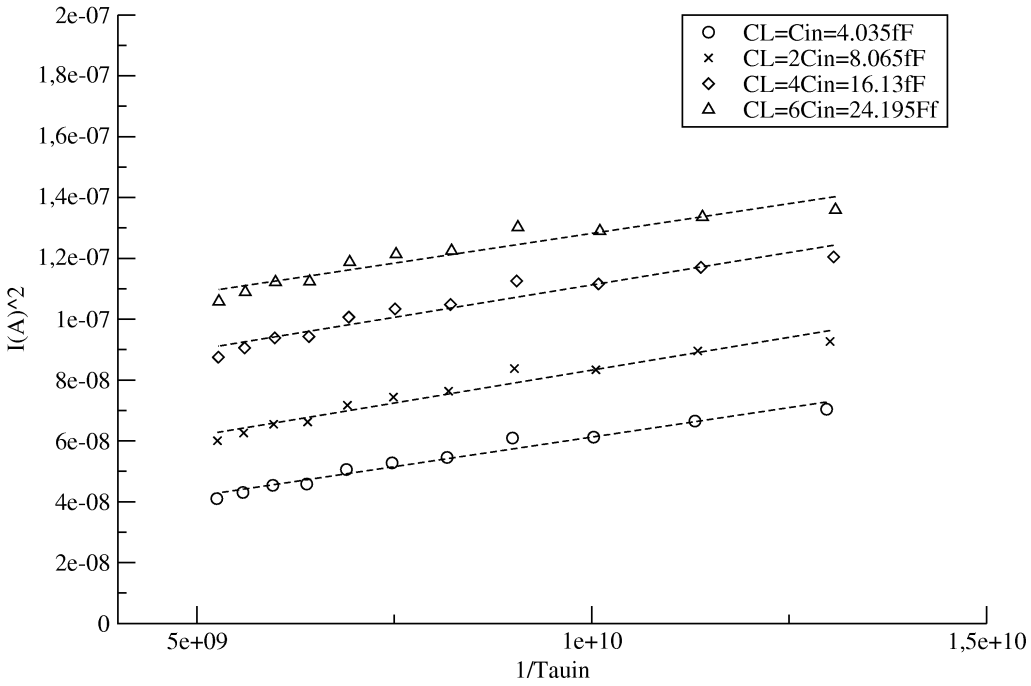


Figura 3.12. $I^2_{máx}$ frente a τ_{in} para diferentes valores de C_L.

C_L	Pendiente	Coeficiente de correlación
4.03fF	$3.88 \cdot 10^{-18}$ $(A^2 \cdot s)$	0.984107
8.06fF	$4.31 \cdot 10^{-18}$ $(A^2 \cdot s)$	0.975753
16.12fF	$4.25 \cdot 10^{-18}$ $(A^2 \cdot s)$	0.967203
24.19fF	$3.91 \cdot 10^{-18}$ $(A^2 \cdot s)$	0.955616

Tabla 3.2. Pendientes obtenidas en las regresiones con valores de C_L constantes.

Una medida válida para comparar las pendientes de las tablas 3.1 y 3.2 es la desviación estándar (o típica), la cual, es una medida del grado de dispersión de los valores respecto del valor medio. Aunque sólo se dispone de dos conjuntos de cuatro pendientes, esta medida es relevante e indicadora de la cercanía en los valores, siendo $0.19 \cdot 10^6$ en la tabla 3.1 y $0.22 \cdot 10^{-18}$ para la tabla 3.2. Los valores son muy bajos, concretamente, medidos en porcentaje respecto al valor medio se obtiene 4.78% y 5.48% para la tabla 3.1 y 3.2 respectivamente.

Con lo anteriormente expuesto no se obtienen las medidas de los parámetros establecidos por el modelo (P, Q y R), pero si se dispone de gran cantidad de puntos, concretamente, todos los obtenidos tanto para las figuras 3.11 y 3.12, esto es, un total de 88 puntos de tres coordenadas ($I^2_{máx}$, C_L, $1/\tau_{in}$). Estos son puntos suficientes para realizar una regresión múltiple de la ecuación 3.4. En la tabla 3.3 se muestran los resultados de dicha regresión con el valor de ajuste de los parámetros P, Q y R, el error de cada parámetro y los coeficientes de correlación.

Parámetro	Valor	Error	Correlación Parcial
Q	$3.362943 \cdot 10^{6}$ (A^2/F)	1.1%	0.974826
R	$4.37797 \cdot 10^{-18}$ $(A^2 \cdot s)$	1.7%	0.931482
P	$9.822429 \cdot 10^{-9}$ (A^2)	2.2%	0.493653

Tabla 3.3. Datos de ajuste con el término independiente.

Del análisis de los datos de la tabla 3.3, lo más significativo es el parámetro P (término independiente de la ecuación 3.4) donde observan dos hechos: (1) el valor de P es relativamente pequeño cuando se compara con los valores de $I^2_{máx}$ de las curvas reales; (2) el coeficiente de correlación de P es bajo (0.49) y mucho peor que los de Q y R. Es decir, con las figuras 3.11 y 3.12 se comprueba la validez de las dependencias lineales de $I^2_{máx}$ con C_L y $1/\tau_{in}$ (parámetros Q y R) y, en cambio, el término independiente P de la ecuación 3.4 da un resultado pequeño y de difícil ajuste.

Por estos motivos se propone una modificación a la ecuación 3.4 que consiste en eliminar el término independiente P. De esta forma, la nueva ecuación propuesta para el cálculo del valor $I^2_{máx}$ es la siguiente:

$$I^2_{max} = Q \times C_L + R \times \frac{1}{\tau_{in}} \tag{3.7}$$

Esta ecuación tiene la ventaja añadida de necesitar un parámetro menos a la hora de la caracterización. En la tabla 3.4 aparecen los resultados de ajustar los 88 puntos de $I^2_{máx}$, C_L, τ_{in} obtenidos por simulación eléctrica a la ecuación 3.7.

Comparándolos con los resultados de la tabla 3.3 se obtienen dos resultados de interés: (1) los errores de los parámetros son menores en el caso del ajuste de la ecuación 3.7 respecto a los de la ecuación 3.4; (2) los coeficientes de correlación también son mejores.

Parámetro	Valor	Error	Correlación Parcial
Q	$3.58834 \cdot 10^{6}$ (A^2/F)	1.1%	0.978436
R	$5.1641 \cdot 10^{-18}$ $(A^2 \cdot s)$	1.2%	0.97598

Tabla 3.4. Datos de ajuste sin el término independiente.

3.3.2. Caracterización de una puerta NAND de 2 entradas

En la práctica, la extensión de un modelo a puertas de más de una entrada consiste en tratar cada entrada de manera individual. Dado el caso, el modelo de intensidad propuesto trata cada entrada por separado obteniendo su propio conjunto de parámetros en cada combinación entrada/salida, criterio también utilizado en modelo IDDM presentado en el capítulo anterior.

Por otro lado, los ajustes multilineales se han realizado para las dos ecuaciones propuestas (3.4 y 3.7) y así, de igual modo que con el inversor, poder concluir que la eliminación del término independiente no introduce error significativo en el modelo de corriente.

Ha sido necesario obtener una nube de puntos en los rangos de interés para cada una de estas entradas. Utilizando de nuevo el circuito de la figura 3.10, se establecen como rango de interés en las cargas C_T y C_L aquellos que abarcan desde de $1 \times C_{IN}$ hasta $6 \times C_{IN}$ para ambas cargas (donde C_{IN} es el valor típico de entrada). Con un total de 88 puntos por entrada, obtenidos por simulación eléctrica con la herramienta AUTOTCM, se han repetido los ajustes lineales múltiples para las dos ecuaciones (3.4 y 3.5). Éstos resultados de ajuste se muestran en la tabla 3.5 y tabla 3.6 respectivamente.

Entrada	Parámetro	Valor	Error	Correlación Parcial
A	**P**	$3.252834 \cdot 10^{-9}$ (A^2)	1.5%	0.368557
A	**Q**	$3.277144 \cdot 10^{6}$ (A^2/F)	1%	0.98744
A	**R**	$4.379074 \cdot 10^{-18}$ $(A^2 \cdot s)$	1.1%	0.980865
B	**P**	$3.98298 \cdot 10^{-8}$ (A^2)	1.2%	0.973338
B	**Q**	$2.204682 \cdot 10^{6}$ (A^2/F)	0.8%	0.964811
B	**R**	$4.517099 \cdot 10^{-18}$ $(A^2 \cdot s)$	0.9%	0.977219

Tabla 3.5. Datos de ajuste con el término independiente.

Entrada	Parámetro	Valor	Error	Correlación Parcial
A	**Q**	$3.411087 \cdot 10^{6}$ (A^2/F)	0.8%	0.992098
A	**R**	$4.62879 \cdot 10^{-18}$ $(A^2 \cdot s)$	0.8%	0.99045
B	**Q**	$3.147238 \cdot 10^{6}$ (A^2/F)	2.2%	0.904219
B	**R**	$9.38126 \cdot 10^{-18}$ $(A^2 \cdot s)$	2.3%	0.964559

Tabla 3.6. Datos de ajuste sin el término independiente.

De nuevo, en la tabla 3.5 se observa un problema de ajuste de correlación en el parámetro P de la entrada A. Este problema no existe en los datos de la tabla 3.6.

3.4. Implementación

Durante la ejecución de la simulación lógica el motor de simulación genera valores de diversos parámetros (tiempos de transición, carga en los nodos, etc.) obtenidos tras aplicar los modelos implementados en la herramienta. Los tiempos relevantes para la implementación del modelo de corriente son los instantes de tiempo de las transiciones (calculadas a partir del modelo IDDM) y los tiempos de duración de las transiciones (calculados con el modelo de pendiente de salida

propuesto en el capítulo anterior).

Con todos estos datos es posible el cálculo durante la simulación de los puntos T_1, T_2, $T_{máx}$ e $I^2_{máx}$, puesto que se dispone, en tiempo de simulación, de los valores de carga en los nodos (C_L) y tiempos de transición tanto de entrada como de salida (τ_{in},τ_{out}) en cada puerta.

En la implementación, estos cálculos presentan peculiaridades durante la simulación originadas, principalmente, por el procedimiento de medida de pendiente de las transiciones, tanto en el proceso caracterización, como en el propio proceso de simulación. La simulación lógica trata las señales de entrada como rampas de subida o bajada con un tiempo de transición asociado. Estos tiempos de transición fueron obtenidos durante el proceso de caracterización mediante una aproximación, ya mencionada, que consistía en medir el 40% del tiempo total. La aproximación hecha presenta un mayor error respecto a la original tanto en la zona inicial de la señal como en la zona final, es decir, en las colas. Una representación de una señal real obtenida con HSPICE y de la aproximación obtenida se muestran superpuestas en la figura 3.13. Además, la figura incluye la representación de una corrección propuesta para utilizar en ciertos cálculos, consistente, en dividir el tiempo de transición aproximado entre 0.7 haciendo así la curva linealizada más parecida a la real en las colas. Este factor fue incluido y justificado en [JUAN00a] para el modelo de degradación y ahora se justifica su uso en el cálculo del punto T_1. Este primer vértice del triángulo (T_1) presenta la peculiaridad de ser instante que la señal de entrada alcanza el valor de tensión umbral del transistor (P ó N según el caso) y, estos valores están en la zonas de colas de la señal de entrada, por tanto, una mejor aproximación se consigue utilizando la pendiente corregida.

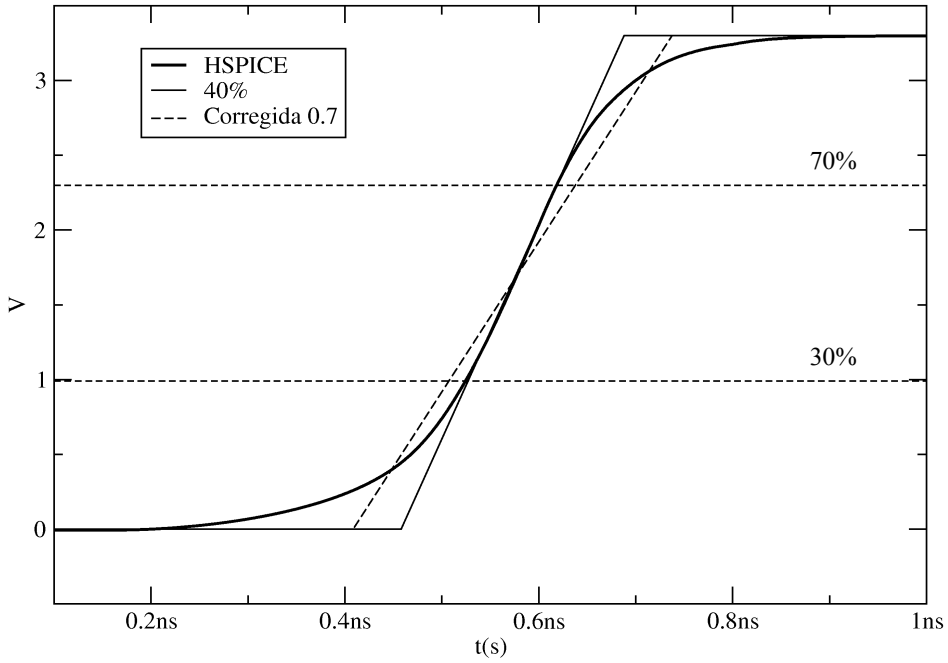


Figura 3.13. Corrección de la pendiente durante el proceso de simulación.

3.4.1. Cálculo de potencia a partir de la curva de intensidad

Cuando se dispone de la curva de corriente tras una simulación, la obtención de la potencia es directa. El objetivo es realizar el cálculo de potencia durante la ejecución del proceso de simulación lógica. El cálculo se basa en la integración de la curva de intensidad, por tanto al disponer de una curva de intensidad se puede calcular la potencia (P) como:

$$P = \frac{Area \times V_{dd}}{\Delta T} = \frac{E_G}{\Delta T} \tag{3.8}$$

$$Area = \int I \cdot dt = \Delta Q \tag{3.9}$$

donde E_G es la energía total de la simulación y ΔT representa el intervalo temporal de simulación. Durante la simulación E_G se irá calculando mediante las aportaciones individuales de cada transición de salida que se procese. Tras haber calculado con el modelo de retraso del simulador la transición de salida, se aplica el modelo de intensidad. Con los resultados de los tres vértices del triángulo de intensidad, se calcula la energía consumida en una transición (E_T). Multiplicando el área por

la tensión de alimentación mediante:

$$E_T = \frac{T_{max} - T_1}{2} \times I_{max} \times V_{DD} + \frac{T_2 - T_{max}}{2} \times I_{max} \times V_{DD} \tag{3.10}$$

al final de la simulación, el simulador lógico temporal habrá calculado:

$$E_G = \sum E_T \tag{3.11}$$

y finalmente, dividiendo por el intervalo temporal se obtendrá la potencia (ecuación 3.8).

Estas ecuaciones se implementarán en el simulador lógico HALOTIS disponiendo el usuario de datos de potencia consumida para cada simulación ejecutada.

Capítulo 4. Simulador lógico HALOTIS

Este capítulo hace un recorrido en el desarrollo del entorno de simulación lógico temporal HALOTIS cuyo nombre hace referencia a *High Accuracy LOgic Timing Simulator*. Partiendo de una visión general del sistema, el capítulo recorre todos los pasos del desarrollo llegando hasta una descripción de la estructura interna de cada una de las partes de HALOTIS.

El capítulo está organizado en siete secciones. La primera presenta una visión general del desarrollo de HALOTIS tanto desde la perspectiva técnica como histórica. En la segunda se describe la arquitectura general de HALOTIS y sus componentes principales. Tras esto, se describe brevemente la metodología de diseño utilizada para facilitar la comprensión de los diagramas que describen la estructura interna de HALOTIS. En los apartados cuatro y cinco es donde se realiza el análisis de requisitos y descripción de HALOTIS, así como el modelado de cada uno de sus componentes. La penúltima sección trata en profundidad algunos aspectos de la implementación de los diferentes componentes de HALOTIS y, en la última sección se comenta brevemente QHALOTIS, herramien-

ta que se presenta como primera versión del interfaz gráfico de HALOTIS.

4.1. Visión general del desarrollo de HALOTIS

HALOTIS es un simulador lógico temporal que implementa las características especiales de DDM y el nuevo algoritmo de tratamiento del efecto inercial. Las principales características de este software es la modularidad y el diseño orientado a objetos, haciendo posible la implementación de una variedad de modelos de retraso además del DDM, como es un modelo de cálculo de intensidad. Actualmente admite ficheros en formato VERILOG, por compatibilidad con otras herramientas comerciales, pero puede ser extendido fácilmente con módulos para otros lenguajes como podría ser VHDL.

Desde una perspectiva técnica, para desarrollar correctamente una herramienta software de cierta complejidad se hace imprescindible realizar una serie de tareas básicas como son: análisis, diseño e implementación, llamadas tradicionalmente ciclo de vida del software. Existe un gran número de metodologías de desarrollo de software que cubren el ciclo de vida, incluyendo o subdividiendo estas tres etapas básicas en otras, que tratan conceptualmente los diferentes aspectos del software, sobre todo en la etapa de diseño. Esta subdivisión de las fases básicas del ciclo de vida da lugar a las diferentes metodologías, algunas orientadas a resolver determinados problemas tipo y otras más generales, capaces de abarcar una gran cantidad de casuísticas. Íntimamente unido a la metodología aparece el concepto de modelado. Un modelo, no es más que una representación en cierto medio de algo que está en otro medio o incluso en el mismo, es decir, se podría considerar como una abstracción conceptual de una determinada realidad [RUMB00]. Por tanto el modelo permitirá captar y enumerar exhaustivamente los requisitos y el dominio de conocimiento de forma que todos los implicados puedan entenderlos y estar de acuerdo con ellos. Con este fin, todo modelo tiene asociada una determinada notación, que deben compartir las personas implicadas y que da forma a una determinada metodología.

Hoy en día las metodologías de modelado orientadas a objetos son las de mayor aceptación en la ingeniería del software. Efectivamente, está bastante aceptado en cualquier caso que el diseño de software relativamente complejo con una metodología orientada a objetos obtiene un resultado, por un lado, fácil de implementar en un lenguaje de programación al ser altamente ordenada, y por otro lado fácil de mantener, de depurar, de ampliar y con un alto grado de robustez. Éstas metodologías de modelado orientadas a objetos comienzan a aparecer entre mediados de los 70 y principios de los 80. El número de lenguajes

de modelado aumentó considerablemente desde menos de diez hasta más de cincuenta durante el período 1989-1994 y muchos usuarios de estas metodologías no se encontraban satisfechos con ninguna de estas técnicas. Esto originó la denominada "guerra de métodos", haciendo evolucionar y destacar algunas de ellas siendo las más notables Booch'93, OMT[1] y Fusion. Hoy en día UML (*Unified Modeling Languaje*) es la evolución de este conglomerado de metodologías, como son OMT, OOSE[2], Booch y UML97. Ha sido estandarizada y es reconocida mundialmente por lo que es la que se ha utilizado para el desarrollo de HALOTIS facilitando la descripción y documentación correcta del software.

Durante el desarrollo de HALOTIS se han recorrido las diferentes etapas del ciclo de vida, desde un análisis del problema, pasando por el diseño y finalizando en una implementación, utilizando la metodología UML [RUMB00, UMLN97]. En la actualidad ya existen ejemplos de compañías dedicadas al CAD para diseño de circuitos integrados que la emplean, como la compañía XILINX [XILINXa].

Desde una perspectiva histórica, HALOTIS se comenzó a desarrollar en 1999 generándose una primera versión operativa del simulador. En esta versión, se desarrolló principalmente el motor de simulación para el modelo IDDM. Sin embargo, inicialmente carecía de módulos para la interpretación de circuitos descritos en lenguajes VDHL o VERILOG, tampoco disponía de ningún formato específico para patrones o librerías de celdas. Dos años después del comienzo del desarrollo, esta primera versión incluía un módulo para lectura de ficheros HDL-VERILOG, patrones en formato MACHTA [MACH00] y librerías de celdas en formato propio de texto. Esta primera versión consistió en un software monolítico, es decir, un único programa con todo incluido, lo que comenzó a dificultar su ampliación y la inclusión de nuevos modelos como es el de intensidad.

Durante el año 2002 y 2003 se reescribió HALOTIS surgiendo la segunda versión, que reutilizó principalmente el código fuente del núcleo de simulación (aproximadamente el 40% del código original). Con esta segunda versión se ha conseguido realizar un software altamente modular, basado en la reutilización de código en todos sus componentes mediante una librería denominada *halotislib*. Esta librería abre las puertas a otros desarrollos adicionales, como por ejemplo, la interfaz gráfica de HALOTIS llamada QHALOTIS, mostrada en el último apartado de este capítulo y el editor de librerías de celdas actualmente en desarrollo. Además ha permitido el desarrollo de forma paralela de herramientas de apoyo como es un chequeador de circuitos y un chequeador de librerías de celdas, ambos descritos en este capítulo.

1. Object Modeling Technique.
2. Object-Oriented Software Engineering.

4.2. Arquitectura general de HALOTIS

Esta sección describe el funcionamiento del entorno de simulación HALOTIS a nivel de usuario. El entorno de simulación está formado por un grupo de programas que se utilizan en los diferentes pasos del proceso de simulación. A continuación se hace una introducción a los programas que forman este entorno de simulación, el orden en el uso y los resultados parciales de cada uno de los mismos.

Todos los programas del entorno HALOTIS funcionan en línea de comandos. Cada uno se encarga de tratar diferentes ficheros necesarios para la simulación (*netlist*, patrones, librerías de celdas) y compilarlos[3] para detectar errores y transformarlos a un formato binario y así, el motor de simulación puede trabajar con ellos. Estos programas son:

- **hcells:** Programa compilador de librerías de celdas, toma un fichero en formato XML o HALOTIS-nativo donde se describen las celdas y sus parámetros. Comprueba la sintaxis y lo compila.
- **hverilog***:* Programa encargado del compilar un *netlist* en formato VERILOG, comprueba la sintaxis y enlaza los componentes existentes con las celdas de la librería de celdas.
- **hmachta:** Conversor de ficheros de patrones en formato de la herramienta MACHTA [MACH00] al formato nativo de patrones de HALOTIS.
- **halotis***:* Simulador lógico, utiliza los ficheros generados por *hcells* y *hverilog* junto con un conjunto de patrones en modo texto y obtiene los resultados de simulación.
- Programas de utilidades y comprobación de apoyo al usuario:
  - **hnetlistdump***:* extrae un fichero binario compilado con *hverilog* a un fichero de texto en formato VERILOG.
  - **hcellsdump***:* extrae un fichero binario compilado con *hcells* a un fichero de texto en formato XML.
  - **hcellscheck***:* chequea los parámetros un fichero de celdas binario comprobando si están en ciertos rangos admitidos como válidos para cada modelo de comportamiento.

La figura 4.1 contiene un diagrama de flujo con el orden de ejecución de los

3 Entendemos por "compilar" el preprocesado realizado a un fichero para comprobar los posibles errores sintácticos, léxicos o semánticos cometidos por un usuario al escribirlo. El resultado de una compilación es un fichero en formato binario libre de errores.

diferentes programas de HALOTIS para realizar una simulación. La figura incluye los ficheros generados con cada programa marcados con fondo oscuro y una línea tramada para indicar el programa que lo utilizará posteriormente. En la secuencia de la figura se observa que, previo a la realización de la simulación es necesario ejecutar dos tareas. La primera de ellas está asociada a un determinado *kit* tecnológico y consiste en compilar los ficheros que describen el comportamiento de las celdas y generar un fichero binario de la librería de celdas en formato HALOTIS. Se considera que esta es una tarea previa al desarrollo del diseño y, más propia de los fabricantes que de los diseñadores de circuitos. La segunda tarea está asociada al propio proceso de diseño y consiste en compilar los *netlists* de los circuitos para generar un fichero binario en formato HALOTIS. En la versión actual la introducción del diseño se realiza en formato VERILOG, compatible con la mayoría de simuladores y, sobre todo, herramientas de síntesis comerciales.

Tras estas dos tareas previas se ejecuta la simulación utilizando como entrada los ficheros generados en las tareas previas (ficheros binarios de librería de celdas y *netlist*) así como un fichero de texto donde quedan especificados los patrones de simulación. Como hemos mencionado anteriormente, si bien existe un formato nativo de HALOTIS para la descripción de patrones, también se ha desarrollado un módulo que permite emplear ficheros de patrones con un formato estándar como es el de la herramienta MACHTA.

Para simplificar todo este proceso de ejecución sucesivo de programas se ha implementado una primera versión de la interfaz gráfica de HALOTIS llamada QHALOTIS. QHALOTIS trabaja internamente con estos ficheros y programas facilitando al usuario la realización de simulaciones y representación de los resultados. Esta aplicación se discute en profundidad en la última sección del capítulo.

Todo este conjunto de herramientas se modelan y estudian a lo largo del capítulo utilizando la metodología de diseño UML. Antes de mostrar el diseño completo de HALOTIS se hace una breve introducción a esta metodología en la siguiente sección.

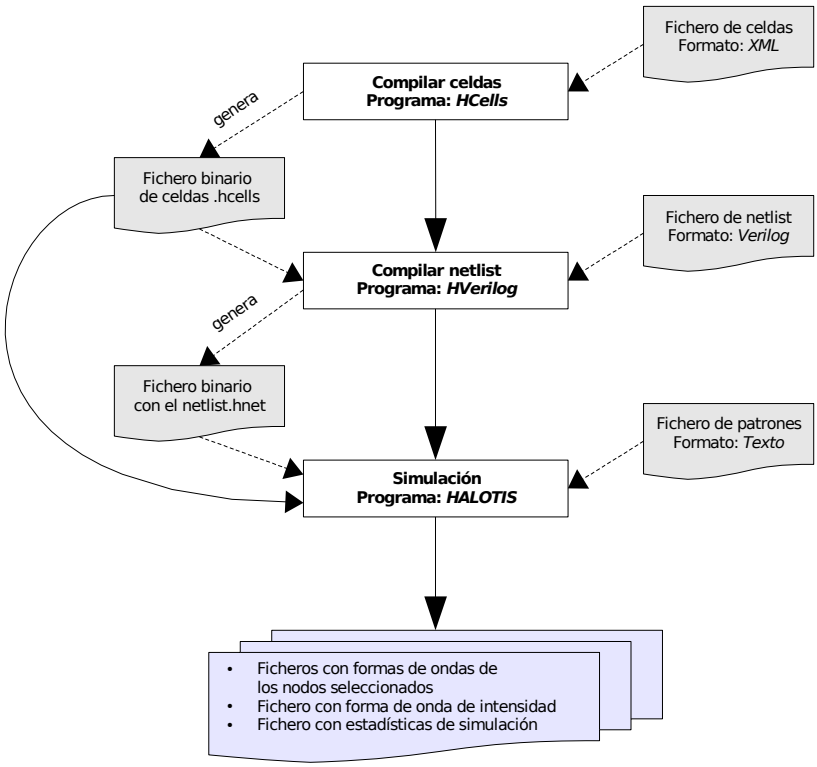


Figura 4.1. Diagrama de ejecución de una simulación con HALOTIS.

4.3. Metodología de diseño orientada a objetos OMG UML

UML (*Unified Modeling Language*) es un lenguaje para especificar, construir, visualizar y documentar los sistemas software. Con UML se tiene un lenguaje visual para el modelado, pero no un lenguaje visual de programación, es decir, a partir de él no se deriva el código en algún tipo de lenguaje de programación. Algunos elementos del software (bucles, saltos) quedan mucho mejor expresados con el propio código fuente, pero la arquitectura y estructura del sistema se puede expresar y comprender perfectamente a partir del lenguaje UML. Con esta metodología se cubrirán las etapas de desarrollo software de análisis, diseño, programación y testado.

UML es una metodología orientada a objetos. Con el concepto de objeto se pretende agrupar en una entidad, en principio abstracta, ciertos aspectos del do-

minio del problema, los cuales comparten algunas características comunes, tal y como ocurre en nuestra realidad más cotidiana. Un objeto, por ejemplo una mesa, es una agrupación de ciertas características: cuatro patas y una tabla colocada en la parte superior. En un modelado orientado a objetos, el diseñador debe plantearse cómo agrupar las características de aquello que se pretende modelar, obteniendo objetos que la representen y, consiguiendo así, que el conjunto de todos los objetos represente en su totalidad el dominio del problema.

Haciendo una breve historia, se puede decir que el desarrollo de UML comenzó en octubre de 1994 cuando Grady Booch y Jim Rumbaugh de Rational Software comenzaron a trabajar en la unificación de los lenguajes de modelado Booch y OMT, es entonces cuando fueron reconocidos mundialmente a la cabeza del desarrollo de metodologías orientadas a objetos. Fue entonces cuando terminaron su trabajo de unificación obteniendo el borrador de la versión 0.8 del denominado *Unified Method* en octubre de 1995. Tras esto también en 1995, Ivar Jacobson padre del la metodología OOSE se unió con Rational Software para obtener finalmente UML 0.9 y 0.91 en junio y octubre de 1996. Muchas organizaciones como Microsoft, Hewlett-Packard, Oracle, Sterling Software MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjectTime, Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam,. se asociaron

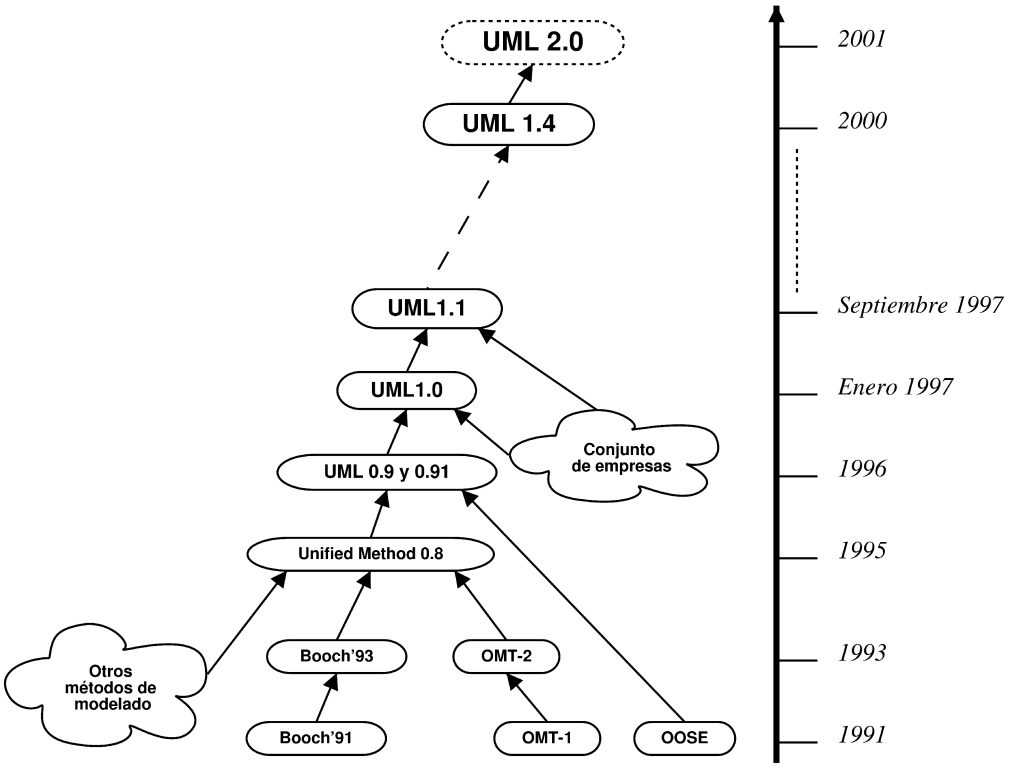


Figura 4.2. Evolución histórica de UML

junto con Rational Software para dar como resultado UML 1.0 y UML 1.1. llegando hoy en día hasta UML 1.4 y UML 2.0. En la figura 4.2 se muestra de forma gráfica y resumida la evolución de UML.

Son muchas las organizaciones que han incorporado UML como un estándar en el desarrollo de sus procesos y productos, teniendo como gran ventaja que al ser no propietario, está abierto a su uso en la comunidad científica [UMLS97].

4.3.1. Características principales de la metodología UML

Durante el desarrollo del capítulo se utilizarán una serie de elementos gráficos que componen la metodología UML. Estos elementos gráficos de la metodología, pretenden aportar un lenguaje de descripción común, y fácil de entender, entre todas las personas involucradas en un proyecto. Este es uno de los objetivos del uso de UML: conseguir intercambiar información sin invertir gran esfuerzo en estas tareas, puesto que se comparte una determinada notación. UML dispone de diferentes construcciones para representar gráficamente el sistema. Estas construcciones se denominan vistas, las cuales, intentan resaltar determinadas características que componen el software. Durante las etapas del ciclo de vida se utilizarán aquellas que se consideren necesarias según la naturaleza del software que se pretenda modelar, es por ello que en el presente capítulo se utilizarán las siguientes:

- **Diagrama de casos:** Para realizar la descripción del sistema desde un punto de vista de alto nivel, UML proporciona los denominados diagramas de casos. Éstos describen la funcionalidad de un sistema siguiendo una metodología descendente, es decir, partiendo de una visión de alto nivel del sistema y continuando con una descomposición del mismo en partes más concretas [KOBR01]. El propósito de un caso de uso es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema. La notación consiste en círculos que representan tareas que se realizan unidos entre sí mediante líneas con dos tipos de cualificación:
  - *uses:* Representa una relación de inclusión, es decir, una tarea que incluye a otras determinadas tareas.
  - *extend:* Representa una relación de extensión. Se pueden tratar estás, como diferentes formas de realización o especializaciones de dicha tarea.
- **Diagrama de objetos:** Este diagrama representa los aspectos estáticos del sistema a desarrollar. Estos aspectos estáticos modelan características del software como pueden ser la estructura interna, tanto del mismo código fuente como lo que es más importante, la representación que se hará de la

información en el sistema informático. Éste suele ser el núcleo de todos los desarrollos software, ya que un buen modelado produce un software de buena calidad, por lo que se debe emplear gran esfuerzo en esta tarea. Este diagrama estará formado por objetos, representados con cajas cuadradas, y enlazados entre sí. Estas relaciones junto con los objetos tienen como objetivo conseguir, en el mayor grado posible, una abstracción de la realidad que pretendemos representar, que abarque en su totalidad los elementos y características del sistema que representa.

En el anexo 2 se presentan ejemplos ilustrativos tanto de los diagramas de casos como de los diagramas de objetos con una explicación detallada sobre como interpretarlos.

Al abarcar UML gran cantidad de tipos especificaciones de software, existen otro componentes visuales como diagramas de estado, diagrama de actividad, etc. que no se utilizarán al no estar en el campo de aplicación del entorno HALOTIS y, por tanto no se describirán.

Una vez descritos los componentes básicos de la metodología, en las siguientes secciones del capítulo serán utilizados para describir todos las fases del ciclo de vida del desarrollo de HALOTIS. Tradicionalmente, este ciclo de vida se compone de una serie de fases, a las cuales se han asociado determinados elementos de UML [RUMB96]:

1. **Análisis global o conceptualización:** Es una primera fase en la que el principal objetivo es marcar las pautas del problema a resolver. Se suelen realizar conjuntos de requisitos, tareas, resultados, etc. que el sistema debe realizar.

2. **Análisis de requisitos:** Esta segunda fase consiste en organizar los requisitos de la primera fase de forma ordenada y refinada, con la finalidad de poder realizar una análisis de los mismos para detectar posibles inconsistencias, omisiones, redundancias, etc. Este software forma parte de tareas de investigación, por tanto, aunque el análisis de requisitos sea exhaustivo, a lo largo del desarrollo puede ser alterado. La mejor solución en este tipo de proyectos es establecer unos requisitos generales y no muy concretos.

3. **Diseño del sistema:** Se obtendrá una visión global del sistema que queremos desarrollar desde un punto de vista de alto nivel. En esta fase se ha han usado los diagramas de casos.

4. **Diseño de objetos:** Esta fase obtiene como resultado un modelo de objetos que será implementado en la siguiente fase. Este modelo de objetos debe representar la realidad que se desea abstraer en el sistema informático

fielmente. Se utilizan los elementos de UML denominados diagrama de clases, diagramas de estado y diagrama de actividades.

5. **Implementación:** Antes de implementar el sistema hay que decidir el lenguaje a utilizar. Se mostrarán las características o facilidades de las que disponemos en determinados lenguajes para optar finalmente por un determinado lenguaje

En el resto de secciones del capítulo se cubren cada uno de estas fases sucesivamente con alto grado de detalle. Se mostrará toda la estructura de HALOTIS en el grado actual de desarrollo en el que se encuentra.

Es importante comprender que la metodología UML cubre todas las etapas de desarrollo, incluidas las de especificación, pero en este caso, HALOTIS está ya desarrollado, y por tanto, la fase de especificación (o análisis de requisitos) se concreta mostrando las capacidades actuales de HALOTIS. Además, en las siguientes secciones (modelado e implementación) se describe la herramienta, enfocada esta descripción en como se ha realizado el modelado e implementación de la herramienta, no como debe modelarse al estar el desarrollo finalizado. Por tanto, se concluye que, el uso de la metodología UML en este documento es para documentar el desarrollo HALOTIS, en cambio, durante el desarrollo de HALOTIS, UML sí se utilizó para facilitar todo el proceso.

4.4. Análisis de requisitos y descripción de HALOTIS

Esta sección concreta todos los aspectos de la funcionalidad de HALOTIS, los lenguajes que interpretará, los datos a mostrar en las simulaciones, el formato de los mismos, el modo de interacción con el usuario y algunos ejemplos para facilitar la comprensión.

4.4.1. Descripción del circuito

Para la descripción de un circuito digital (*netlist*) se propone utilizar el lenguaje VERILOG [WENG03]. VERILOG es un lenguaje de descripción de hardware que soporta el diseño de circuitos digitales analógicos y mixtos a diferentes niveles de abstracción. HALOTIS sólo interpretará un subconjunto de VERILOG limitado a descripciones estructurales de circuitos. La estructura modular con la que se diseña HALOTIS hace posible la inclusión de otros módulos capaces de interpretar otros lenguajes de descripción como VHDL. Un ejemplo de circuito junto con su descripción VERILOG se muestra en la figura 4.3 y en el ejemplo 4.1 respectiva-

mente.

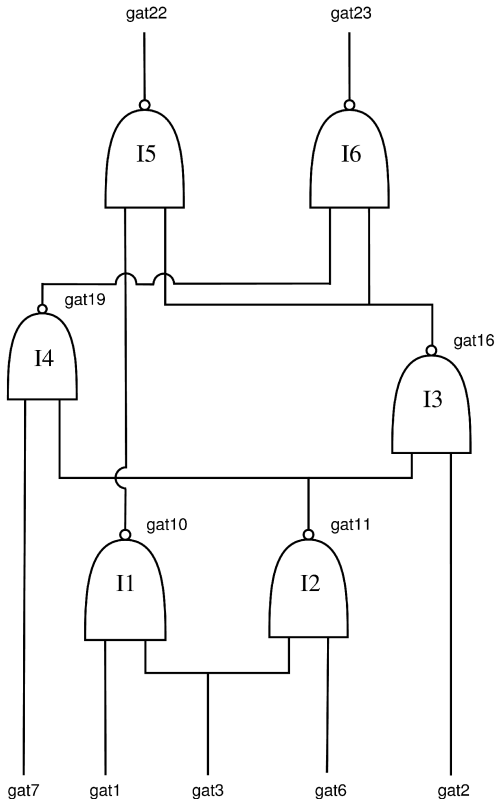


Figura 4.3. Circuito para el ejemplo 4.1.

Ejemplo 4.1. Descripción de circuito en formato Verilog.

```
`timescale 1ns / 1ps
module c17 ( gat1, gat2, gat3, gat6, gat7, gat22, gat23);

     input gat1, gat2, gat3, gat6, gat7;
     output gat22, gat23;

     NA2 I1 ( .Q(gat10), .A(gat1), .B(gat3));
     NA2 I2 ( .Q(gat11), .A(gat3), .B(gat6));
     NA2 I3 ( .Q(gat16), .A(gat2), .B(gat11));
     NA2 I4 ( .Q(gat19), .A(gat11), .B(gat7));
     NA2 I5 ( .Q(gat22), .A(gat10), .B(gat16));
     NA2 I6 ( .Q(gat23), .A(gat16), .B(gat19));

endmodule
```

4.4.2. Ficheros de patrones

Se entenderá por patrones el conjunto de estímulos iniciales a los que se somete el circuito en sus nodos de entrada. HALOTIS utiliza un formato en modo texto tal que, la forma de onda queda definida por un conjunto de tripletes formados por: nombre de la entrada, instante y tiempo de transición. A este formato genérico y simple se convierten conjuntos de patrones escritos en otros formatos, como por ejemplo MACHTA [MACH00], utilizando un simple programa de conversión (incluido en la primera versión de HALOTIS). En los ejemplos 4.2 y 4.3 se muestra la conversión de un formato a otro.

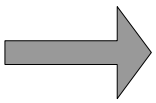
Ejemplo 4.2. Formato MACHTA

```
CODEFILE
UNITS ns
RISE_TIME 0.1
FALL_TIME 0.1
INPUTS
gat1,gat2,gat3;
OUTPUTS gat22,gat23;

CODING(ROM)

@0 <000>00;
@5 <011>01;
@10 <111>01;
@15 <011>00;
@20 <000>01;
```

hmachta



Ejemplo 4.3. Formato HALOTIS

```
gat1    0   0         0
gat2    0   0         0
gat3    0   0         0
gat2    1   0.5e-9    1e-10
gat3    1   0.5e-9    1e-10
gat1    1   1e-9      1e-10
gat1    0   15e-9     1e-10
gat2    0   20e-9     1e-10
gat2    0   20e-9     1e-10
```

Mientras el formato MACHTA contiene una serie de elementos como cabeceras (para especificar como interpretar el contenido del fichero de patrones), el formato propuesto para HALOTIS simplifica la descripción, conteniendo sólo datos sobre la forma de onda en formato de texto. Este fichero de texto estará formado por líneas tales que:

- Cada línea corresponde a una transición la cual puede de subida o bajada.
- Cada línea está formada por 4 columnas separadas por espacios o tabuladores donde en estricto orden, cada columna tiene el siguiente significado:
  - **Columna 1**: Nombre de la conexión sobre la que se aplica el estímulo.
  - **Columna 2:** Tipo de transición: 0 → bajada, 1→ subida.
  - **Columna 3:** Instante inicial de la transición.

- **Columna 4:** Duración de la transición.

Este formato de texto es entrada directa para el motor de simulación de HALOTIS.

4.4.3. Librerías de celdas

Desde mediados de los 80 los fabricantes de ASIC implementan Celdas Estándar. Estas consisten en una serie de bloques funcionales diseñados por el fabricante y parámetros de conmutación conocidos, tales como los tiempos de propagación, capacidades e inductancias, que pueden ser utilizadas en herramientas de diseño comerciales. Esto da paso al diseño basado en Celdas Estándar.

La simulación de un circuito basado en una librería de celdas implica disponer de todos los parámetros necesarios, para el modelo de comportamiento con el que se simula, para cada una de las celdas. En primer lugar hay que disponer del modelo funcional para simular la función lógica de la celda especificada habitualmente mediante una ecuación lógica o tabla de comportamiento. En segundo lugar, hay que incluir los parámetros del modelo con el que se simulará. La mayoría de los fabricantes incluyen descripciones eléctricas de las celdas, éstas se pueden utilizar para extraer parámetros de diferentes modelos. El proceso de obtención de parámetros (caracterización) está automatizado para el modelo DDM, utilizando la herramienta AUTODDM [JUAN01b]. También para la obtención de los parámetros del modelo de intensidad se ha desarrollado otra herramienta de caracterización (AUTOTCM). Ambos procesos de caracterización se basan en simulaciones eléctricas de manera reiterada sobre las celdas de una librería, extrayendo así los parámetros de cada modelo. Los parámetros necesarios para cada celda en el entorno de simulación HALOTIS se concretan de la siguiente forma:

- Modelo Funcional:
  - Función lógica escrita en notación polaca con los operadores AND, OR y NOT, los cuales forman un conjunto completo de operadores lógicos.
  - Tabla de comportamiento contemplado todos los casos en las entradas y los valores de la salida.
- Modelo para el tiempo propagación normal $t_{p0} = D_x \times C_L + E_x \times \tau_{in} + F_x$: Los parámetros resultantes para transiciones de subidas son D_r, E_r y F_r. Para transiciones de bajada son D_f, E_f, y F_f. (ecuación 2.13).
- Modelo de pendiente de salida $\tau_{out} = G_x \times C_L + H_x$ (ecuación 2.14): siendo los parámetros para transiciones de subida G_r, H_r, y, los de bajada G_f, H_f

(ecuación 2.14).

- Modelo DDM: Los parámetros hacen referencia a las ecuaciones 2.11 y 2.12, quedando para transiciones de subida A_r, B_r, C_r y, para bajada A_f, B_f, C_f.
- Modelo de intensidad: Los parámetros hacen referencia a la ecuación 3.7, por tanto son Q y R.
- Parámetros de la celda comunes a los modelos: C_{in}, V_t.
- Parámetros comunes a todas las celdas: V_{TN}, V_{TP}, V_{DDH} y V_{DDL}.

Todo este conjunto de parámetros es leído por HALOTIS antes de comenzar la simulación. HALOTIS utiliza un fichero de texto que contiene la descripción de cada celda junto con los valores de todos los parámetros necesarios. En la primera versión de HALOTIS la sintaxis del fichero de celdas utilizaba un formato propio, mostrado en el ejemplo 4.4. El número de celdas en una librería suele ser elevado, además, por cada combinación entrada/salida hay que indicar un conjunto completo de parámetros. Al incluir celdas de varias entradas y salidas, el tamaño del fichero crece considerablemente, por tanto, se opta por un preprocesado del fichero de texto usando un programa independiente que aporta, entre otras funcionalidades, la detección de errores en el propio fichero.

Ejemplo 4.4. Librería de celdas en formato de texto simple.

```
BEGIN_CELL("na2",2,1);
     INPUT_NAME(0,"a");
     INPUT_NAME(1,"b");
     OUT_NAME(0,"q");

     // Expresión de la salida
     EXPRESION(0,NAND(INPUT(0),INPUT(1)));

     // Parámetros de la entrada 0
     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(2e-6+2e-6)*LMIN*COX);
     PARAMETER(Cinf,(2e-6+2e-6)*LMIN*COX);
     PARAMETER(Af,177.033e-12);
     PARAMETER(Bf,10.3333e3);
     PARAMETER(Cf,0.882377);
     PARAMETER(Ar,64.6769e-12);
     PARAMETER(Br,4.41094e3);
     PARAMETER(Cr,1.31169);
     PARAMETER(Dr,4.20692e3);
     PARAMETER(Er,0.201717);
     PARAMETER(Fr,55.4206e-12);
     PARAMETER(Df,2.77464e3);
     PARAMETER(Ef,0.0825482);
     PARAMETER(Ff,42.3562e-12);
```

```
      PARAMETER(Gr,10340.8);
      PARAMETER(Hr,1.33747e-10);
      PARAMETER(Gf,6054.05);
      PARAMETER(Hf,8.64447e-11);
      ASSIGN_MODEL(0,0);

      // Modelo para la segunda entrada
      PARAMETER(Af,239.906e-12);
      PARAMETER(Bf,9.48001e3);
      PARAMETER(Cf,0.816024);
      PARAMETER(Ar,86.6308e-12);
      PARAMETER(Br,5.17971e3);
      PARAMETER(Cr,1.63747);
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(2e-6+2e-6)*LMIN*COX);
      PARAMETER(Cinf,(2e-6+2e-6)*LMIN*COX);
      PARAMETER(Dr,4.15532e3);
      PARAMETER(Er,0.213299);
      PARAMETER(Fr,93.0931e-12);
      PARAMETER(Df,2.7454e3);
      PARAMETER(Ef,0.0184577);
      PARAMETER(Ff,61.0572e-12);
      PARAMETER(Gr,10319.5);
      PARAMETER(Hr,2.15881e-10);
      PARAMETER(Gf,6012.03);
      PARAMETER(Hf,8.8294e-11);
      ASSIGN_MODEL(0,1);
END_CELL;
```

El programa encargado de tratar este fichero de celdas se denomina *hcells* y genera un fichero binario cuando tiene éxito la compilación de las celdas quedando libre de errores. La ventaja de los ficheros binarios, en general, es la carga directa en memoria sin ningún tipo de comprobación. El motor de simulación al trabajar con estos ficheros binarios ahorra un tiempo considerable de ejecución, a veces, equiparable al tiempo de simulación en pequeños circuitos.

En la segunda versión de HALOTIS se incluye otro formato de especificación de librerías de celdas basado en un estándar XML. XML proviene de un lenguaje inventado por IBM en los años '70, llamado GML (*General Markup Language*), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje fue normalizado por ISO en 1986 creando SGML [BRYAN88] (*Standard General Markup Language*) y es capaz de adaptarse a un gran abanico de problemas. A partir de él se han creado otros sistemas para almacenar información. Las siguientes particularidades de este lenguaje lo hacen candidato para el uso en la especificación de librerías de celdas:

- Es un lenguaje fácil de leer y editar en cualquier procesador de textos sin previos conocimientos del mismo (utilizando plantillas de ejemplo).

- Está normalizado y es conciso desde el punto de vista de los datos y la manera de guardarlos.
- Es extensible y se puede utilizar en todos los campos del conocimiento.
- Es fácil de implantar y programar, existen librerías para la lectura del mismo (*libxml* [LIBXML]), al enlazarlas con HALOTIS resuelve el problema del tratamiento de estos ficheros, automatizando incluso la detección de errores.

Estas características lo hacen un buen candidato para su uso en HALOTIS y su uso en el programa *hcells*. El ejemplo 4.5 muestra un ejemplo de descripción de un inversor en este formato.

Ejemplo 4.5. Librería de celdas en formato XML.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<!DOCTYPE library PUBLIC "http://www.dte.us.es"
"hcells.dtd">
<library>
  <name>AMS035</name>
  <modelname>iddm</modelname>
  <vddh>3.3</vddh>
  <vddl>0</vddl>
  <param name="vtn">0.81804</param>
  <param name="vtp">2.7622</param>
  <cells>
    <logiccell>
      <name>in1</name>
      <input>a</input>
      <out>
        <name>q</name>
        <expression>not(a)</expression>
      </out>
      <model input="a" out="q">
          <param name="Threshold">1.65</param>
              <param name="Cinr">4.032415e-15</param>
              <param name="Cinf">4.032415e-15</param>
              <param name="Af">94.1408e-12</param>
              <param name="Bf">10.1678e3</param>
              <param name="Cf">0.926382</param>
              <param name="Ar">32.005e-12</param>
              <param name="Br">5.24642e3</param>
              <param name="Cr">1.05815</param>
              <param name="Dr">4.33892e3</param>
              <param name="Er">0.176655</param>
              <param name="Fr">30.4103e-12</param>
              <param name="Df">3.64233e3</param>
              <param name="Ef">0.0946134</param>
              <param name="Ff">32.1833e-12</param>
              <param name="Gr">9916.13</param>
              <param name="Hr">8.86367e-11</param>
```

```
            <param name="Gf">7005.88</param>
            <param name="Hf">7.01589e-11</param>
        </model>
    </logiccell>
  </cells>
</library>
```

La definición de los elementos permitidos en el formato XML de las librerías de celdas de HALOTIS se establecen formalmente en el DTD[4] *hcells.dtd,* tal y como indica el estándar XML. En el anexo 1 se muestran las librerías de celdas empleadas en las diferentes versiones de HALOTIS y, el DTD que define el formato utilizado.

4.4.4. Salida y presentación de datos

El motor de simulación genera tres tipos de datos: formas de onda de tensión, forma de onda global de intensidad y datos estadísticos. Las formas de onda de tensión pueden ser referentes a cualquier salida del circuito o nodo interno. La forma de onda de intensidad calcula el consumo de intensidad global durante todo el tiempo de simulación. Con el fin de optimizar el uso de memoria del simulador, las formas de onda son requeridas al usuario previamente a la simulación, en la propia línea de comandos de HALOTIS.

Por cada nodo o salida seleccionado, HALOTIS guarda los datos de la forma de onda en un fichero de texto cuyo nombre es el nombre dado en el *netlist* al nodo/salida. El fichero de texto contiene una representación de tiempo frente a tensión escrita en dos columnas numéricas en formato científico, tal y como se muestra en el ejemplo 4.6. También, para la curva de intensidad se utiliza este formato, siendo en este caso el nombre del fichero "*current.hwav*".

Ejemplo 4.6. Salida de HALOTIS en formato de texto para una onda de tensión.

```
0       0
1.0805e-09      0
1.0805e-09      0
1.1127e-09      0
1.1127e-09      0
1.15722e-09     8.50165e-05
1.15722e-09     8.50165e-05
1.17973e-09     0.000232205
```

4. *Document Type Definition*

```
1.17973e-09     0.000232205
1.18547e-09     0.000252818
1.18547e-09     0.000252818
1.21547e-09     0.000360557
1.21547e-09     0.000360557
1.23102e-09     0.000322406
1.23102e-09     0.000322406
1.28497e-09     0.000293038
1.28497e-09     0.000293038
1.29421e-09     0.000342271
1.29421e-09     0.000342271
1.295e-09       0.000350069
1.295e-09       0.000350069
1.29805e-09     0.00038042
1.29805e-09     0.00038042
```

Se elige este formato de representación de ondas por ser compatible con la mayoría de las herramientas, tanto de análisis numérico y matemático, como de visualización de formas de ondas. Ejemplos de estos programas son *xgraph, xplot, grace, octave, gnuplot, labplot, etc.* También, cualquier hoja de cálculo admite este tipo de entrada de datos. Las figuras 4.4 y 4.5 muestran un ejemplo de representación de forma de onda capturada con el programa *xgraph* y un ejemplo de importación con la herramienta de análisis matemático *labplot* respectivamente.

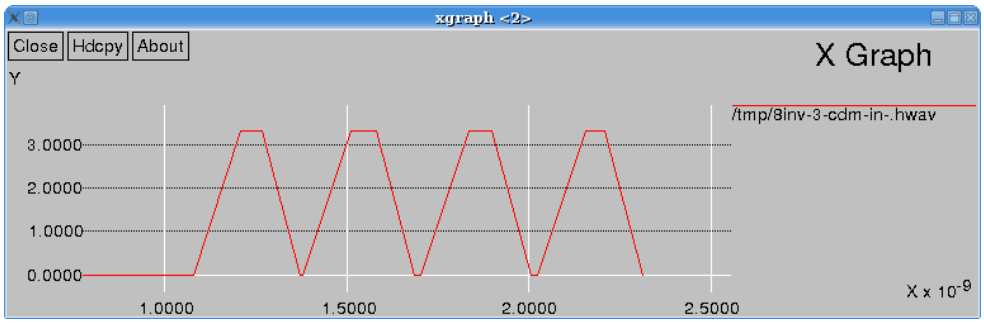


Figura 4.4. Ejemplo se captura de forma de onda de HALOTIS en el programa XGRAPH.

Además de formas de onda, HALOTIS guarda información sobre el propio proceso de simulación que denominaremos datos estadísticos. Los datos generados son los siguientes:

- Eventos filtrados durante el proceso de simulación.
- Número de transiciones de subida durante la simulación.
- Número de transiciones de bajada durante la simulación.
- Número de transiciones a estado desconocido durante la simulación.

- Número de patrones simulados
- Número total de eventos durante la simulación.
- Número total de eventos generados por los patrones
- Número de veces que se produce efecto de degradación.
- Número de eventos cancelados debido al efecto inercial y de degradación.

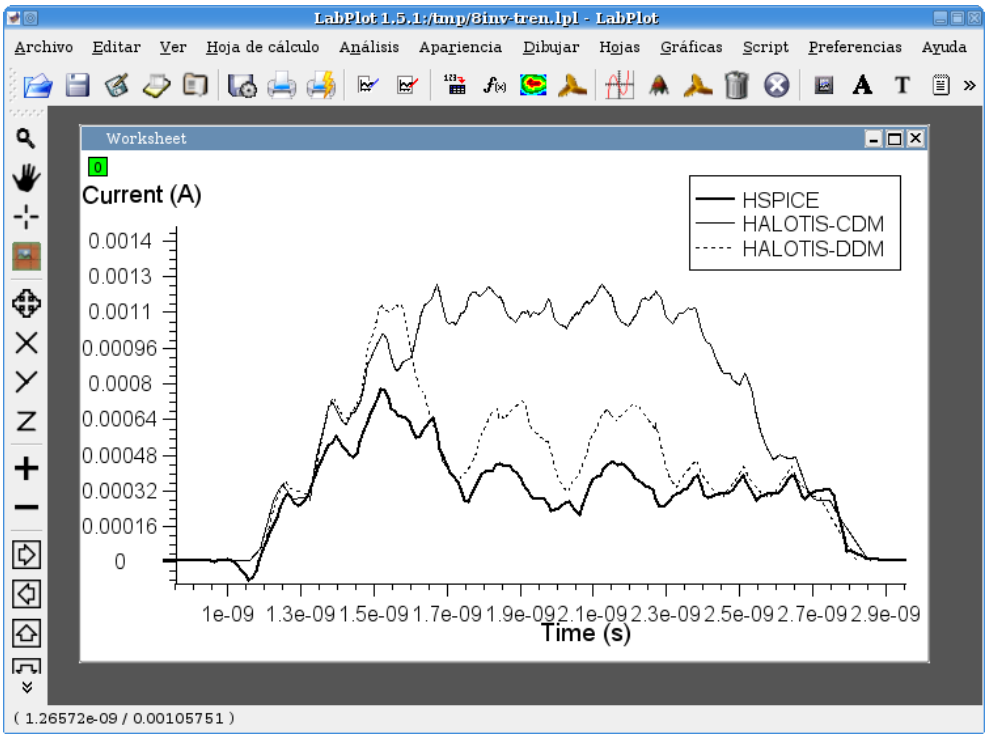


Figura 4.5. Ejemplo de captura de formas de onda de intensidad con el programa de análisis matemático LABPLOT.

4.4.5. Modo interactivo y de depuración

Como última característica, HALOTIS posee un modo especial de funcionamiento interactivo pensado para utilizarlo en circunstancias tanto de depuración del propio simulador o del modelo que se esté utilizando como en búsqueda de situaciones de fallo en los circuitos simulados.

Este modo permite al usuario ejecutar paso a paso el proceso de simulación y obtener datos en cada uno de estos pasos sobre nodos, transiciones, eventos y el propio estado del simulador. La figura 4.6 presenta la captura de un terminal en el que se ha ejecutado HALOTIS. El primer comando que se ejecuta en este ejemplo

es "*help*" y muestra un resumen de los comandos que admite HALOTIS cuando opera en ese modo.

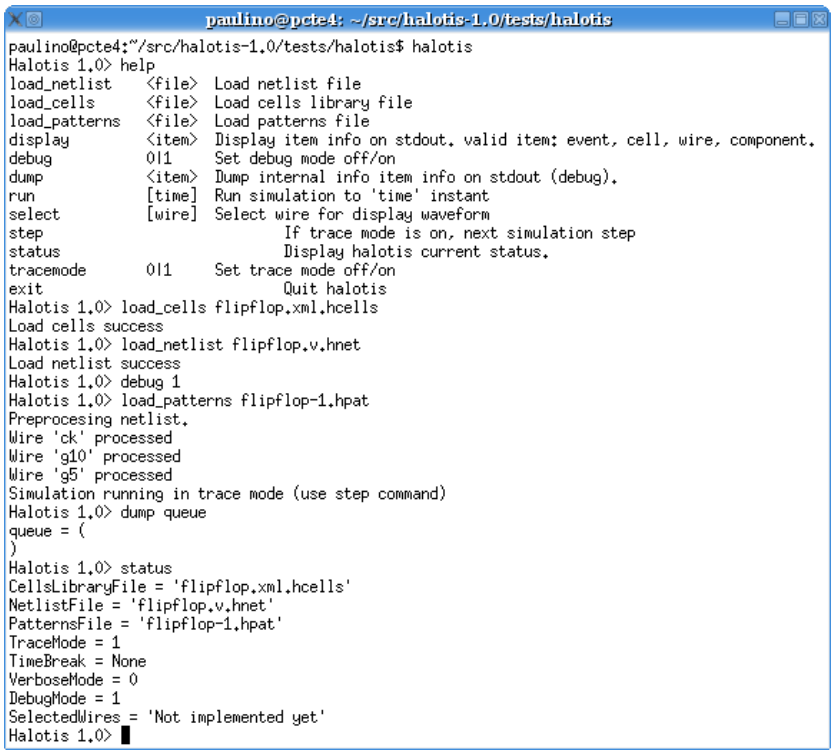


Figura 4.6. Funcionamiento de HALOTIS en modo interactivo

La relación de comandos admitidos son los siguientes:

- **help:** muestra un resumen de los comandos existentes, junto con su sintaxis.
- **load_netlist:** carga un *netlist* precompilado para simularlo.
- **load_cells:** carga una librería de celdas compiladas.
- **load_patterns:** analiza un fichero de patrones y encola todos los eventos generados por estos patrones en los nodos del circuito. No comienza la simulación.
- **debug:** activa el modo de depuración.
- **display:** muestra las propiedades de un elemento, pudiendo ser este elemento:
  - **event:** datos del primer evento de la cola de eventos.
  - **wire:** nodo interno, salida o entrada identificado por el nombre dado en el *netlist*.

- **component:** componente del *netlist* identificado por el nombre.
- **cell:** celda de la librería.

- **step:** procesa el primer evento que se encuentre en la cola, y para en el siguiente evento pendiente de procesar (ejecución paso a paso).
- **status:** muestra información interna sobre el estado del motor de simulación.
- **tracemode:** activa el modo de simulación paso a paso (comando *step*)
- **dump:** hace un volcado en pantalla completo de alguno de estos elementos:
  - **netlist:** volcado del *netlist* con todos los componentes y todas las conexiones.
  - **cells:** volcado de la librería de celdas completa.
  - **queue:** volcado completo de la cola de eventos

4.5. Modelado de HALOTIS

A lo largo de este apartado se mostrará una visión arquitectural y modular de HALOTIS. Siguiendo la metodología UML, la sección queda dividida en dos partes: la primera muestra una visión modular mediante diagramas de casos, en la segunda parte se realiza el modelado de objetos.

El modelado comienza presentado los componentes por los que está formado HALOTIS (figura 4.7). Todos los componentes están basados en una librería de clases que agrupa todos los datos y métodos disponibles en el código de HALOTIS, llamada a nivel de código *halotislib*. Esta librería es un es un conjunto de clases y funciones comunes a todo el entorno de simulación HALOTIS.

Antes de profundizar en la estructura de los componentes de HALOTIS y de la librería compartida, se muestra una descripción formal a nivel funcional de diferentes tareas que se realizan con este entorno en la siguiente subsección.

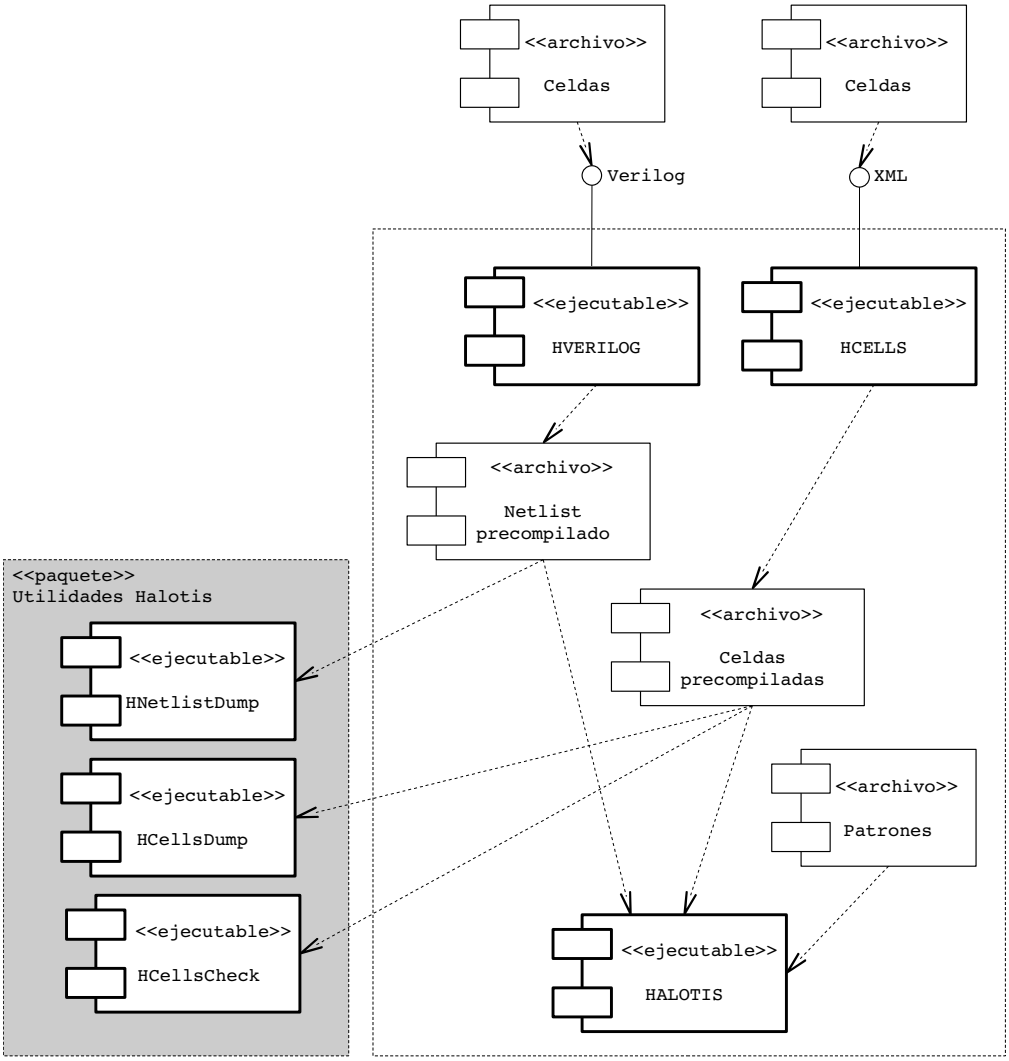


Figura 4.7. Diagrama de componentes de HALOTIS

4.5.1. Descripción funcional

La descripción funcional se presenta mediante diagramas de casos de usos en los que se descomponen las tareas que realiza el software en sus correspondientes subtareas sucesivamente. La figura 4.8 presenta las tres tareas de alto nivel que realiza el usuario al utilizar el entorno HALOTIS.

Las tres tareas ya fueron presentadas en la figura 4.1, en la que además, se indicaron los ficheros intermedios generados tras la compilación de las celdas y del *netlist*, ambos necesarios para realizar la simulación.

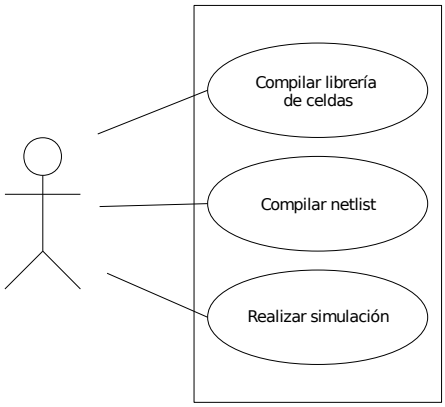


Figura 4.8. Diagrama de casos global.

La compilación de librerías corresponde a la herramienta *hcells* (también presentada al principio del capítulo) y se desglosa en la figura 4.9. La figura distingue dos ramas de procesamiento de librerías, una para formato de librería de celdas en texto plano, marcada en gris al ser el formato utilizado en la versión 1 de HALOTIS, y otra para el formato XML. Aunque aún se puede utilizar este el formato de texto, las librerías existentes ya han sido reescritas en XML, siendo éste el formato nativo para la versión 2 de HALOTIS. En la sección 4.4.3 se presentaron las mejoras de XML frente al formato texto, en este punto, la mejora más importante es la existencia de la librería XML (*libxml* [LIBXML]*)* que automatiza la lectura y detección de errores en ficheros XML, facilitando la programación de esta parte.

La segunda tarea es la compilación del *netlist* y se desglosa en la figura 4.10 presentando dos ramas, una para ficheros VERILOG y otra para ficheros VHDL. Desde la primera versión de HALOTIS el mayor esfuerzo se ha invertido en el desarrollo del analizador para el formato VERILOG. Esta tarea culmina con la herramienta *hverilog* y utiliza la librería de celdas compilada en la tarea anterior.

La descripción funcional finaliza presentando la tarea de simulación junto con un desglose en subtareas en la figura 4.11. La simulación es tarea más amplia, al utilizar los datos generados en las dos tareas anteriores, mas los patrones de simulación. Cabe destacar la existencia de dos algoritmos de simulación, utilizados según el modelo de comportamiento que se utilice durante la simulación, DDM o CDM.

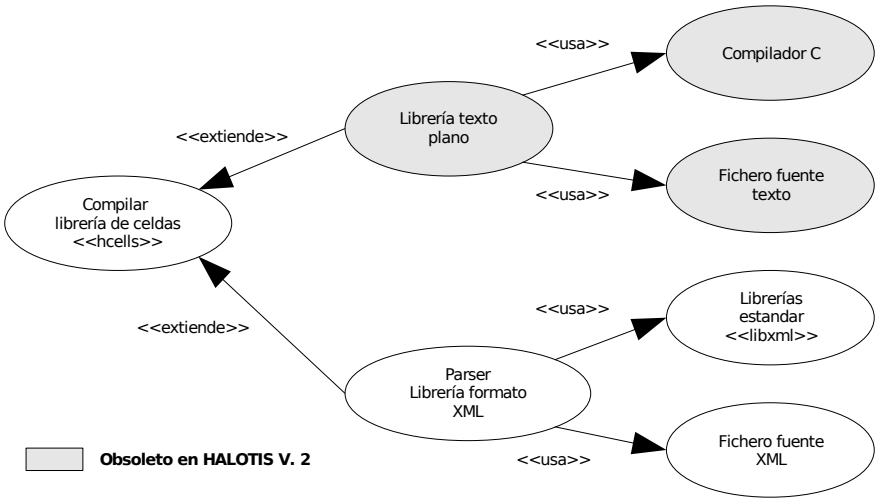


Figura 4.9. Diagrama de casos para la compilación de librerías de celdas.

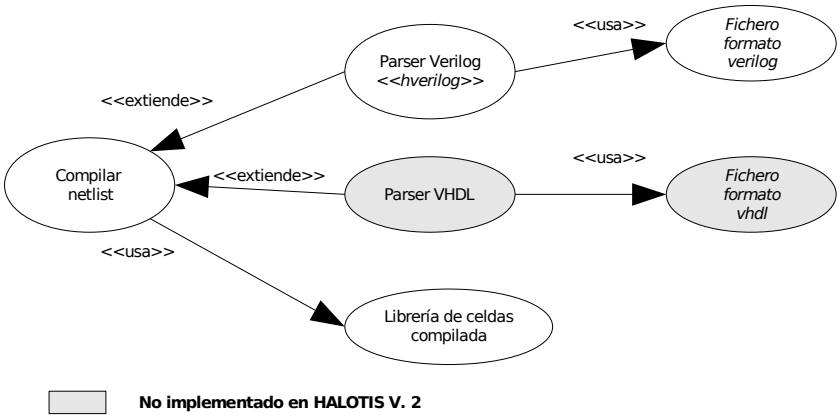


Figura 4.10. Diagrama de casos para la compilación del netlist.

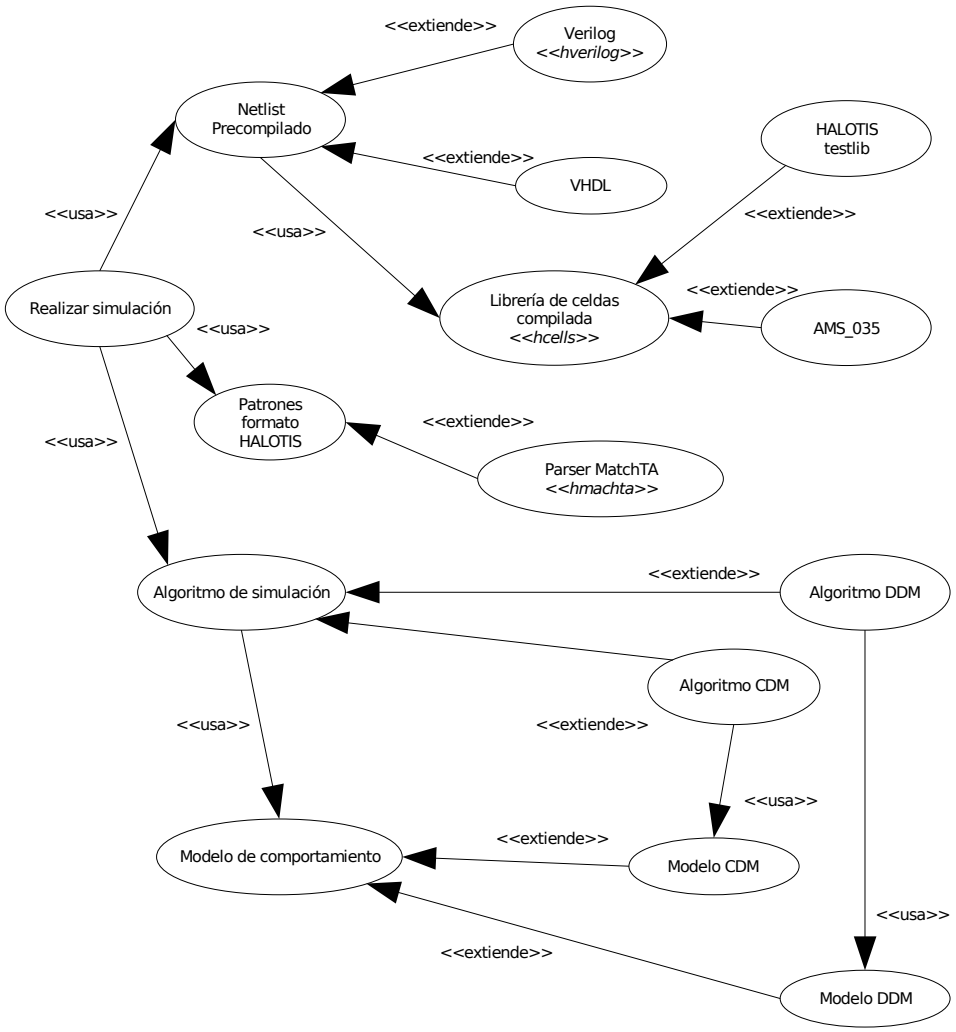


Figura 4.11. Diagrama de casos de la simulación con HALOTIS.

4.5.2. Modelo de objetos

El modelo de objetos presenta la estructura estática del sistema mostrando objetos, relaciones entre objetos, atributos y métodos o funciones. Una buena organización para la enumeración y descripción del conjunto de clases utilizadas en HALOTIS es utilizar la división global en tareas propuesta en la sección anterior y mostrada en la figura 4.8. Para cada tarea se mostrarán las clases asociadas junto con sus relaciones. Muchas de estas clases utilizadas no son exclusivas de una única tarea, por ejemplo, las clases utilizadas por el *netlist* son comunes a la simulación. El conjunto de clases comunes a más de una tarea o

herramienta software (ejecutable) serán albergadas en una librería común en el entorno HALOTIS. Por tanto, tras la descripción completa del modelo de objetos se presenta la librería *halotislib* que incluye toda la funcionalidad.

Es importante mencionar que en el diseño de software empleando metodologías orientadas a objetos la clave del éxito está en la identificación de las clases, ya que éstas permiten agrupar los datos. La granularidad en el conjunto de clases que modelan un problema es un aspecto muy importante en un diseño software. Mientras mayor sea está, es decir, más clases se identifiquen y mayor sea la generalización, el software será capaz de adaptarse un mayor abanico de problemas a resolver y, será fácilmente ampliable. Por el contrario, un exceso en la granularidad provoca un gran esfuerzo a cualquier técnico que pretenda incorporarse al desarrollo, ya sea para ampliar el software, corregir errores o incluso reutilizarlos en un futuro proyecto.

La segunda versión de HALOTIS ha aumentado considerablemente el número de clases existentes para modelar el sistema y, desde nuestro punto de vista, se ha alcanzado un compromiso correcto entre un número amplio de clases, que permite adaptar el software a un gran número de problemas pero sin llegar a ser un número excesivo que dificulte el diseño del mismo. Concretamente el diseño completo consta de 46 clases, de las cuales 29 son las completamente reutilizables e incluidas en la librería compartida *halotislib* y son la base de la continuación de HALOTIS. Este aspecto se detalla en la sección 4.6.2 con una figura y acompañada de una tabla con la distribución de clases en componentes.

El modelo de objetos que presenta en esta sección se ha dividido en 5 partes que son: *netlist*, librería de celdas, la celda con su modelo de comportamiento, el parser de VERILOG y la simulación. Estas agrupaciones son también válidas respecto al comportamiento durante el proceso de simulación. Las agrupaciones Netlist, Librería de celdas contienen objetos que son creados antes de comenzar la simulación y además permanecen estáticos, es decir, no cambian de estado a lo largo de todo el proceso. Diremos que está relacionado con la parte estática del sistema. En cambio, la agrupación simulación contendrá objetos que son creados y destruidos de forma dinámica y además cambian de estado. En este caso diremos que está relacionado con la parte dinámica del sistema.

Finalmente, todas las clases forman parte de un único diagrama de objetos global en la que todo está interrelacionado y muestra el diseño completo del entorno HALOTIS.

4.5.2.1. Modelado del netlist

El *netlist* queda modelado por las clases de la figura 4.12, la cual, consta de

una clase principal *HNetlist* enlazada con el resto de las clases. HALOTIS representa el *netlist* mediante un conjunto de componentes (*HComponent*) que a su vez, son instancias de una definición de celda lógica implementada en la clase *HCell*. Las conexiones se modelan con la case *HWire* relacionada con la case *HPad,* que representa un pad de un determinado componente del circuito. La conexión utiliza una estructura de datos agregada (*Node)* para almacenar el conjunto de conexiones de la clase *HWire*.

El conjunto de clases de la figura 4.12 se detallan a continuación:

- **HNetlist**: Clase que engloba el *netlist* completo, incluye un método (*read*) que crea la estructura completa de objetos en memoria leyéndola desde un fichero binario compilado. Incluye todos los componentes que forman parte del circuito, las conexiones que los interconectan y está relacionado necesariamente con una librería de celdas.
- **HComponent:** Hace referencia a un componente del *netlist*. Está asociado a una celda de la librería de celdas, la cual, define su comportamiento dinámico y los pads de entrada/salida. También almacena datos durante la simulación, como el estado lógico o las variables de estado para componentes secuenciales.
- **HPad:** La clase *HComponent* está formada por un conjunto de clases de este tipo que representan cada uno de los pads que forman parte de un componente. Relaciona las conexiones del *netlist* (*HWire*) con los componentes. También incluye datos referentes al proceso de simulación, como son los eventos producidos en él. Se distinguen tres tipos:
  - **HIPad**: pad de entrada.
  - **HOPad**: pad de salida.
  - **HIOPad**: pad de entrada/salida.
- **HWire**: Son las conexiones entre los componentes, pero no se asocian directamente a ellos, sino a través de la los pads (clase *HPad).* Almacenan parámetros necesarios para la simulación, como es la capacidad de los nodos a los que está conectado. También guarda datos sobre la forma de onda generada en dicha conexión para un posterior volcado.
  - **Node**: Estructura de datos definida dentro de la clase *HWire* para almacenar las conexiones entre los componentes y la conexión. Permite navegar de forma directa al pad o al componente.

La descripción de la clase *HCell* de la figura 4.12 se presenta en la subsección 4.5.2.3 tras el modelado de la librería de celdas.

4.5.2.2. Modelado de la librería de celdas

Toda descripción de un circuito digital en HALOTIS queda ligado a una determinada librería de celdas, tras la compilación del *netlist*. Esta librería contiene la definición de las celdas que se pueden instanciar en el *netlist*, así como

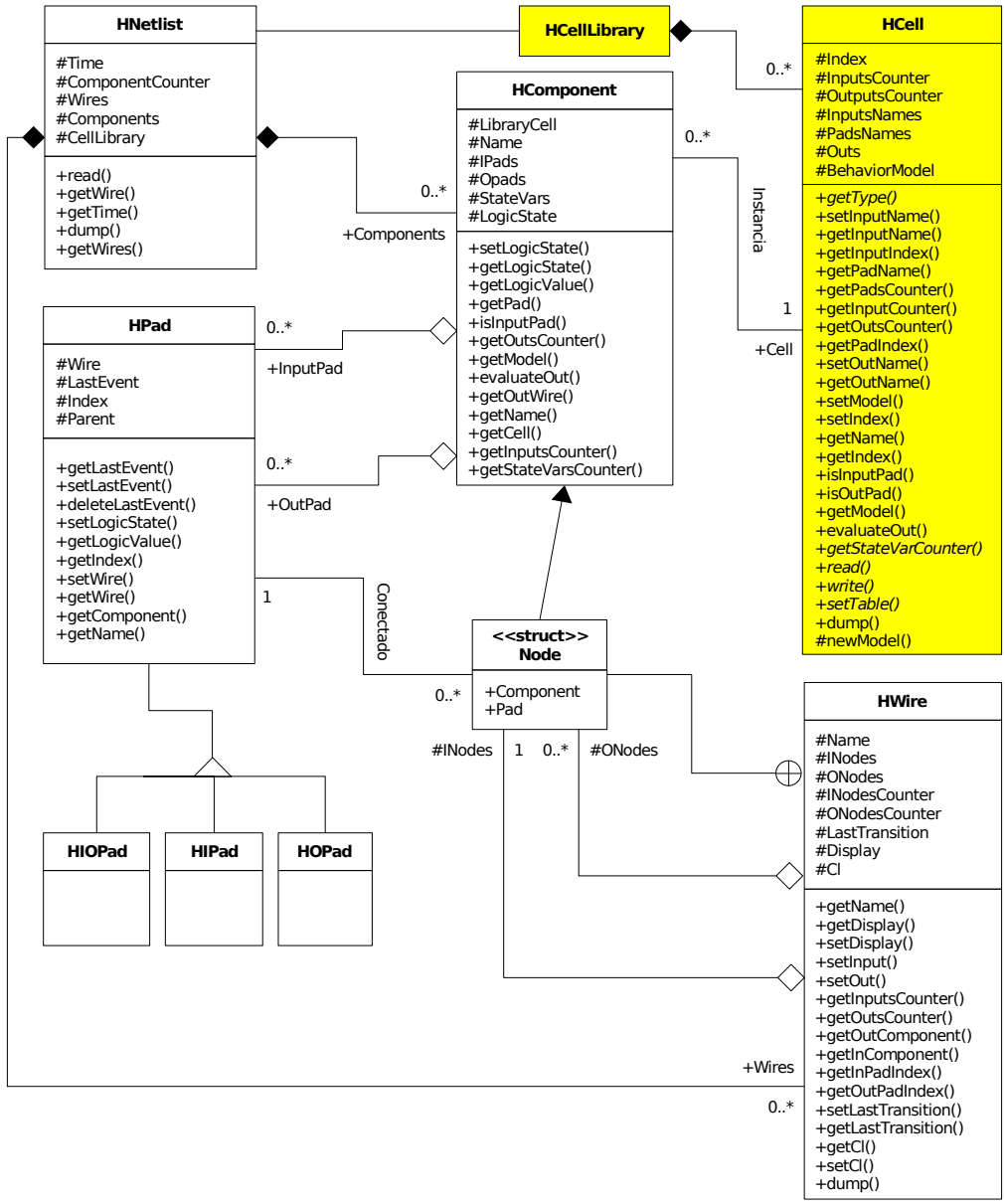


Figura 4.12. Diagrama de objetos para el netlist.

la descripción de la función que realizan. Los valores de los parámetros para un determinado modelo de comportamiento también quedan incluidos en la definición de la librería. Estos parámetros caracterizan cada una de las celdas y crea una fuerte dependencia entre la simulación y la librería de celdas. Es comprensible esta dependencia, ya que la inclusión o alteración de un modelo de comportamiento puede provocar la alteración de los parámetros de las celdas existentes en la librería.

Ante la necesidad de disponer en el sistema de diferentes librerías asociadas a los fabricantes, se opta por realizar el modelado de esta parte del sistema como un paquete separado del motor de simulación. Este paquete incluye los programas: *hcells, hcellscheck y hcelldump,* descritos la sección 4.2 de este capítulo y procesa las descripciones de las celdas que forman la librería en diferentes formatos de texto también descritos anteriormente: XML ó HALOTIS-TXT.

Una vez realizado el preprocesado y la compilación de una librería descrita por un usuario, HALOTIS carga esta librería en memoria usando la clase *HCellLibrary*. La figura 4.13 presenta el conjunto clases que forman la librería de celdas siendo la clase central *HCellLibrary.* Por si misma, esta clase sólo presenta una interfaz para el tratamiento de la librería en memoria, quedando la escritura y lectura soportada en la clase hija *HCellLibraryFile*. La escritura y lectura de la primera clase hija es para formato binario, otra clase derivada se especializa en el tratamiento de ficheros en formato XML (*HCellLibraryXML*). La estructura de herencia de la librería permite utilizar sólo la clase necesaria en cada programa del entorno HALOTS, es decir, el motor de simulación no necesita incluir la lectura en formato XML, mientras que el compilador de celdas si lo utilizará.

En la figura 4.13 también aparece la case *HCell* que es donde realmente queda almacenada cada una de las celdas. Por su complejidad, es tratada en la siguiente sección junto con otras clases que dan soporte a la celda y que también se engloban en la librería de celdas.

Por último, se enumeran a continuación todas las clases que intervienen en la figura 4.13 detalladamente:

- **HCellLibrary**: Contenedor de todas las celdas de la librería. Incluye métodos para la búsqueda de celdas y los parámetros comunes de la librería, como son tensiones de alimentación. Esta clase actúa como superclase para la otras que añaden funcionalidad para el tratamiento de ficheros en diferentes formatos: binarios, XML o HALOTIS-TXT. La clase se apoya en la estructura de datos *HModelType,* como atributo agregado, para distinguir el modelo de comportamiento con el que se puede simular.
- **HCellLibraryFile**: Añade métodos para lectura y escritura de ficheros

compilados binarios a la clase básica contenedora de celdas *HCellLibrary,* mediante el mecanismo de generalización o herencia. La utilidad de esta clase es doble, por un lado es capaz de escribir, en formato binario, una librería de celdas mediante el método *write,* y con ello, es útil para el compilador de celdas. Por otro lado, el motor de simulación la utilizará para leer del disco la librería de celdas compilada previamente.

- **HCellLibraryXML:** Esta clase añade la funcionalidad a la clase anterior mediante herencia. Es utilizada en el compilador de librerías de celdas *hcells* para realizar el análisis sintáctico y léxico del fichero XML que representa la librería. En el diagrama aparece una dependencia con una librería externa *libxml,* librería de software abierto para el tratamiento XML escrita en C [LIBXML].

- **HBinaryFileHeader**: Ha sido creada para conseguir uniformidad en los ficheros binarios. Todos los ficheros binarios manejados por HALOTIS incluyen una cabecera controlada por esta clase, de forma que, toda clase que manipule ficheros binarios contendrá esta clase como agregada. Esta clase aporta métodos para escribir, leer e identificar los ficheros binarios, dotando a HALOTIS de un mecanismo común para el reconocimiento de ficheros binarios antes del procesamiento. Este es el caso de la clase *HCellLibraryFile* de la figura 4.13 que agrega esta cabecera como atributo. También en la figura aparece como agregado un nuevo tipo de datos *enum* (enumeración) para indicar el tipo de fichero binario con el que se trabaja en cada momento. Esta estructura de datos facilita la ampliación de HALOTIS para tratar otro tipo de ficheros necesarios en un futuro.

- **HModelStaticData**: Es una clase que almacena los parámetros comunes a toda la librería de celdas. Son datos que comparten todas las celdas como son tensiones de alimentación y tensiones umbrales de los transistores.

- **HCell**: Representa la celda lógica de la librería, se describe en la siguiente sección detalladamente.

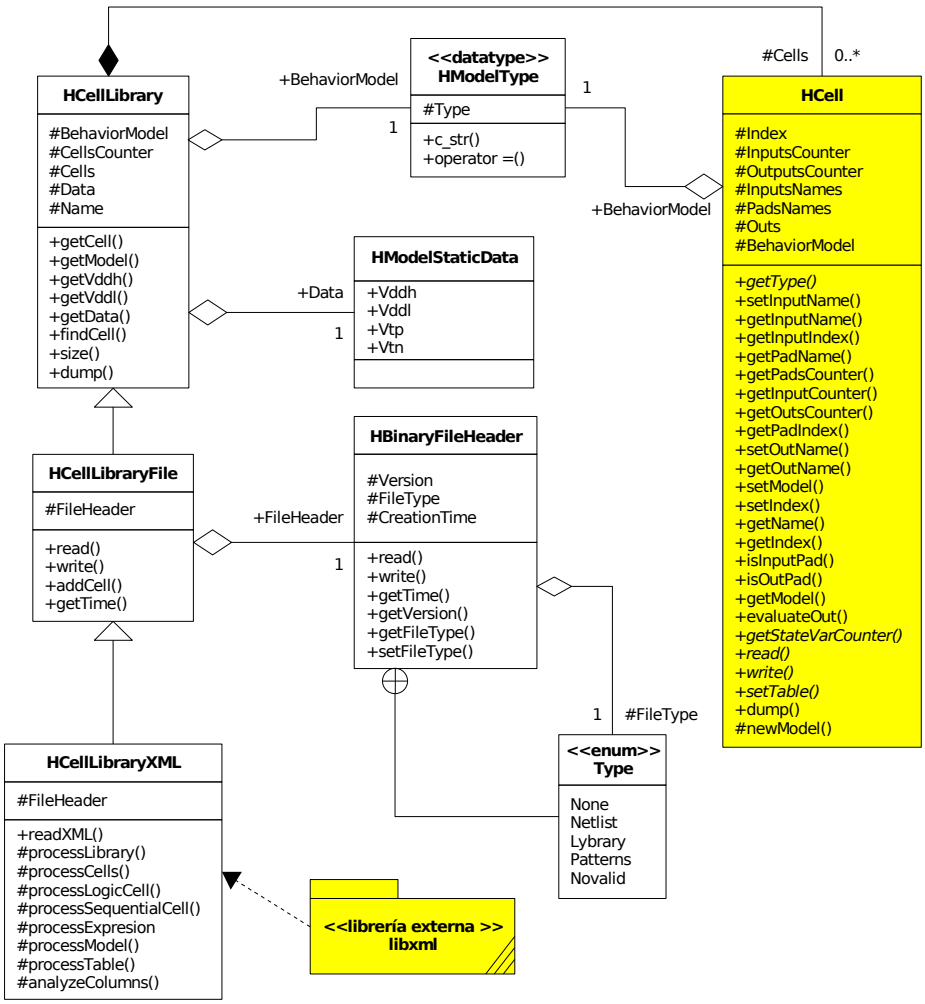


Figura 4.13. Diagrama objetos de la librería de celdas.

4.5.2.3. Modelado de la celda lógica en una librería de celdas.

El diseño de la celda lógica y el motor de simulación son las partes de mayor complejidad de HALOTIS. Muestra de ello es la cantidad de clases y relaciones que intervienen en la figura 4.14 donde se describe la estructura interna de representación de celdas utilizada en HALOTIS.

De la figura 4.14 se concluye que HALOTIS divide la celda lógica (*HCell)* en celdas y sus salidas (*HCellOut*). Cada salida tiene asociada dos clases, una que representa la expresión lógica (*HExpression*) y otra que contiene los valores de los parámetros del modelo de comportamiento (*HModel*). Cada combinación salida/entrada contiene un conjunto diferente de parámetros, por ello, la salida de la celda tienen asociado un conjunto de clases *HModel*. Por otro lado, la celda

lógica distingue dos tipos de celdas, las secuenciales y las combinaciones existiendo, una clase asociada a la celda secuencial para tratar la tabla de estados basada en variables de estado. También el modelo de comportamiento distingue varios tipos, concretamente tres actualmente implementados.

A continuación se detallan las clases de la figura 4.14:

- **HCell**: Clase que modela la celda lógica y asociada a todas las clases que contienen todas las propiedades de la misma. Contiene una gran cantidad de procedimientos para utilizar durante la compilación del *netlist* y la simulación. Se destacan los siguientes:
  - Procedimiento *evaluateOut:* Utilizado durante la simulación para ver el estado lógico de una salida en función del valor lógico de las entradas.
  - Procedimientos *write* y *read:* tras su ejecución escriben y leen de un fichero binario toda la estructura existente en memoria de la celda.
  - Procedimiento *getModel:* Devuelve los parámetros del modelo de comportamiento para esa celda, para una determinada combinación de entrada y salida.
  - Procedimientos para información sobre pads: Utilizados tanto en la simulación como en la compilación de *netlist*, estos son: *getPadsName, getPadsCounter, getInputsCounter, getOutsCounter, getPadIndex, isOutPad*.
- **HCellLogic/HCellSequential:** Son especializaciones de la clase *HCell* para tratar celdas secuenciales o lógicas. El comportamiento durante la simulación de estas celdas difiere, tanto en cuanto, las celdas secuenciales contienen variables de estado. Para la definición del comportamiento, a las celdas secuenciales se les ha añadido una clase relacionada *HTable*, que contiene una tabla de estados.
- **HCellOut**: Para cada salida de una celda existe un objeto de esta clase y, está enlazada con los datos del modelo de comportamiento (*HModel*) y la expresión lógica de la salida (*HExpresion*).
- **HModel**: Contiene el conjunto de parámetros del modelo de comportamiento y un método virtual (*createTransition*) que tras su ejecución aplica la ecuación del modelo con el que se está simulando. La implementación de este método se realiza en una clase hija especializada para cada modelo de comportamiento:
  - **HModelIdd**: Clase especializada para la simulación con en modelo de degradación (ecuaciones 2.11, 2.12, 2.13 y 2.14)

- **HModelTp0**: Clase para la simulación con el modelo de propagación normal sin incluir la degradación (ecuaciones 2.13, 2.14)
- **HModelAss**: Este modelo implementa sólo el modelo lógico del circuito con retraso unitario o cero.

- **HExpresion**: Es una estructura de datos que almacena y evalúa expresiones lógicas utilizando árboles binarios y con un conjunto de operadores predefinidos. Este conjunto de operadores es completo y quedan representados por el tipo de datos *NodeType*. Este tipo de datos contiene un identificador único para cada tipo de operador y se muestra en la tabla 4.1. También contiene dos métodos para la escritura y lectura (volcado) en formato binario, *read* y *write* respectivamente.
- **HStateVars**: Es una clase encargada de manejar la tabla de estados que define el comportamiento lógico de una celda secuencial. La tabla de estados junto con las variables de datos (tipo de datos *HStateVar*) se engloban en esta clase.

Identificador de nodo	**Tipo de nodo**	**Tipo de operador**	**Valor/es**
E_LITERAL	Literal lógico	Unario	0,1
E_INPUT	Entrada de puerta	Unario	Nombre de la entrada
E_AND	Operador lógico AND	Binario	Expresiones
E_OR	Operador lógico OR	Binario	Expresiones
E_NOT	Operador lógico NOT	Unario	Expresión
E_NOR	Operador lógico NOR	Binario	Expresiones
E_NAND	Operador lógico NAND	Binario	Expresiones
E_BRACKET	Paréntesis	Binario	Expresiones

Tabla 4.1. Expresiones admitidas en la definición de la ecuación lógica en la clase HExpesion.

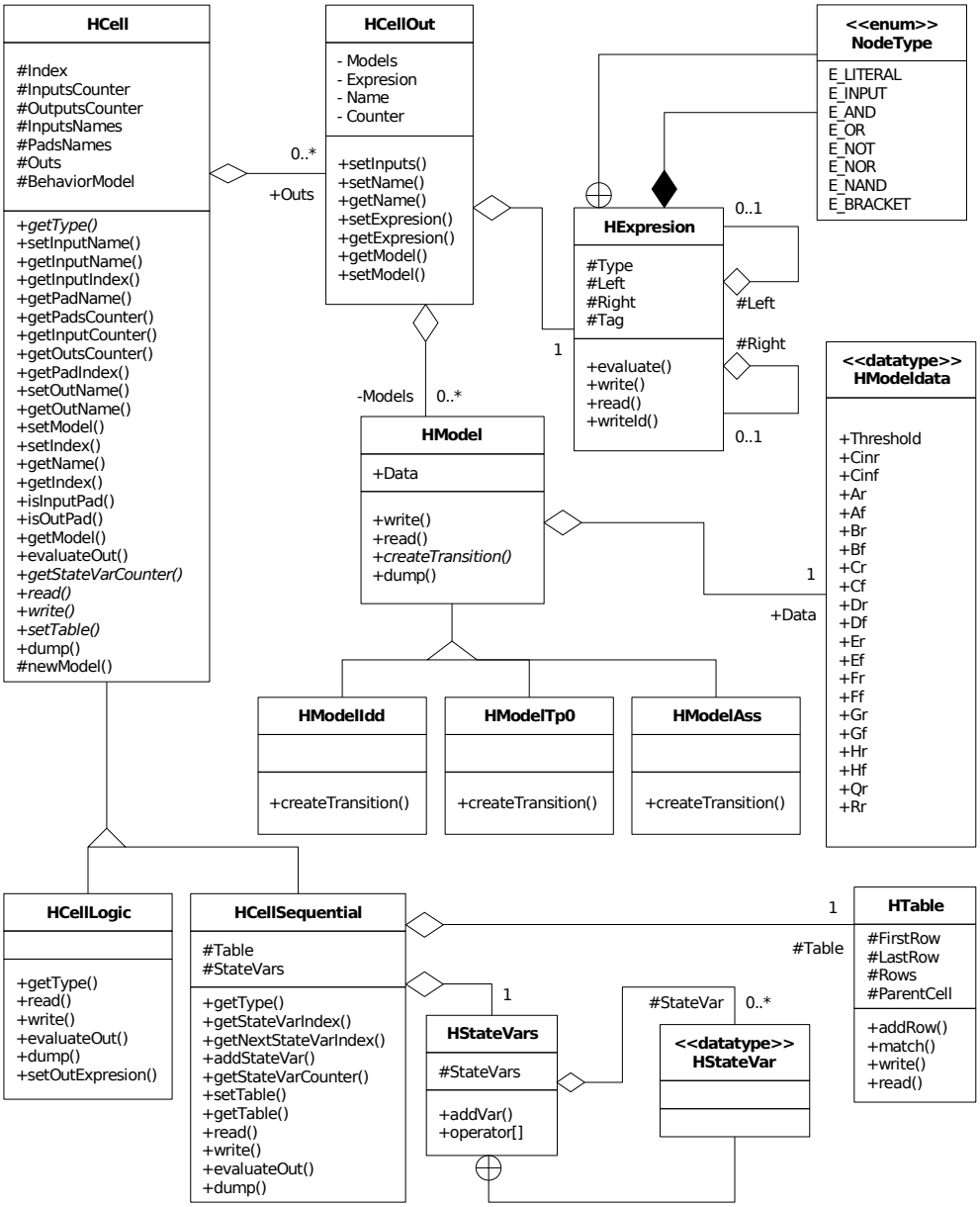


Figura 4.14. Diagrama objetos de una celda de la librería de celdas.

4.5.2.4. Modelado del parser VERILOG

Tradicionalmente para el análisis y procesado de gramáticas formales existen herramientas informáticas desarrolladas, cuya utilización se hace imprescindible cuando se procesa cualquier lenguaje formal en un programa. VERILOG es un lenguaje formal y se puede tratar informáticamente con estas herramientas.

Para la construcción de compiladores de lenguajes de alto nivel se utilizan autómatas de pila y autómatas finitos [ISASI01], que son máquinas de estado que realizan sobre el fichero de entrada un análisis para comprobar si el código fuente cumple las reglas de la gramática del lenguaje. Este análisis consiste en: análisis léxico, análisis sintáctico y análisis semántico. Tras la finalización con éxito del análisis, el código se considera libre de errores, y se puede generar el resultado del procesado. Aunque esta generación se contempla como una fase posterior al análisis, habitualmente la generación se realiza junto con el análisis, abortándose en el caso de fallar la sintaxis del fichero.

El la herramienta *hverilog*, encargada de interpretar el código VERILOG ha sido diseñada utilizando el paradigma clásico de desarrollo de compiladores, en el que se utilizan las herramientas también clásicas LEX y YACC [LEVI92] o sus sucesoras FLEX++, BISON++ para programación orientada a objetos y con licencia GPL. Respectivamente estas herramientas cubren las tareas de análisis léxico y sintáctico. En la sección de implementación, incluida en este capítulo, se analizan estas herramientas.

En el modelado de la herramienta *hverilog* representado en la figura 4.15 se mezclan clases del entorno HALOTIS con otras que proceden o son envolturas de otras clases generadas por herramientas de tratamiento de la gramática. La clase *HVerilogParser* engloba el parser[5] completo e incluye la clase *LexInfo* como soporte para la detección de errores. Ambas clases son envolturas de clases generadas por las herramientas de análisis gramatical utilizadas, siendo muchos de sus métodos llamadas a trozos de códigos autogenerados.

El resto del diagrama son un conjunto de clases y tipos de datos utilizados para construir el *netlist* durante el procesado del fichero VERILOG. Los tipos de datos *map* son estructuras árbol para indexar elementos, bien por el nombre o por el identificador. Estos mapas son necesarios para conseguir que tanto las conexiones como las celdas y los componentes, queden inequívocamente identificados por un nombre, proveniente del nombre dado el fichero VERILOG. Con todo esto, *hverilog* utiliza estas estructuras para acceder rápidamente a estos elementos a través de su nombre. Por último, para el tratamiento de los subcircuitos aparece la clase *HSubCircuit* y se encarga de mantener la definición de un subcircuito para crear instancias cada vez que aparece su nombre. Hay que considerar que HALOTIS sólo simula circuitos digitales formados únicamente por componentes, por tanto, el resultado final del procesado (compilación) de un *netlist* en cualquier formato HDL será una representación del *netlist* plano. Un *netlist* plano significa que, todos los subcircuitos deben instanciarse componente por componente, no quedando en el *netlist* compilado un subcircuito como tal.

5. Se entiende por parser un software capaz de hacer las tres tareas del análisis de una gramática: léxico, sintáctico y semántico

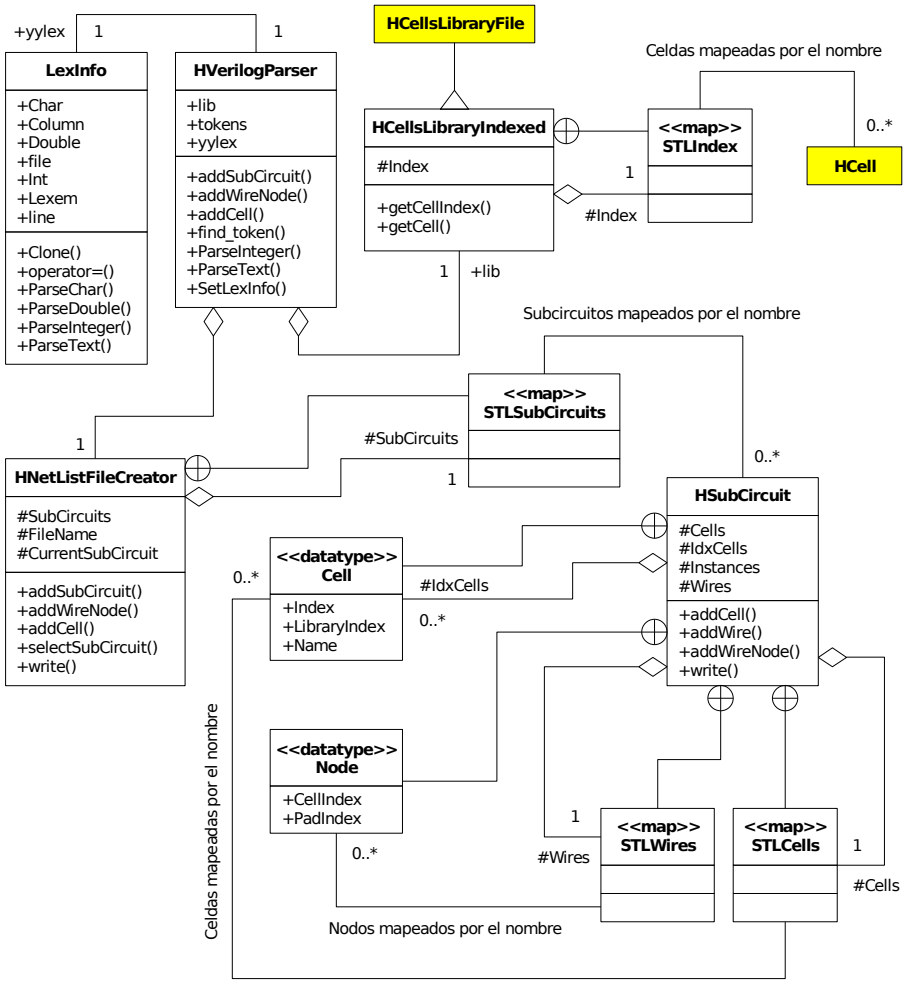


Figura 4.15. Diagrama objetos de una celda de la librería de celdas.

A continuación de detallan las clases del diagrama de la figura 4.15:

- **HVerilogParser**: Es una envoltura del parser de VERILOG, enlaza con el analizador léxico mediante el atributo *yylex* (clase *LexInfo*), incluye la librería de celdas compilada (clase *HCellsLibraryIndexed*) y genera el *netlist* mediante la clase *HNetlistFileCreator*.

- **LexInfo**: Clase para almacenar los datos de las palabras de la gramática VERILOG que se van analizando. Se utiliza como clase por defecto del parser para detectar el tipo de palabra y su posición en el fichero de texto mediante los atributos *line* y *column*. Se distinguen los siguiente tipos: carácter (*char*), número en notación científica (*double*), entero (*integer*) y texto (*text*).

- **HCellsLibraryIndexed**: Es una clase hija de la librería estándar de HALOTIS (*HCellsLibraryFile*) y añade un índice de nombres a la librería para realizar búsquedas rápidas de las celdas a las que hace referencia cada componente del fichero VERILOG. Por defecto, la librería de HALOTIS no trabaja por nombre, sino por índices enteros para mayor tener velocidad en el motor de simulación, por ello, se justifica la existencia de esta clase.
  - **STLIndex:** Es la estructura que indexa mediante un árbol binario equilibrado la librería de celdas. STL[6] hace referencia a la librería estándar utilizada para su implementación (véanse los detalles de la implementación en la sección 4.6.1).
- **HNetlistFileCreator**: Se encarga de ir construyendo el *netlist*, para ello, instancia las clases que lo forman. Incluye subcircuitos con la capacidad de instanciarse por completo cada vez que sea necesario y así, obtener un *netlist* final plano. Mediante el método *write* se escribe en formato binario el *netlist* compilado cuando se ha terminado de construir en memoria.
- **HSubCircuit**: Mantiene información sobre los subcircuitos que leídos en VERILOG para aplanarlos cuando aparezcan de nuevo en el fichero. Todos estos datos se guardan en estructuras que representan los nodos (*Node*) y los componentes (*Cell*). Estas estructuras son indexadas utilizando de nuevo árboles binarios de la librería STL. También, las conexiones quedan almacenadas mediante la estructura *STLWires* y, consisten en un conjunto de nodos pertenecientes a la conexión. Las estructuras nodo y celda contienen los siguientes elementos:
  - **Node**: Representa un nodo de una conexión (*HWire*). En el nodo se enlaza la conexión y un pad de una celda. El nodo almacena el índice del pad en el atributo *PadIndex* y el índice de la celda en el atributo *CellIndex*.
  - **Cell**: Representa el componente y contiene un índice a la celda de la librería correspondiente (*LibraryIndex*) y el nombre de la instancia dado en el fichero VERILOG.

Toda la infraestructura mostrada es reutilizable para cualquier otro lenguaje HDL que se desee incluir en HALOTIS. Sólo hay que generar un parser nuevo y reescribir la clase *HVerilogParser* para adaptarla a la nueva gramática.

6. Standard Template Library

4.5.2.5. Modelado de la simulación

Tanto el *netlist* como la librería de celdas descritas en las secciones anteriores son estructuras de datos que permanecen estáticas durante el proceso de simulación, es decir, una vez alojadas en memoria, el motor de simulación sólo consulta los datos existentes no los modifica. Durante el proceso de simulación las estructuras dinámicas que se crean y destruyen son: eventos, transiciones y formas de onda.

Las relaciones entre la parte dinámica y la estática durante el proceso de simulación queda representado gráficamente en la figura 4.16. Los eventos (*HEvent*) durante la simulación se almacenan en una cola de eventos (*HEventQueue*) para su procesado en estricto orden temporal. Un evento está relacionados con un único pad de un componente, a su vez, este evento tiene acceso a los datos de la transición que lo generó (*ParentTransition*) y se utiliza para el cálculo en las ecuaciones del modelo con el que se simula. Las transiciones están asociadas a la conexión en la que tienen lugar, quedando, si fuera necesario, almacenadas para reconstruir la forma de onda cuando la simulación termine.

HALOTIS sólo guarda las transiciones de las conexiones solicitadas por el usuario puesto que, en simulaciones largas, se necesitaría gran cantidad de memoria si se quisieran guardar todas.

Las clases que intervienen en la figura 4.16 son:

- **HEvent:** Almacena el instante en el que se produce el evento y el valor lógico. Estos objetos se crean dinámicamente durante la simulación y se almacenan en la cola de eventos (*HEventQueue*). Contienen un identificador único (*UID*)[7] utilizado para poder localizar un evento en la cola y poder borrarlo, puesto que el algoritmo de simulación de algunos modelos de comportamiento así lo requieren. Las relaciones indicadas hacen referencia al pad, al componente en el que se produce y la transición que lo originó.
- **HEventQueue**: Estructura de datos de tipo pila que almacena de forma ordenada en el tiempo los eventos que suceden durante la simulación. Contiene métodos para el tratamiento de la pila: *push*, *pop* y un método para borrar determinados eventos identificados por el *UID*.
- **HTransition:** Representa las transiciones que tienen lugar en las conexiones (*HWire*). Posee métodos para el cálculo del voltaje inicial y final en caso de no ser completas y, para cálculos de cruce con otras que tengan lugar en el mismo intervalo temporal en la misma conexión.

7. Unique IDentificator.

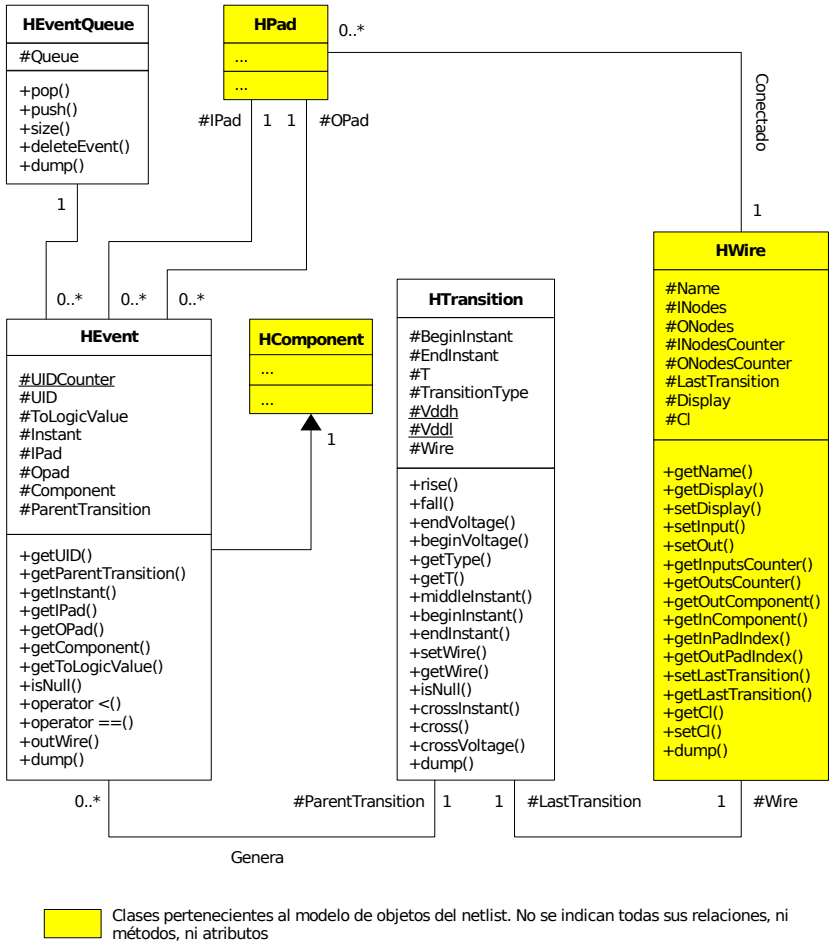


Figura 4.16. Diagrama objetos de la simulación con eventos y transición.

Todo el conjunto de clases presentados a lo largo de esta sección, son utilizadas por el motor de simulación en su conjunto. La interpretación de un diagrama de objetos con todas las clases intervientes en el proceso de simulación sería complicado debido a su magnitud. Por tanto, en la figura 4.17 se representa el modelo de objetos de la simulación centrado en la clase *HSimulation* la cual está relacionada con el *netlist*, la librería de celdas y la cola de eventos, quedando estas tres últimas clases marcadas con fondo de color para indicar que no se desarrollan ya que fueron tratadas en las anteriores secciones.

HSimulation está formada por tres clases para el tratamiento y presentación de los resultados al usuario: resultados de tensión (*HResultV*), resultados de intensidad/potencia (*HResultsC*) y resultados estadísticos (*HResultsS*). Para el cálculo de la curva de intensidad se utiliza una clase que almacenan las aportaciones de intensidad producidas por cada celda *HGraphBlock*. Esta clase provee métodos de cálculo de aportaciones de intensidad basadas en suma de

triángulos (véase el modelo de intensidad en el capítulo 3).

También se indicó que HALOTIS incluye un modo interactivo de funcionamiento y el soporte lo da una librería externa *libreadline* [GNU] enlazada a la clase *HReadLine* que hace de envoltura de esta librería escrita en C y con licencia GPL.

Las clases de la figura 4.17 se detallan a continuación:

- **HSimulation**: Contiene el algoritmo de simulación y está enlazada con todos las estructuras de datos que contienen en memoria el *netlist* y la librería de celdas. Entre los métodos que posee se incluye la lectura de patrones (en formato HALOTIS) y el algoritmo de preprocesado del *netlist*.
- **HResultsV**: Almacena los resultados de las formas de onda de tensión sólo

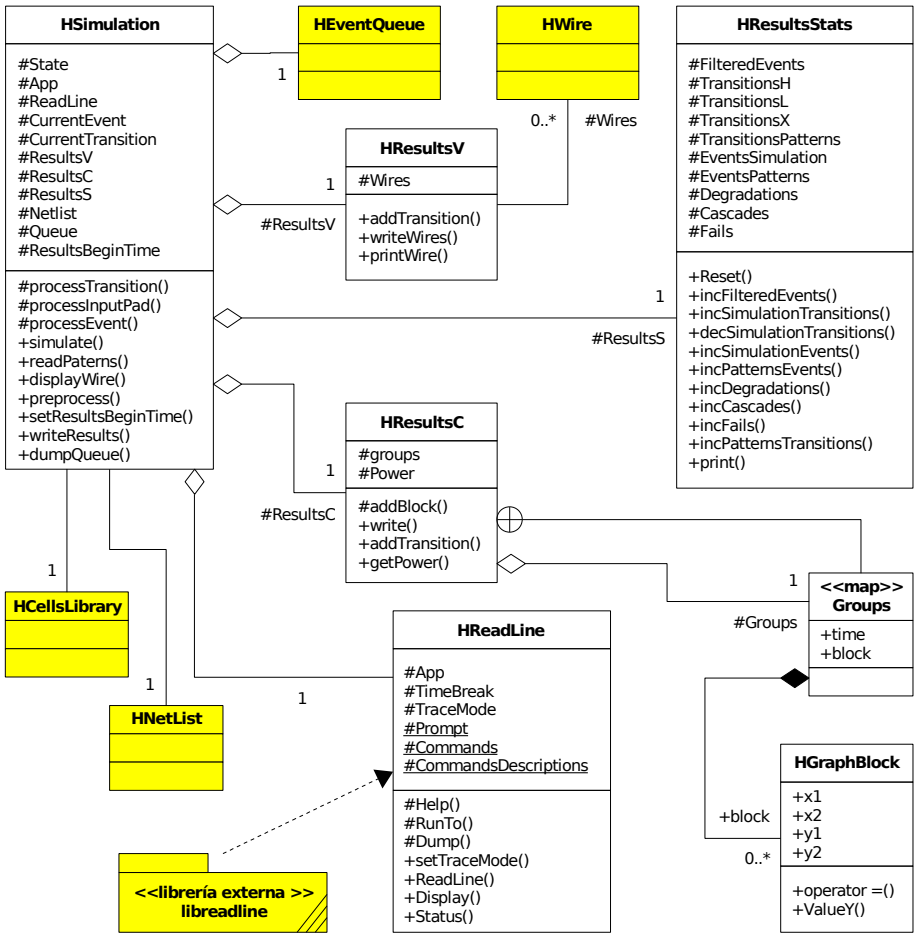


Figura 4.17. Diagrama objetos del motor de simulación

para los nodos que el usuario seleccionó antes de simular. Se relaciona con el conjunto de objetos del tipo *HWire,* de los que guarda la forma de onda. También incluye un método para presentar en formato tabular la forma de onda de tensión (*printWires*).

- **HResultsS**: Esta clase almacena los resultados estadísticos e información sobre el proceso de simulación. Son:
  - Eventos filtrados durante la simulación, debidos al efecto de degradación o el efecto inercial.
  - Transiciones totales, diferenciando tres posibilidades: transiciones ascendentes, descendentes y a valores desconocidos.
  - Transiciones existentes en los patrones de entrada.
  - Eventos generados y procesados durante la simulación.
  - Eventos generados por los patrones de entrada.
  - Degradaciones: número de veces que se aplica la ecuación de degradación por la proximidad de dos transiciones.
  - Cascadas en eliminación de eventos de la cola, debidos a efectos de degradación e inercial.
- **HResultsC**: La curva de intensidad y la potencia total es calculada y almacenada durante la simulación por esta clase. Utiliza la clase *HGraphBlock* que contiene el triángulo de intensidad que aporta cada transición producida en un componente. Una estructura de tipo mapa almacena el conjunto de triángulos que deben ser sumados por superposición para la obtención de la curva global.
- **HReadLine**: Implementa el modo interactivo de HALOTIS utilizando la librería GNU *libreadline*. Contiene el conjunto de comandos y una ayuda resumida de cada uno de ellos.

4.6. Implementación

Esta sección analiza la implementación del entorno HALOTIS donde se trataran dos aspectos: detalles del lenguaje de programación y el empaquetamiento de los componentes.

En primer lugar se analizarán los detalles de implementación referentes al lenguaje de programación y las herramientas y librerías externas usadas. Seguidamente, se tratará el empaquetamiento el cual hace referencia a la ubica-

ción de las clases en cada componente final de HALOTIS (librería o ejecutable). En esta descripción volverán a aparecer las clases mostradas a lo largo del modelado (sección anterior) y algunas nuevas que sólo forman parte del componente en cuestión descrito.

4.6.1. Lenguaje de programación

Dada la metodología seguida, el lenguaje de programación utilizado también es un lenguaje de programación orientado a objetos. La obtención del código a partir de un modelo de objetos a un lenguaje de este tipo es directa. Actualmente, los lenguajes de programación orientados a objetos con mayor aceptación son JAVA, C++, PYTHON para los cuales existen gran cantidad de herramientas CAD de apoyo a la programación. De hecho, todas las figuras mostradas en el apartado anterior, han sido obtenidas con dos de estas herramientas: *ArgoUML* [ARGO06] y *Umbrello* [HENS03].

Finalmente la implementación se ha realizado en lenguaje C++, y se enumeran los motivos que han llevado a tomar esta decisión:

- C++ ofrece mayor flexibilidad frente a la robustez y el fuerte tipado de JAVA.
- JAVA funciona sobre una máquina virtual y esto influye en la velocidad de ejecución del código, pudiéndose considerar C++ cómo más rápido al usar código nativo de cada máquina sobre la que se ejecuta la aplicación.
- PYTHON es un lenguaje interpretado, por tanto, el entorno de explotación necesita tener el intérprete instalado, mientras que para C++ existen compiladores para todas las plataformas y se puede distribuir el software sin dependencia del interprete.
- Es posible combinar código C++ con código C y en la actualidad existen gran cantidad de librerías de utilidades que generan de forma automatizada código C, capaz de resolver ciertas tareas extremadamente complejas, cómo son los análisis léxicos, sintácticos y semánticos de ficheros en entrada, utilizados para la interpretación de ciertos lenguajes (VERILOG, XML, MACHTA, VHDL)

Otro aspecto importante de la implementación es la programación de los tipos de datos comunes a todo proyecto software: listas, mapas, colas, vectores, etc. En C++ existe una librería estándar para este fin, denominada STL (*Standart Template Library*) [SCHI99]. Proporciona tipos de datos usados frecuentemente en todo software como son listas, colas, árboles, vectores, algoritmos, etc. Esta librería se incluye como parte de todo compilador de C++, siendo así posible la compilación

de cualquier programa que las use en diferentes plataformas con soporte C++. En la mayoría de las relaciones con multiplicidad mayor de uno, se han utilizado colecciones basadas en STL para su implementación, aplicándose a todos los diagramas de objetos (figuras desde 4.12 a 4.17).

La lectura y procesamiento de lenguajes HDL se implementa mediante el uso de las herramientas y metodologías existentes para generación de compiladores que procesan gramáticas. Tradicionalmente estas herramientas son LEX como analizador léxico y YACC como analizador sintáctico. La evolución de estas herramientas en el ámbito de la programación orientada a objetos y el software libre culminan hoy en día en FLEX++ y BISON++. Ámbas generan código C++ y son los equivalentes a FLEX y YACC respectivamente.

También, en la implementación de la herramienta *hmachta* se ha utilizado la herramienta FLEX++. La sintaxis de este tipo de patrones no es de gran complejidad, por lo que ha bastado con un analizador léxico.

4.6.2. Empaquetamiento de los componentes

Todo el software del entorno de simulación HALOTIS se agrupa en diferentes componentes de los que sólo se distinguen dos tipos: ejecutables y librerías. La figura 4.18 incluye todas las clases descritas a lo largo de la sección 4.5 (Modelado de HALOTIS) ubicadas en su correspondiente paquete. Todos estos componentes se generan tras la compilación completa del sistema. Las dependencias externas se marcan con <<*externa*>> en el diagrama. Un resumen de la figura 4.18 se hace en la tabla 4.2 en la que se enumeran de nuevo todas las clases y su ubicación.

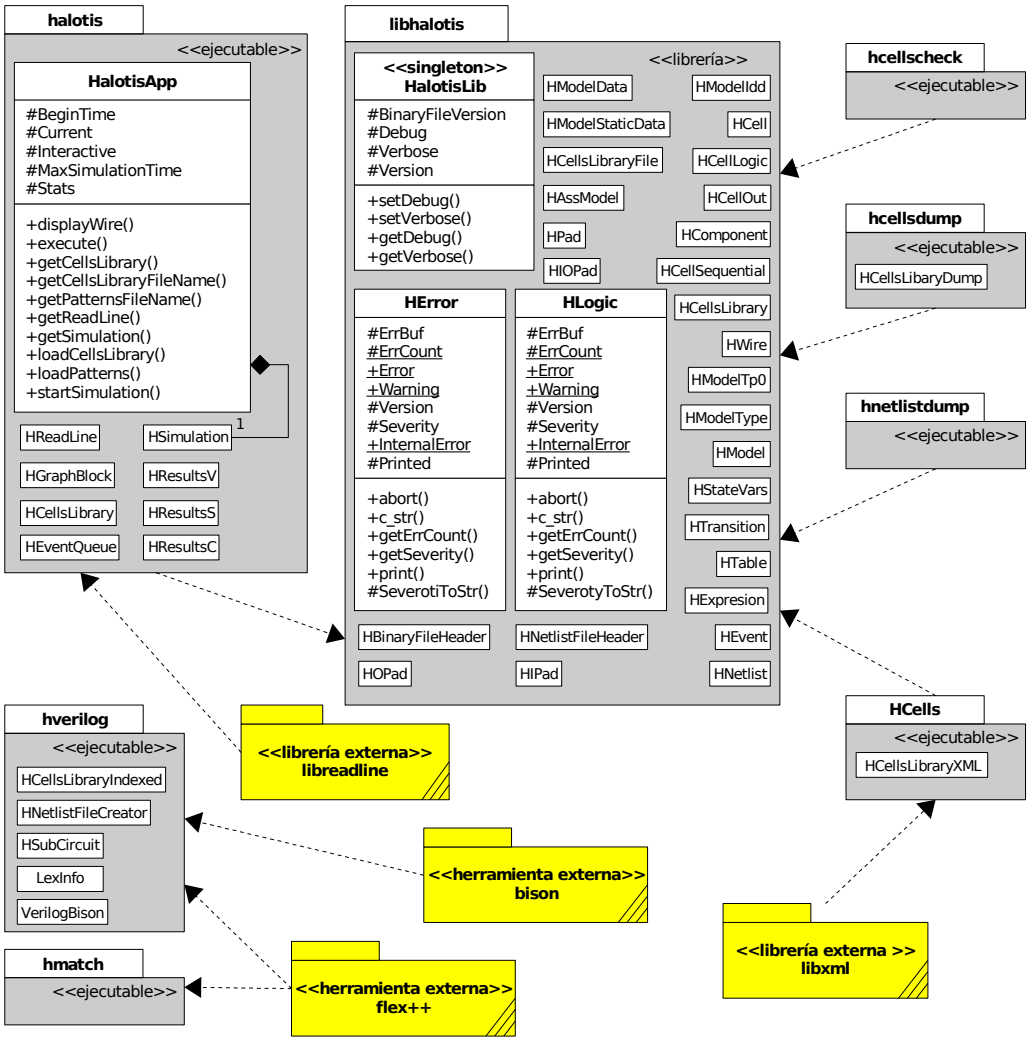


Figura 4.18. Organización de los componentes de HALOTIS.

Objetivo	Clases	Tipo
libhalotis	Halotislib, HAssModel, HBinaryFileHeader, HCell, HCellLogic, HCellOut, HCellSequential, HCellsLibrary, HCellsLibraryFile, HComponent, HError, HEvent, HExpresion, HPad, HIPad, HOpad, HIOPad, HLogic, HModel, HModelData, HModelIdd, HModelStaticData, HModelTp0, HModelType, HNetlist, HNetlistFileHeader, HStateVars, HTable, HWire	librería
hcells	HCellsLibraryXML	ejecutable
hverilog	HCellsLibraryIndexed, HNetlistFileCreator, HSubCircuit, Lexinfo, VerilogBison	ejecutable
hmachta	Parser, LexInfo	ejecutable
hcellscheck		ejecutable
hcellsdump	HCellLibraryDump	ejecutable
hnetlistdump		ejecutable
halotis	HalotisApp, HalotisCommandLine, HEventQueue, HGraphBlock, HReadLine, HResultsC, HResultsStats, HResultsV, HSimulation	ejecutable

Tabla 4.2. Ubicación de las clases en cada componente de HALOTIS.

4.6.3. Resumen del resultado de la implementación

El resultado final de la implementación de HALOTIS es un código C++ que contiene 13409 líneas de código, distribuidas en 117 ficheros fuente y con un total de 392Kbytes. Debido a esta enorme extensión y a que la información que aportaría no es muy significativa se ha optado por no incluirlo en esta documentación. No obstante, sí se incluye a continuación la tabla 4.3 para dar una idea de la magnitud de cada uno de los componentes medidos en número de ficheros, líneas de código y tamaño.

Objetivo	Clases	Ficheros	LCD	Tamaño
libhalotis	29	64	6720	196KBytes
hcells	1	8	724	24KBytes
hverilog	4	15	1722	49KBytes
hmachta	2	4	650	17KBytes
hcellscheck	0	1	137	5KBytes
hcellsdump	1	3	835	27KBytes
hnetlistdump	0	1	126	2Kbytes
halotis	9	21	2495	72KBytes
Total:	46	117	13409	392KBytes

Tabla 4.3. Datos sobre el tamaño de cada componente de HALOTIS.

4.7. QHALOTIS

El capítulo termina con un breve recorrido a la herramienta QHALOTIS que es una interfaz gráfica para HALOTIS, desarrollada para los sistemas de escritorio KDE/QT [SWEET01] y compatible con el escritorio GNOME.

El principal objetivo ha sido desarrollar componentes software de representación gráfica reutilizables. Se ha desarrollado por tanto un visor de ondas capaz de interpretar la salida de HALOTIS, y así, poder representarla en pantalla o exportarla a diferentes formatos gráficos. Este visor de ondas puede ser incrustado en cualquier aplicación KDE/QT mediante la técnica KPARTS [SWEET01] que incorpora este escritorio.

La aplicación QHALOTIS tiene incrustado este componente en su área de programa y permite la representación de ondas en un entorno gráfico, siendo estas capturadas directamente de la salida de simulación de HALOTIS, para ello, es capaz de controlar el entorno HALOTIS (compilación de celdas, compilación de *netlist* y simulación). Además, este software soporta la importación de datos externos, como por ejemplo, los generados por el simulador eléctrico SPICE/HSPICE para una comparación visual en su visor de ondas.

QHALOTIS establece la base para el futuro desarrollo de una interfaz gráfica con más posibilidades en el entorno de simulación HALOTIS. La aplicación QHALOTIS se diseñó con las siguientes especificaciones como guía:

- Poder abrir/añadir gráficas en distintas ventanas o en una misma ventana (esto último facilitaría la comparación de gráficas). La aplicación por tanto es una de aplicación MDI (*Multiple Document Interface*), que son aquellas que pueden gestionar varios documentos a la vez.
- Ser capaz de imprimir dichas representaciones utilizando el sistema de impresión del escritorio.
- Exportar las representaciones en diversos formatos gráficos como son PNG, JPG, etc.
- Poder guardar las representaciones también en formato *PostScript*.
- Facilitar la configuración del programa al usuario final. Para ello, se han creado dos menús de configuración que permiten especificar opciones para una correcta comunicación entre en entorno HALOTIS y QHALOTIS. Por otro lado, la configuración se guarda en un fichero de texto plano permitiendo una simple edición como alternativa para modificar los parámetros del programa.
- Realizar una buena gestión del visor de ondas, incluyendo la gestión de

zooms (acercar/alejar), desplazamientos y selección de diferentes señales en el área de dibujo.

La figura 4.19 muestra una captura de pantalla de QHALOTIS tras una simulación con HALOTIS.

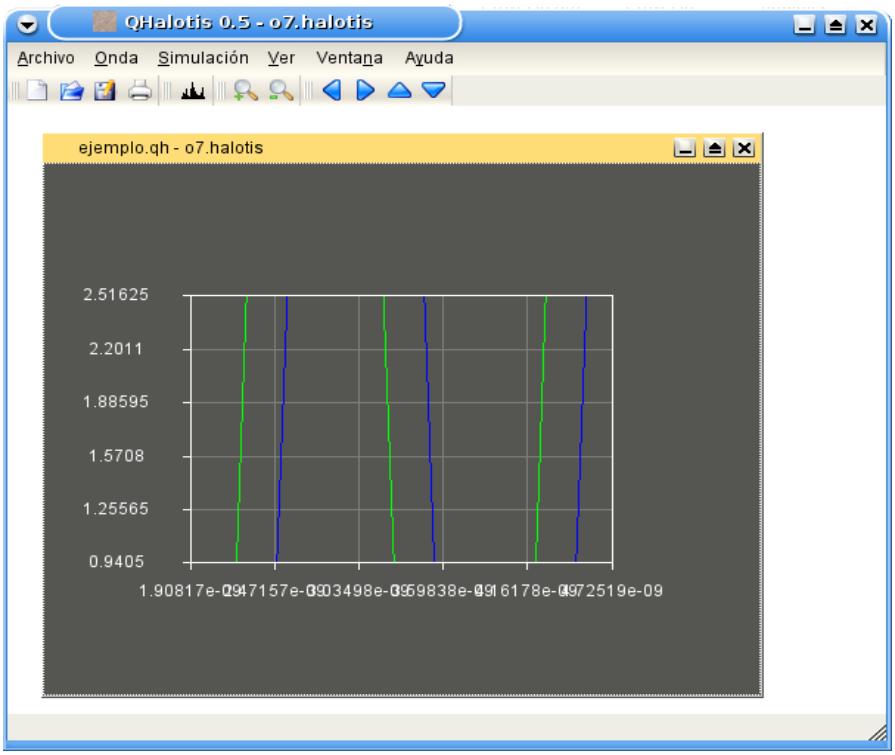


Figura 4.19. Visión general de QHALOTIS.

Para completar esta herramienta habría que desarrollar las partes que interactúan con el resto de componentes de HALOTIS. Basar el desarrollo en la técnica KPARTS, como se ha hecho con el visor de ondas, permitiría ir incrustando en QHALOTIS todos los módulos necesarios para controlar el sistema HALOTIS, que serían:

- **Editor y compilador de librerías de celdas:** encargado de gestionar la librería de celdas con la que se realizará cada simulación. Esta gestión debería contemplar la edición todos los parámetros necesarios en cada celda, la compilación y gestión completa de errores y, chequeos de consistencia.
- **Editor y visor de patrones:** módulo el para tratamiento de patrones tanto en modo texto como en modo visual.
- **Gestión del modo interactivo de HALOTIS:** módulo para integrar el modo interactivo del motor de simulación, facilitando al usuario la visua-

lización de la evolución temporal de la simulación y la depuración, tanto del circuito, como de la librería de celdas.

Capítulo 5. Resultados y análisis de rendimiento de HALOTIS

En este capítulo se han agrupado los diferentes resultados tanto de simulación como de aplicaciones y medida del rendimiento sobre el simulador HALOTIS.

Así, el capítulo queda organizado en tres secciones principales. La primera está dedicada a mostrar dos aspectos fundamentales de HALOTIS: el buen funcionamiento desde la perspectiva de los resultados temporales obtenidos en la simulación de circuitos digitales y la precisión alcanzada en los mismos [RUIZ01a, JUAN01a]. La segunda sección se centra en las capacidades incorporadas en el simulador para su aplicación en tareas de estimación del consumo de potencia [BAENA02, RUIZ05a, RUIZ06, BELL06]. Por último, la tercera sección está dedicada a análisis del rendimiento de HALOTIS desde una perspectiva

informática. Así, se somete a HALOTIS a cargas grandes de trabajo con las que se evalúan los tiempos de ejecución y uso de recursos informáticos en circuitos de alta complejidad.

5.1. Análisis de los resultados lógicos-temporales de HALOTIS

En este primer conjunto de resultados se presentan una serie de simulaciones de circuitos sencillos pero que sirven para ilustrar tanto el correcto funcionamiento de la herramienta HALOTIS como la gran precisión que adquiere la simulación lógica-temporal cuando emplea el modelo de retraso DDM, sobre todo, en la propagación tanto de pulsos espurios o *glitches* como en la operación de alta frecuencia de un circuito.

Para realizar el análisis de los resultados de HALOTIS, se comparan con los obtenidos directamente con el simulador eléctrico HSPICE. Por otra parte, para verificar la precisión del modelo DDM se presentan simulaciones obtenidas con HALOTIS empleando tanto el modelo DDM (HALOTIS-DDM) como el modelo CDM (*Conventional Delay Model*), el cual, como ya se mencionó en la capítulo 2, no incluye el efecto de degradación.

Concretamente, los ejemplos que se presentan son los siguientes:

- Circuito formado por una cadena de inversores CMOS donde se contemplan dos tipos de estímulos:
  - Propagación de un único pulso con la finalidad de analizar la propagación de *glitches*.
  - Propagación de un tren rápido de pulsos para analizar el circuito sometido a alta frecuencia de operación.
- Circuito formado por una cadena de inversores CBL[8]
  - Propagación de un tren rápido de pulsos donde se analiza la validez de DDM en celdas CBL.
- Oscilador en anillo formado por un número impar de inversores.
  - Medida de la frecuencia de oscilación para analizar la alta precisión del modelo DDM.

8. Current Balanced Logic

5.1.1. Cadena de inversores CMOS

El circuito combinacional más interesante para analizar la propagación de *glitches* y la operación a altas frecuencias es una cadena de varios niveles de puertas. Para realizar esta tarea se ha seleccionado el circuito de la figura 5.1, una cadena de inversores de 8 niveles. Las simulaciones se han realizado utilizando *kit* tecnológico CMOS de 0.6μm de AMS y los parámetros para HALOTIS son los que aparecen en el anexo 1 para dicho *kit*. Además, en el ejemplo 5.1 se muestra el *netlist* en formato VERILOG para HALOTIS.

Para realizar comparaciones entre los resultados obtenidos con HALOTIS y HSPICE se utiliza el criterio de paso por el 50% de V_{DD} para medir el ancho de los pulsos.

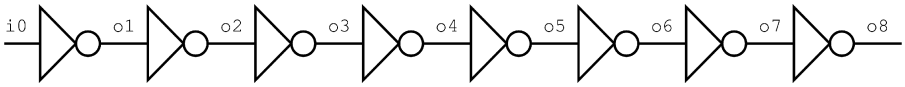


Figura 5.1. Cadena de ocho inversores.

Ejemplo 5.1. Descripción en formato VERILOG de la cadena de inversores.

```
module inversor (i0,o8);
     output o8;
     input i0;

     inv inv0 (in,o1);
     inv inv1 (o1,o2);
     inv inv2 (o2,o3);
     inv inv3 (o3,o4);
     inv inv4 (o4,o5);
     inv inv5 (o5,o6);
     inv inv6 (o6,o7);
     inv inv7 (o7,o8);
endmodule
```

5.1.1.1. Propagación de un único pulso

El primer caso presentado es la simulación de un único pulso en la entrada de la cadena.

En las figuras 5.2, 5.3 y 5.4 se muestran los resultados de propagación del mismo pulso empleando HPSICE, HALOTIS-CDM y HALOTIS DDM respectivamente. En el resultado de HSPICE se observa como el pulso es degradado (reduciendo su tamaño) a medida que se propaga a través de las puertas e incluso

en la salida del quinto inversor se considera que ha sido completamente eliminado (efecto inercial).

Respecto a los resultados obtenidos con HALOTIS, en primer lugar hay que destacar que, efectivamente, el pulso se propaga a través de las puertas en ambos casos, lo cual indica el correcto funcionamiento del motor de simulación implementado. Ahora bien, los resultados son muy diferentes en el caso del modelo CMD y DDM. Efectivamente, con el modelo CDM no se considera el efecto de degradación por lo que el pulso se propaga a través de todos los niveles de la cadena de inversores sin reducción del tamaño, llegando a obtenerse un resultado incorrecto a partir de la salida del quinto inversor al compararlo con HSPICE. En cambio, el modelo DDM da lugar al mismo efecto observado con HSPICE donde hay una reducción progresiva del tamaño del pulso y posterior eliminación a partir de la salida del quinto inversor. Este resultado muestra la eficiencia del modelo DDM en la propagación de pulsos espurios o *glitches*.

En la tabla 5.1 aparecen medidas cuantitativas sobre los tamaños del pulso en los diferentes salidas de las puertas para las tres simulaciones. Se observa en estos datos el comportamiento explicado anteriormente. El modelo DDM reduce muy significativamente el error introducido en el caso que no se considere la degradación (modelo CDM)

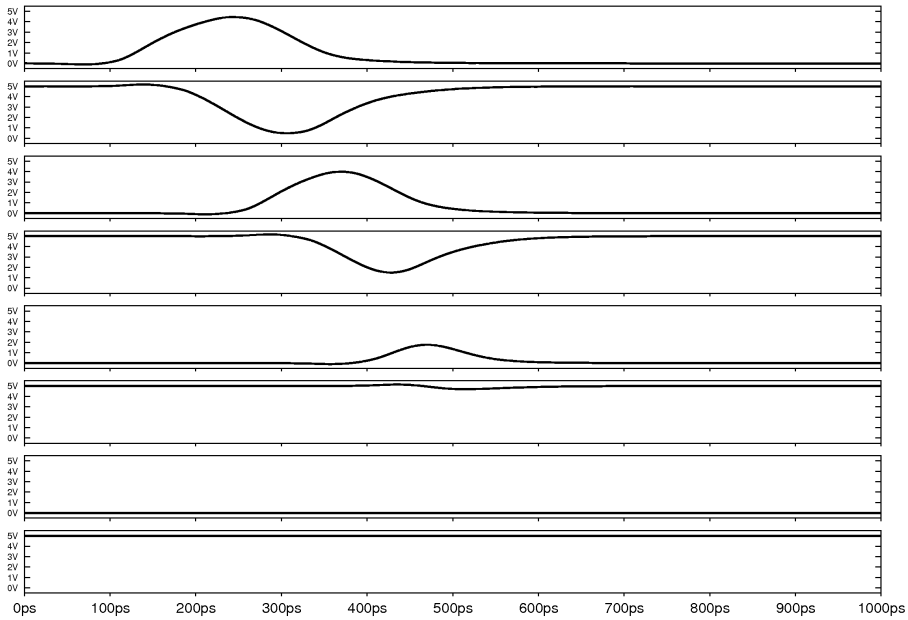


Figura 5.2. Pulso en la cadena de inversores simulado con HSPICE.

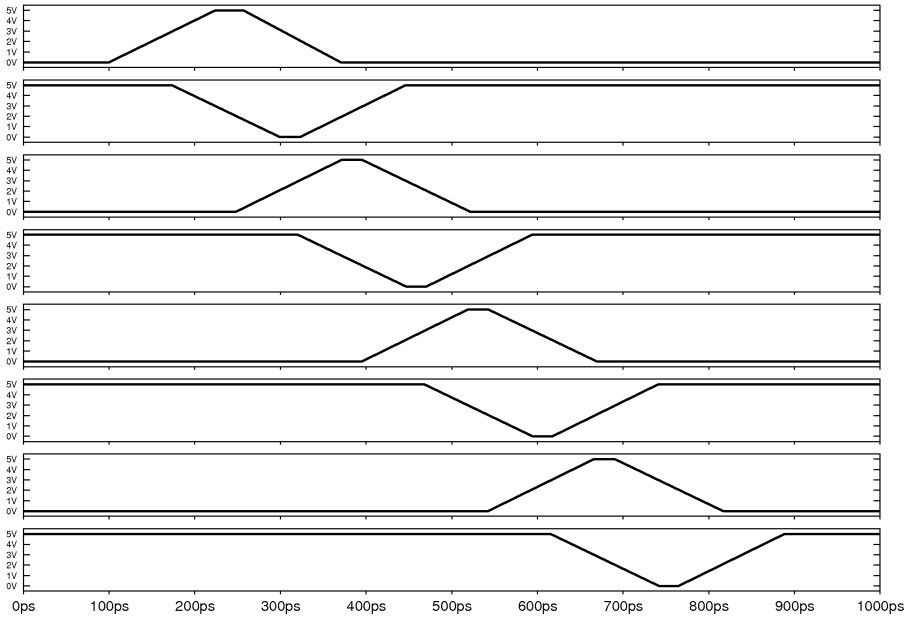


Figura 5.3. Pulso en la cadena de inversores simulado con HALOTIS-CDM.

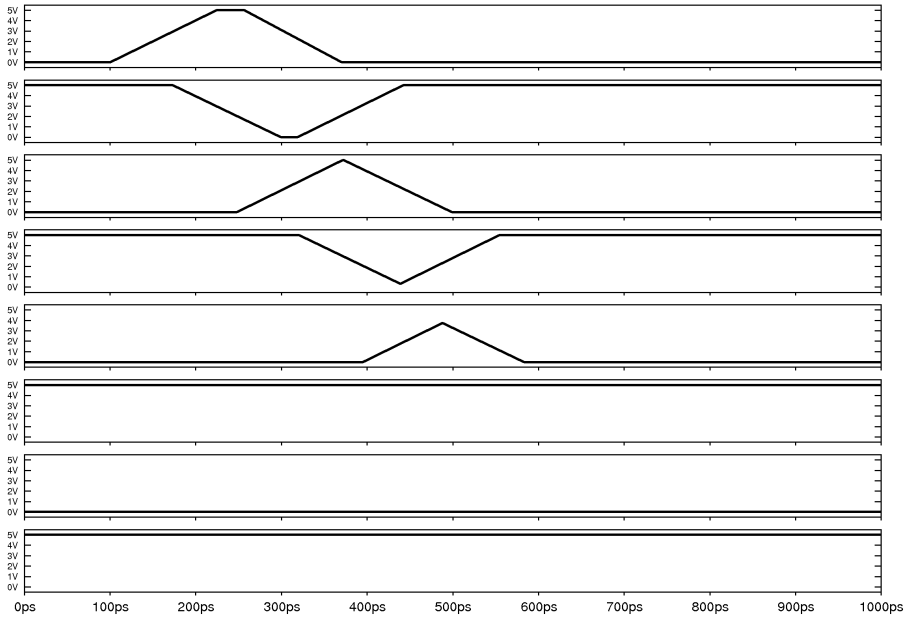


Figura 5.4. Pulso en la cadena de inversores simulado con HALOTIS-DDM.

Nodo	HSPICE	DDM		CDM	
		Anchura	Error	Anchura	Error
In	151,5ps	151,5ps	0%	151,5ps	0%
O_1	137,4ps	144,07ps	4,9%	148,2	7,9%
O_2	116,1ps	126,5ps	8,6%	149,3	28%
O_3	82,78ps	108,8ps	32%	148,2	79%
O_4	-	62,82ps	-	149,3	-
O_5	-	-	-	148,2	-
O_6	-	-	-	149,3	-
O_7	-	-	-	148,2	-

Tabla 5.1. Comparación de anchuras de pulso en la cadena de inversores.

5.1.1.2. Propagación de un tren de pulsos rápidos

El segundo caso presentado es la propagación de un tren de pulsos rápidos. Con estas simulaciones se ilustra cómo al operar a alta frecuencia, para obtener resultados fiables en la simulación lógica es necesario considerar el efecto de degradación, pues en caso contrario, los resultados pueden estar muy alejados de la realidad. Si bien se han llevado a cabo simulaciones de varios trenes de pulsos (en cuanto a la separación entre pulsos) en esta sección se muestran dos ejemplos. Así, en la figuras 5.5, 5.6 y 5.7 presentan los resultados del primer ejemplo, una entrada en forma de tren de pulsos equiespaciados, para HSPICE, HALOTIS-CDM y HALOTIS-DDM respectivamente.

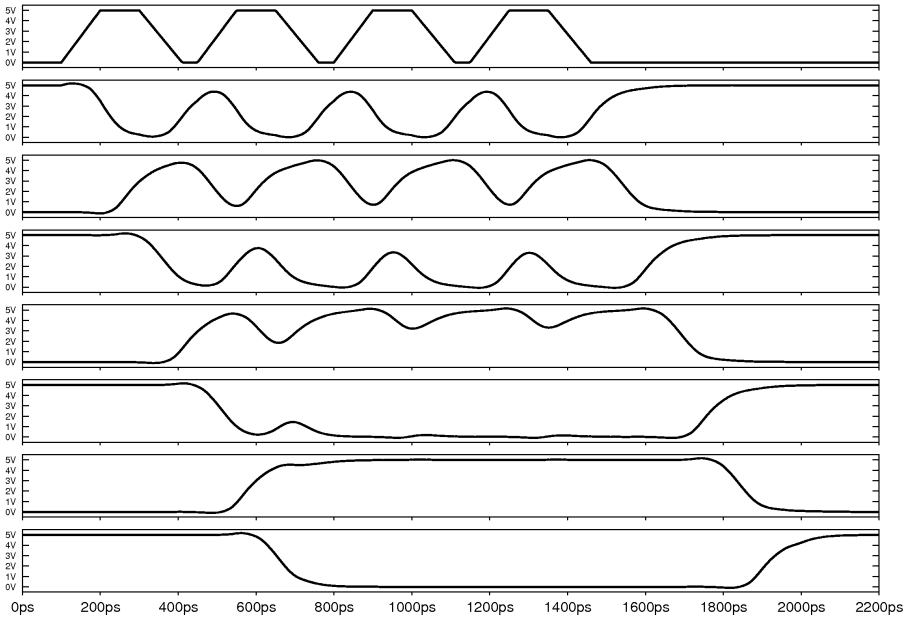


Figura 5.5. Tren en la cadena de inversores simulado con HSPICE.

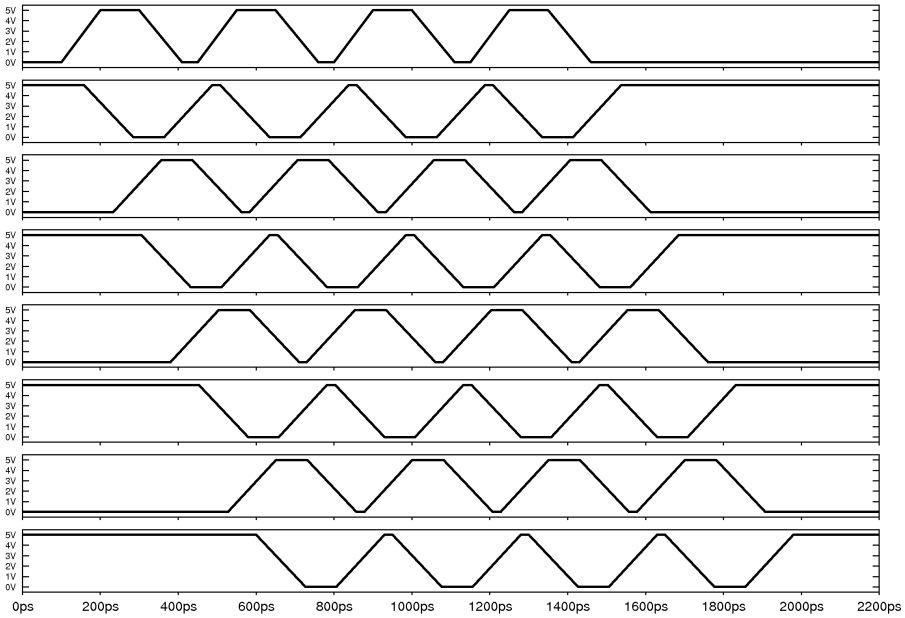


Figura 5.6. Tren en la cadena de inversores simulado con HALOTIS-CDM.

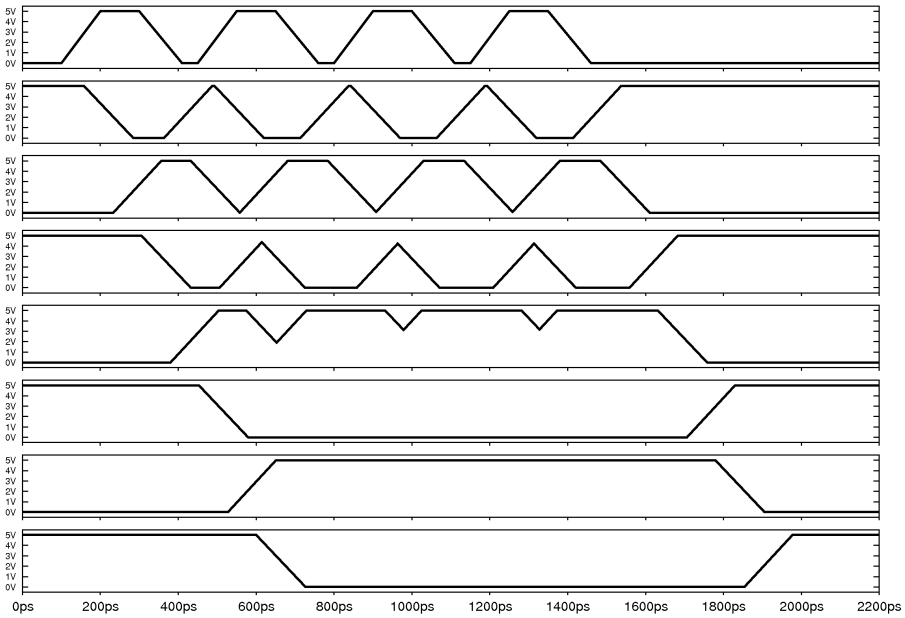


Figura 5.7. Tren en la cadena de inversores simulado con HALOTIS-DDM.

En la simulación considerada resultado experimental (HSPICE), es interesante observar que en la salida del primer inversor los pulsos en nivel bajo sufren una mayor degradación que los pulsos en nivel alto provocando en las sucesivas salidas el efecto de reducción del tamaño de pulsos sea más fuerte, llegando hasta el filtrado final en la salida del quinto inversor. Tras este filtrado el resultado final es un único pulso de gran tamaño no sujeto a degradación.

Nuevamente, en los resultados con HALOTIS se manifiesta el mismo hecho comentado en la sección anterior, con HALOTIS-CDM, sin degradación, el tren de pulsos acaba propagándose en su totalidad a través de toda la cadena de inversores. En cambio, con HALOTIS-DDM se observa el mismo efecto que con HSPICE, disminución fuerte de los pulsos en nivel bajo al propagarse por el primer inversor y, a partir de este nivel, mayor reducción y posterior eliminación de estos pulsos. Es de destacar como las formas de ondas de la salida cuarta y quinta son enormemente coincidentes entre HSPICE y HALOTIS-DDM. De esta forma se concluye que la no inclusión de la degradación provoca un resultado de simulación lógica muy diferente del real tal y como se comprueba en la tabla 5.2 donde se muestra la secuencia de valores lógicos en la salida de la cadena para las diferentes simulaciones.

Resultados a nivel lógico	
Simulación con HSPICE	101
Simulación con HALOTIS-DDM	101
Simulación HALOTIS-CDM	101010101

Tabla 5.2. Comparación a nivel lógico de la salida de la cadena de inversores.

En el segundo ejemplo se simula otro tren de pulsos con el cual se persigue aumentar la complejidad en la forma de onda de la salida.

En las figuras 5.8, 5.9 y 5.10 se muestran los resultados según HSPICE, HALOTIS-CDM y HALOTIS-DDM. Para el patrón de entrada elegido se observa en el resultado de HSPICE como el pulso de mayor tamaño en nivel bajo se degrada mucho menos que los otros dos pulsos pequeños en nivel bajo, de manera que, en la salida del tercer inversor los dos pulsos pequeños en nivel bajo han sido filtrados, quedando finalmente un único pulso intermedio. El mismo efecto se observa en la simulación de HALOTIS-DDM y, de nuevo, el resultado con CDM es claramente erróneo (tabla 5.3).

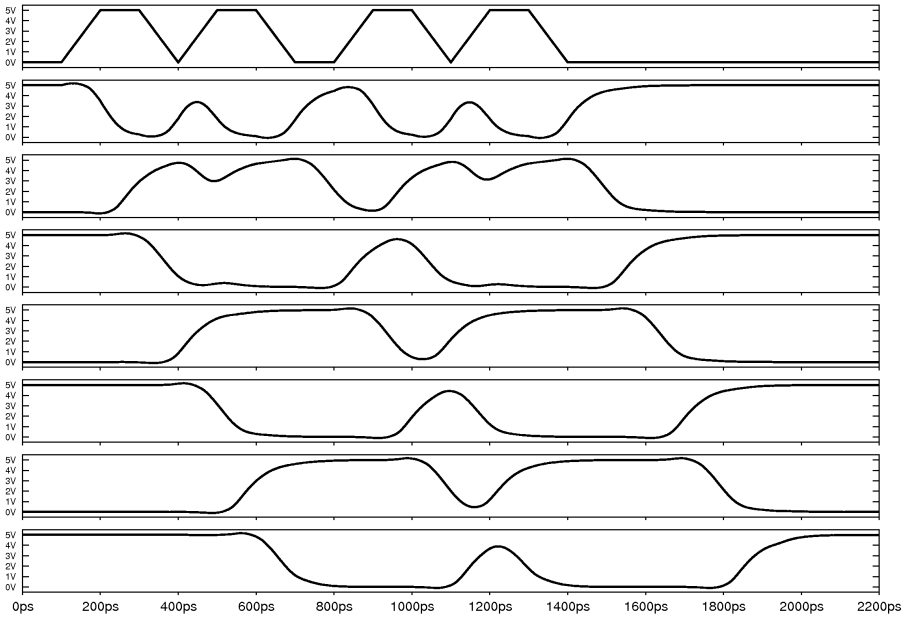


Figura 5.8. Segundo tren de pulsos con degradación simulado con HSPICE.

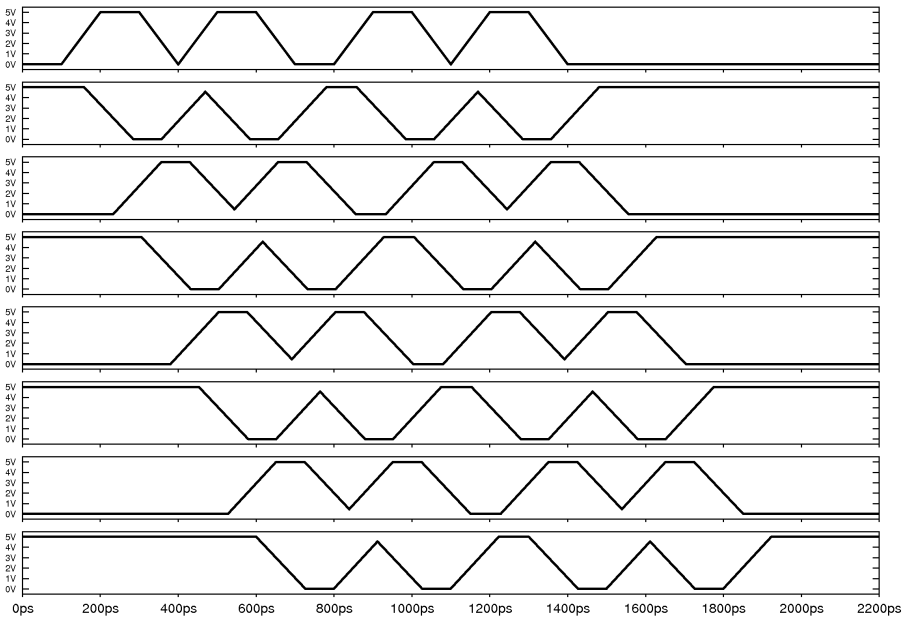


Figura 5.9. Segundo tren de pulsos con degradación simulado con HALOTIS-CDM.

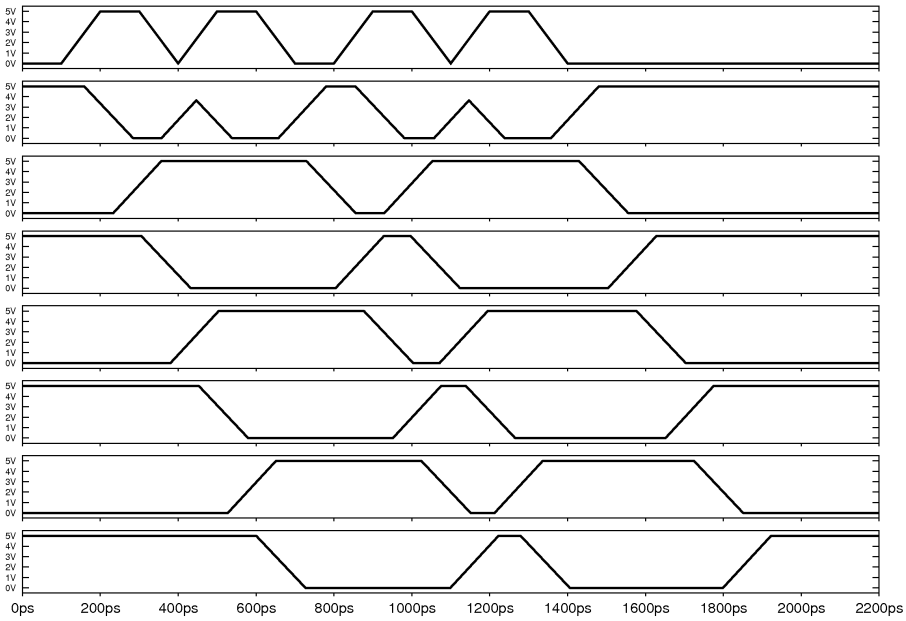


Figura 5.10. Segundo tren de pulsos con degradación simulado con HALOTIS-DDM.

Resultados a nivel lógico	
Simulación con HSPICE	10101
Simulación con HALOTIS-DDM	10101
Simulación HALOTIS-CDM	101010101

Tabla 5.3. Comparación a nivel lógico de la salida de la cadena de inversores.

5.1.2. Cadena de inversores CBL

En el capítulo 2 sección 2.5 se validó el modelo DDM para las celdas CBL [ALBU99a]. Para mostrar la precisión de DDM cuando se simulan circuitos diseñados con estas celdas, se ha llevado a cabo la simulación de una cadena de inversores CBL [RUIZ03].

En las figuras 5.11 y 5.12 se muestran los resultados de la propagación de un tren de pulsos rápido según HSPICE y HALOTIS-DDM respectivamente. El comportamiento es similar al comentado anteriormente para la cadena de inversores CMOS. Esto es, según HSPICE, a medida que el tren de pulsos se propaga por la cadena los pulsos en nivel bajo (en la señal de entrada) son degradados para finalmente ser eliminados. El mismo resultado se observa en la simulación de HALOTIS-DDM.

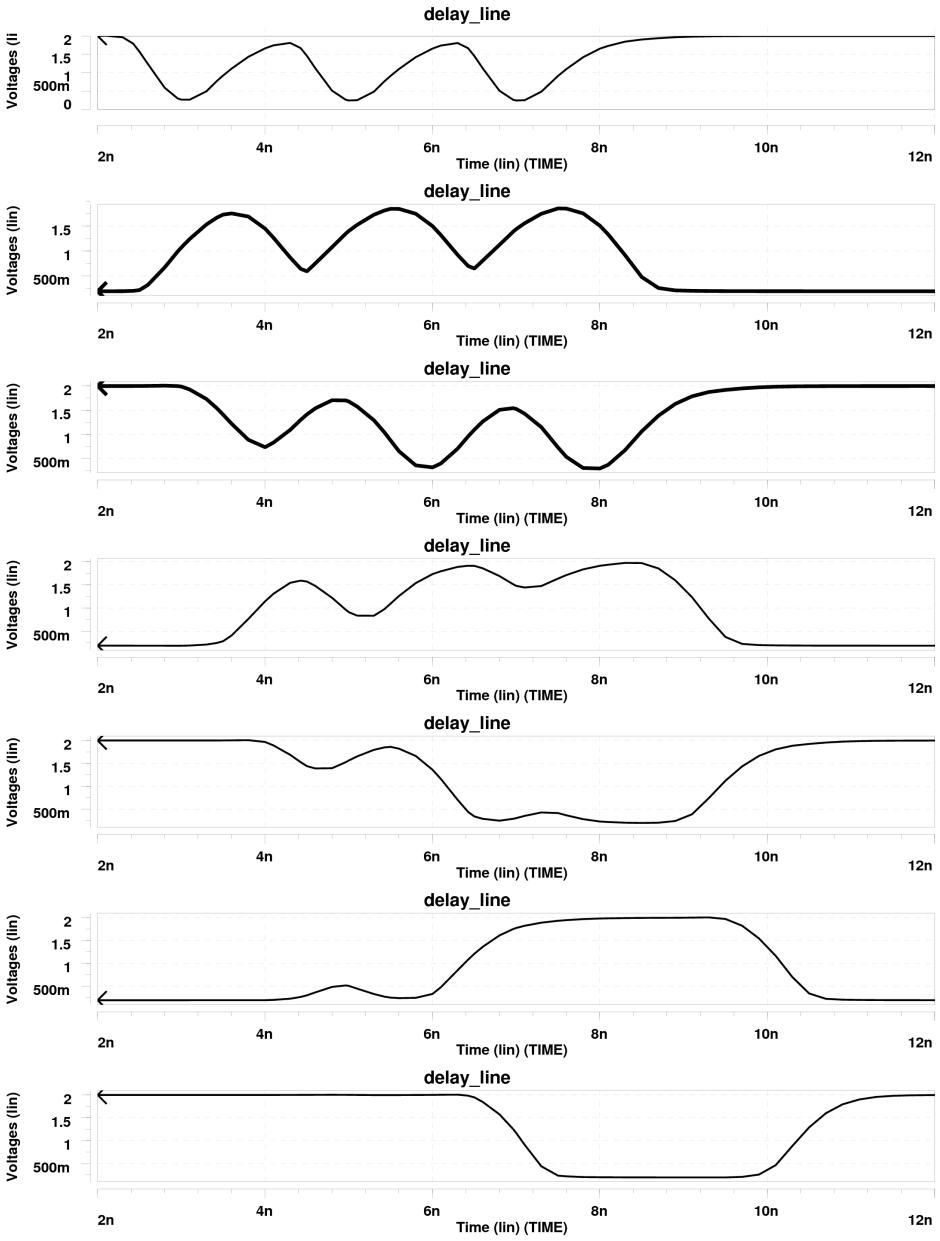


Figura 5.11. Simulación HSPICE de la cadena de inversores CBL.

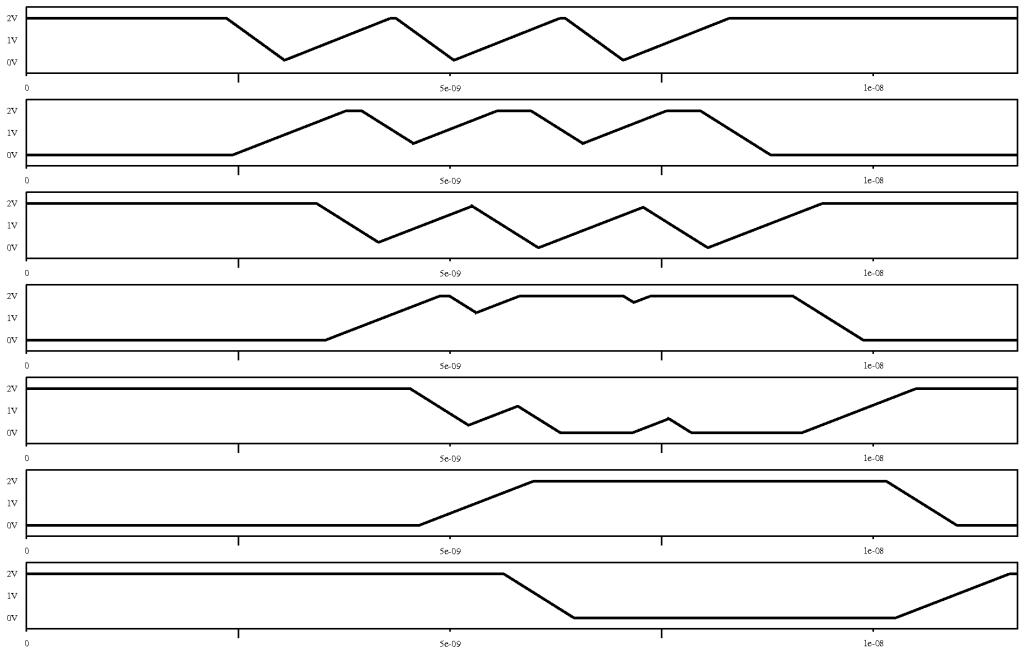


Figura 5.12. Simulación con HALOTIS de la cadena de inversores CBL.

5.1.3. Oscilador en anillo

El segundo circuito estudiado es un oscilador en anillo construido con tres inversores cuyo esquemático se muestra en la figura 5.13 y, el *netlist,* en el ejemplo 5.2. Esta simulación ha sido limitada en el tiempo, puesto que al estar realimentado, la simulación no converge y nunca se vacía la cola de eventos del simulador. El final de la simulación en un simulador guiado por eventos está marcada por la cola de eventos. Cuando esta se vacía, la simulación termina, pero en ciertos circuitos con realimentación, la cola de eventos nunca llega a vaciarse, teniendo en estos casos, que limitar la simulación a una ventana temporal. Este circuito es una de estas situaciones.

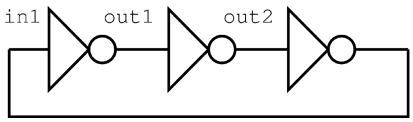


Figura 5.13. Circuito oscilador.

Ejemplo 5.2. Descripción en formato VERILOG del oscilador.

```
module oscilador(in1);
  input in1;
  output out1;
  inv inv1 (in1,out1);
  inv inv2 (out1,out2);
  inv inv3 (out2,in1);
end module
```

La excitación inicial de este circuito es una única transición en algún nodo interno. Tras esta primera transición comienzan las oscilaciones y los resultados de simulación con HSPICE, HALOTIS-CDM y HALOTIS-DDM se han superpuesto en la figura 5.14. En la figura se observa que el error acumulado tras varias oscilaciones hace que las oscilaciones ya no se superpongan visualmente. Las oscilaciones de alta frecuencia están sujetas a cierto factor de degradación obteniéndose mejores resultados en la simulación con degradación. Midiendo el período de oscilación se obtiene una estimación cuantitativa del error cometido en el proceso de simulación. Las medidas del período están resumidas en la tabla 5.4 incluyendo los errores cometidos por los diferentes modelos respecto a HSPICE.

Este efecto se pronuncia al añadir al circuito oscilador un *buffer* de salida (figura 5.15). Este *buffer* influye en las oscilaciones ya que aporta una carga adicional C_B en el nodo in_1. Tras repetir las simulaciones con este nuevo circuito,

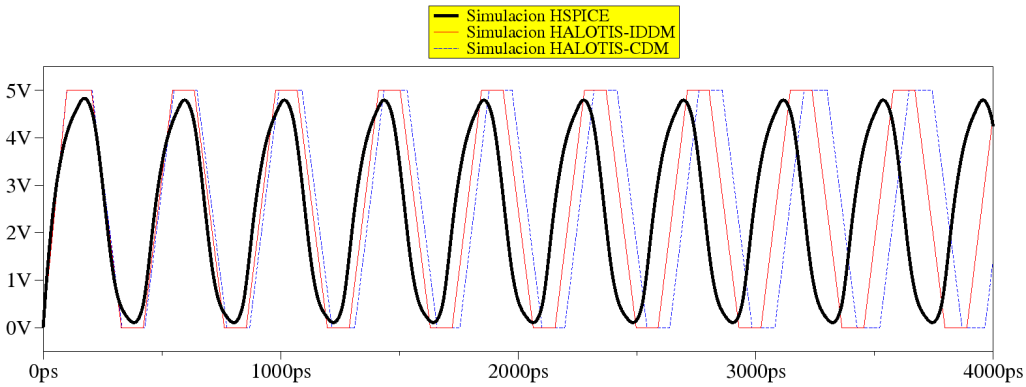


Figura 5.14. Formas de onda del oscilador en anillo.

	HSPICE	HALOTIS-CDM	HALOTIS-DDM
Período (ps)	402,2	428,9	419,8
Frecuencia (Ghz)	2,380	2,332	3,382
Error respecto HSPICE	0%	6,64%	4,37%

Tabla 5.4. Comparación del período oscilación del oscilador.

se obtiene la figura 5.16, observándose que, con HSPICE, la forma de onda no llega en los picos a 0V y 5V, agudizándose así el efecto de degradación en las oscilaciones. La comparación de los períodos de oscilación se ha calculado de nuevo en la tabla 5.5, y se concluye con estos resultados que la diferencia entre el uso de un modelo de sin degradación puede llegar a aumentar el error más del doble .

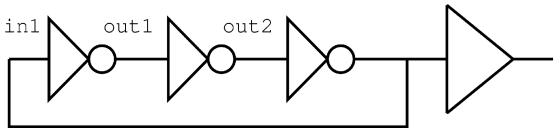


Figura 5.15. Oscilador en anillo con buffer.

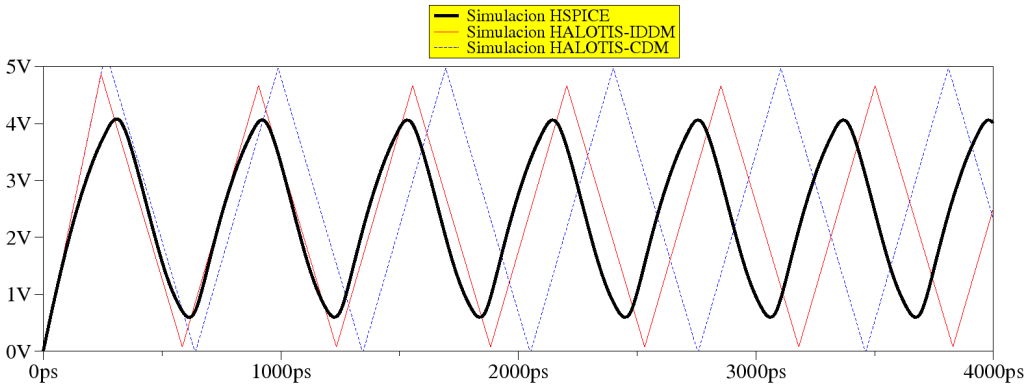


Figura 5.16. Formas de onda del oscilador en anillo con buffer.

	HSPICE	HALOTIS-CDM	HALOTIS-DDM
Período (ps)	611,7	705,5	648,8
Frecuencia (Ghz)	1,635	1,417	1,541
Error respecto HSPICE	0%	15,3%	6,1%

Tabla 5.5. Comparación del período de oscilación del oscilador con buffer.

5.1.4. Multiplicador combinacional de 4 bits

Para completar este apartado se presentan los resultados de simulación del circuito multiplicador de 4 bits mostrado en la figura 5.17. El objetivo de estas nuevas simulaciones es aumentar la complejidad del *netlist*, aumentando el número de puertas y el número de conexiones. El multiplicador está formado internamente por sumadores completos (bloques F.A.), cada uno de ellos con una estructura a nivel de puertas mostrado en la figura 5.18. En total, el multiplicador consta de 124 puertas, además, los bloques F.A. se han incluido en el *netlist* como subcircuitos (ejemplo 5.3), para comprobar también el correcto funcionamiento del compilador de VERILOG (*hverilog*).

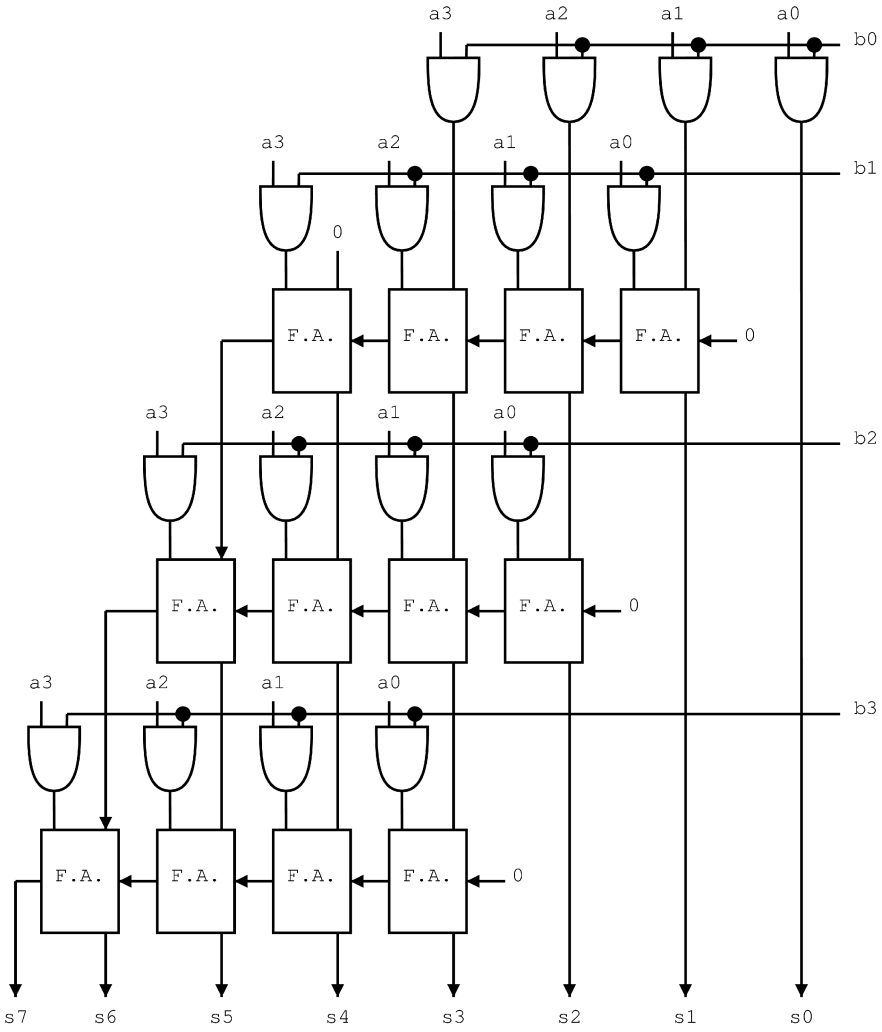


Figura 5.17. Circuito multiplicador de cuatro bits.

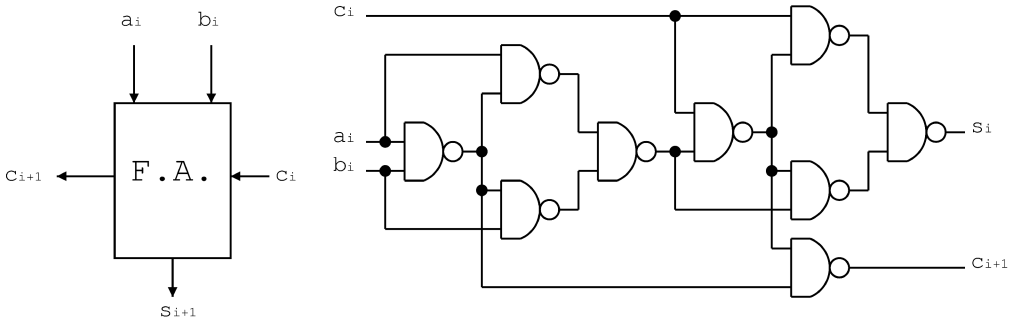


Figura 5.18. Sumador completo a nivel de puertas.

Ejemplo 5.3. Descripción en formato VERILOG del multiplicador.

```
`timescale 1ns / 1ps

module and2 ( Q, A, B );
  output  Q;
  input  A, B;
specify
    specparam CDS_LIBNAME  = "multi_lib";
    specparam CDS_CELLNAME = "and2";
    specparam CDS_VIEWNAME = "schematic";
endspecify
  na2  I0 ( .Q(net4), .B(B), .A(A));
  in1    I1 ( .A(net4), .Q(Q));
endmodule

module full_adder ( c_out, s, a, b, c );
  output  c_out, s;
  input  a, b, c;
specify
    specparam CDS_LIBNAME  = "multi_lib";
    specparam CDS_CELLNAME = "full_adder";
    specparam CDS_VIEWNAME = "schematic";
endspecify

na2  I8 ( .Q(s), .B(net15), .A(net9));
na2  I7 ( .Q(net9), .B(net18), .A(c));
na2  I6 ( .Q(c_out), .B(net30), .A(net18));
na2  I5 ( .Q(net15), .B(net21), .A(net18));
na2  I4 ( .Q(net18), .B(net21), .A(c));
na2  I3 ( .Q(net21), .B(net24), .A(net27));
na2  I2 ( .Q(net24), .B(b), .A(net30));
na2  I1 ( .Q(net27), .B(net30), .A(a));
na2  I0 ( .Q(net30), .B(b), .A(a));

endmodule

module multiplicador ( s, a, b );
  output [7:0]  s;
  input [3:0]  b;
  input [3:0]  a;
specify
    specparam CDS_LIBNAME  = "multi_lib";
    specparam CDS_CELLNAME = "multiplicador";
    specparam CDS_VIEWNAME = "schematic";
endspecify

and2 I23 ( .B(a[3]), .A(b[3]), .Q(net81));
and2 I22 ( .B(a[2]), .A(b[3]), .Q(net86));
and2 I21 ( .B(a[1]), .A(b[3]), .Q(net91));
and2 I20 ( .B(a[0]), .A(b[3]), .Q(net96));
and2 I15 ( .B(a[0]), .A(b[2]), .Q(net45));
and2 I14 ( .B(a[1]), .A(b[2]), .Q(net48));
and2 I13 ( .B(a[2]), .A(b[2]), .Q(net51));
and2 I12 ( .B(a[3]), .A(b[2]), .Q(net54));
and2 I11 ( .B(a[0]), .A(b[0]), .Q(s[0]));
and2 I10 ( .B(a[1]), .A(b[0]), .Q(net137));
```

```
and2 I9 ( .B(a[2]), .A(b[0]), .Q(net132));
and2 I8 ( .B(a[3]), .A(b[0]), .Q(net127));
and2 I7 ( .B(a[3]), .A(b[1]), .Q(net69));
and2 I6 ( .B(a[2]), .A(b[1]), .Q(net126));
and2 I5 ( .B(a[1]), .A(b[1]), .Q(net131));
and2 I4 ( .B(a[0]), .A(b[1]), .Q(net78));
full_adder I27 ( .c_out(s[7]), .c(net84), .a(net81),
.b(net82),.s(s[6]));
full_adder I26 ( .c_out(net84), .c(net89), .a(net86),
.b(net87),.s(s[5]));
full_adder I25 ( .c_out(net89), .c(net94), .a(net91),
.b(net92),.s(s[4]));
full_adder I24 ( .c_out(net94), .c(cds_globals.gnd_),
.a(net96),.b(net97), .s(s[3]));
full_adder I19 ( .c_out(net105), .c(cds_globals.gnd_),
.a(net45),.b(net102), .s(s[2]));
full_adder I18 ( .c_out(net110), .c(net105), .a(net48),
.b(net107),.s(net97));
full_adder I17 ( .c_out(net115), .c(net110), .a(net51),
.b(net112),.s(net92));
full_adder I16 ( .c_out(net82), .c(net115), .a(net54),
.b(net117),.s(net87));
full_adder I3 ( .c_out(net117), .c(net124), .a(net69),
.b(cds_globals.gnd_), .s(net112));
full_adder I2 ( .c_out(net124), .c(net129), .a(net126),
.b(net127),.s(net107));
full_adder I1 ( .c_out(net129), .c(net134), .a(net131),
.b(net132),.s(net102));
full_adder I0 ( .c_out(net134), .c(cds_globals.gnd_),
.a(net78),.b(net137), .s(s[1]));
endmodule
```

La respuesta a nivel lógico del circuito se ha comprobado con numerosos patrones de entrada y, finalmente, se han seleccionado dos secuencias consideradas suficientemente significativas. La primera secuencia de multiplicación es 0x0 (inicialización)-7x7-5xA-Ex6-FxF y los resultados obtenidos se muestran en las figuras 5.19, 5.20 y 5.21, correspondiendo éstas a resultados obtenidos con HSPICE, HALOTIS-CDM y HALOTIS-DDM respectivamente. Una segunda secuencia es 0x0-FxF-0x0-FxF-0x0, con unos resultados de simulación mostrados en las figuras 5.22, 5.23 y 5.24, también respectivamente obtenidas con HSPICE, HALOTIS-CDM y HALOTIS-DDM.

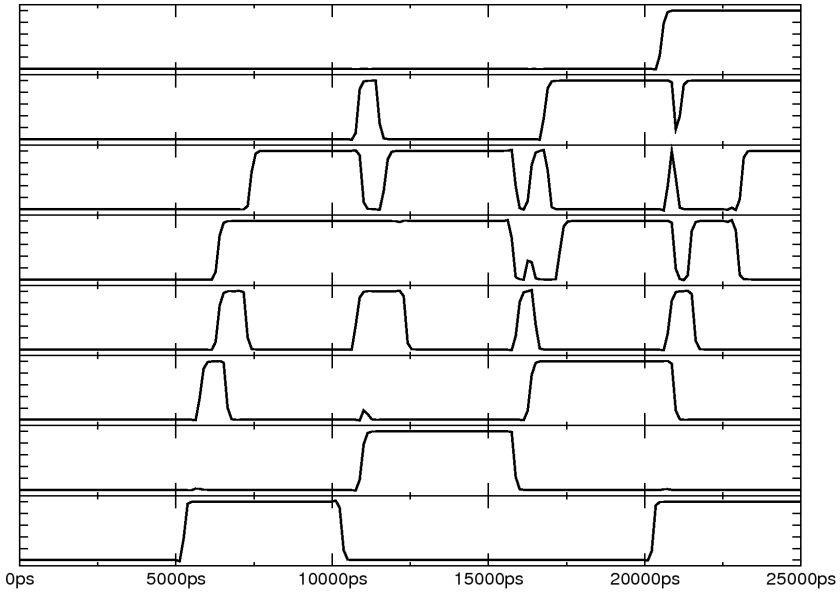


Figura 5.19. Resultados de la multiplicación de la secuencia 0x0,7x7, 5xA, FxF con HSPICE.

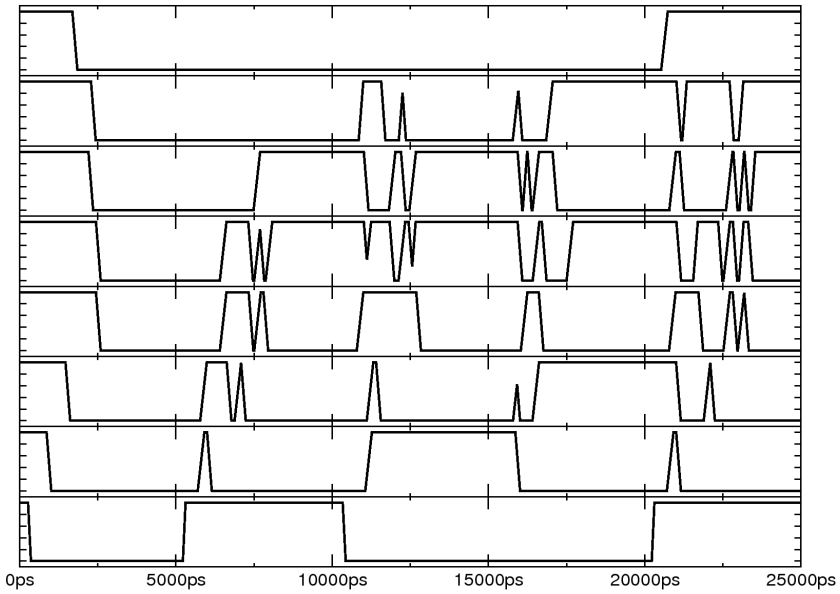


Figura 5.20. Resultados de la multiplicación de la secuencia 0x0,7x7, 5xA, FxF con HALOTIS-CDM.

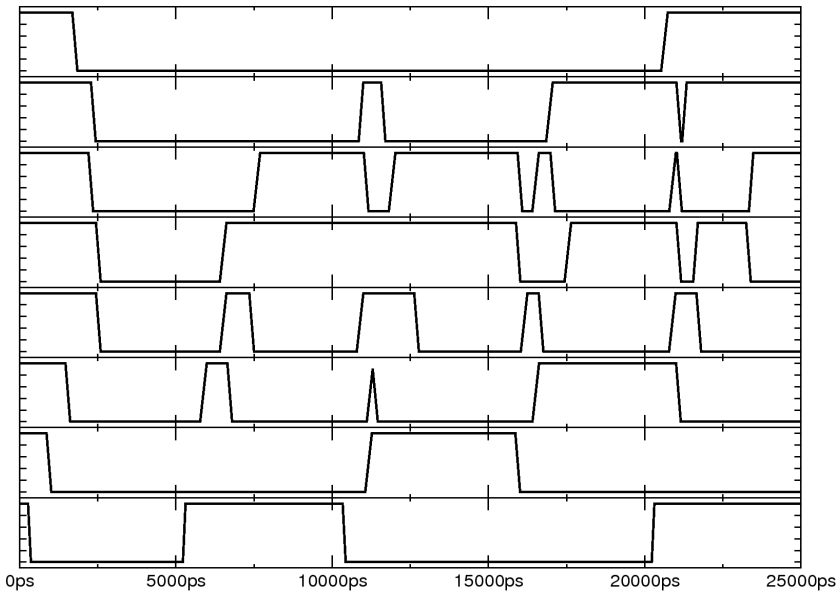


Figura 5.21. Resultados de la multiplicación de la secuencia 0x0,7x7, 5xA, FxF con HALOTIS-DDM.

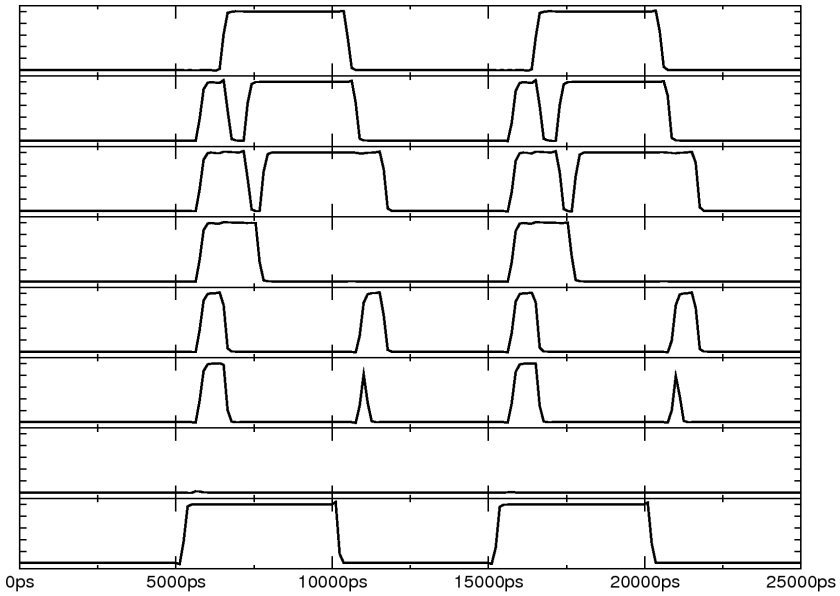


Figura 5.22. Resultados de la multiplicación de la secuencia 0x0, FxF repetida, simulada con HSPICE.

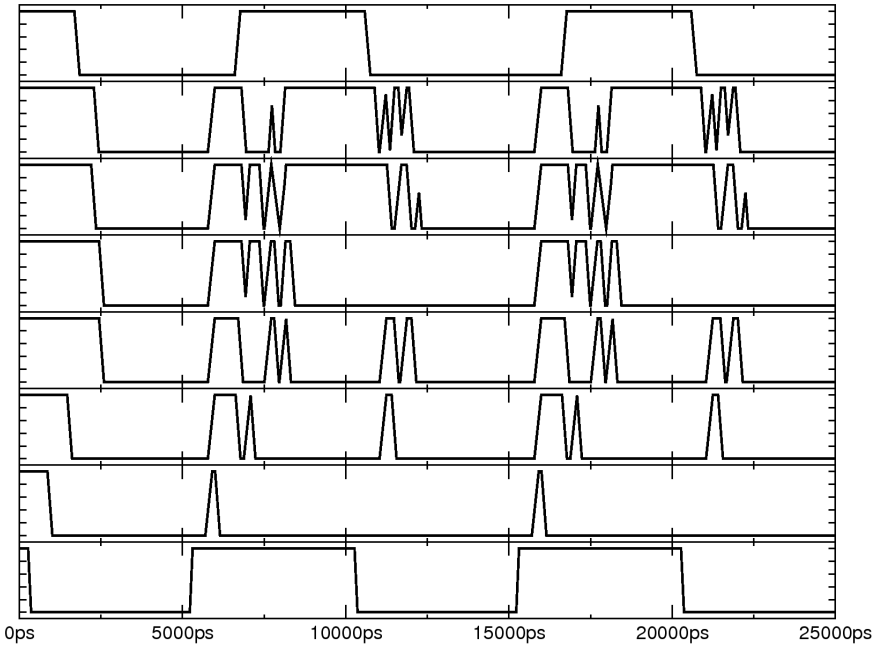


Figura 5.23. Resultados de la multiplicación de la secuencia 0x0 - FxF repetida, simulada con HALOTIS-CDM.

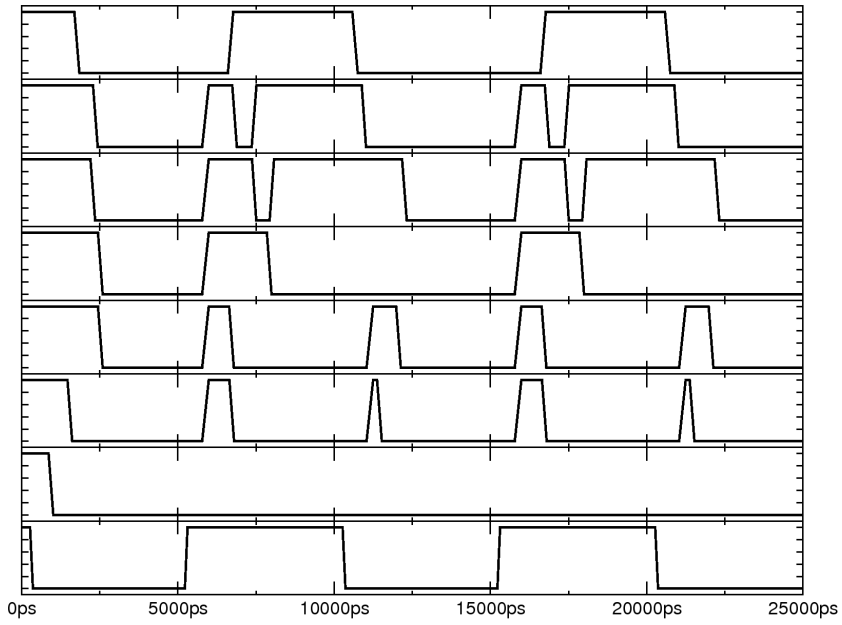


Figura 5.24. Resultados de la multiplicación de la secuencia 0x0 - FxF repetida, simulada con HALOTIS-DDM.

Con estos resultados se visualiza la mejora que aporta el modelo DDM en el tratamiento de *glitches* y azares. La diferencia significativa entre las formas de onda de un modelo de retraso convencional (no considera el efecto de degradación) con las obtenidas tanto con HSPICE como con el modelo DDM conduce a una aplicación muy importante del simulador HALOTIS-DDM, la medida de la actividad de conmutación (*switching activity*), que es de enorme utilidad en el proceso de diseño de sistemas digitales.

Las medidas de la actividad de conmutación se han incluido en la tabla 5.6. Esta tabla muestra los resultados obtenidos a partir de las simulaciones realizadas con HSPICE, HALOTIS-DDM y un simulador comercial como es VERILOG. Los resultados confirman lo observado a nivel de forma de ondas, esto es, que los simuladores convencionales sobrestiman la actividad de conmutación con un error muy alto. Dada esta situación, se concluye que cualquier metodología de diseño basada en este parámetro para comparar diseños en cuanto al consumo de potencia o ruido de conmutación no tiene mucha fiabilidad.

Secuencia	HSPICE	HALOTIS-DDM	VERILOG
0x0-7x7-5xA-Ex6-FxF	416	408 (2%)	697 (68%)
0xF-0xF repetida	584	608 (4%)	857 (47%)

Tabla 5.6. Actividad de conmutación del multiplicador medida respecto al número de transiciones.

En la siguiente sección se desarrolla un análisis más exhaustivo sobre la medida de la actividad de conmutación comprobándose de nuevo este hecho.

5.2. Aplicación de HALOTIS a la estimación del consumo de potencia e intensidad

Uno de los trabajos más importantes desarrollados en esta Tesis es haber conseguido aplicar la simulación lógica-temporal a la estimación muy precisa del consumo de potencia e intensidad. Esta precisión se consigue por una parte, debido al modelo DDM que proporciona formas de onda mucho más precisas que los modelos de retraso convencionales incluidos en las herramientas de simulación lógica temporal y, por otra, por el modelo de intensidad incluido en HALOTIS y expuesto en el capítulo 3.

Esta sección está dedicada a mostrar los resultados, muy significativos, obtenidos en esta aplicación de HALOTIS a la estimación de potencia-intensidad. Los resultados aparecen divididos en dos partes: (1) medida de la actividad de conmutación y, (2) medida de las curvas de intensidad.

5.2.1. Medida de la actividad de conmutación

La actividad de conmutación es un parámetro clave en el proceso de diseño para realizar análisis del consumo de potencia [GHOSH92] e incluso del ruido de conmutación generado por los circuitos digitales [ARAG99]. Este parámetro se obtiene directamente del número de transiciones en cada nodo.

Como ya se comprobó en la sección 5.1.4, medir la actividad de conmutación mediante simulación lógica-temporal (método ampliamente extendido hoy en día) da lugar a grandes inexactitudes al no considerar la degradación. Con este antecedente, esta sección presenta un estudio exhaustivo sobre la medida de la actividad de conmutación empleando tres herramientas diferentes como son HSPICE, que se considerarán resultados cuasi-experimentales, HALOTIS-DDM y el simulador VERILOG.

El estudio pormenorizado está publicado en [BAENA02] y en el capítulo 8 del libro "*Logic Timing Simulations*" [BELL06]. Esta sección se centra en presentar los resultados comparativos entre HALOTIS y VERILOG mostrándose la eficiencia del primero.

Para este estudio, en primer lugar, se ha desarrollado en [BAENA02, BELL06] una metodología de medida de la actividad de conmutación asociada a cada una de las herramientas. Dos aspectos fundamentales y comunes a dichas metodologías son, por una parte, el conjunto de circuitos analizados y, por otra parte, el conjunto de patrones para cada circuito. Respecto los primeros, han sido escogidos los circuitos del *benchmark* ISCAS'85 [BRYAN85], ampliamente aceptados como banco de referencia de pruebas tanto de metodologías como de herramientas y, respecto los segundos, es importante escoger un conjunto de patrones adecuado al existir una fuerte dependencia entre los patrones y la actividad de conmutación en un circuito [NAJM94].

Los circuitos del *benchmark* ISCAS'85 se enumeran en la tabla 5.7 indicando la función lógica que realizan y algunos datos referentes a la complejidad de cada uno de ellos. Las principales razones para utilizarlos son:

- Se consideran circuitos estándar y están disponibles.
- Implementan funcionalidades muy diferentes.
- Tienen gran diversidad en el número de entradas y salidas.
- Implementan componentes lógicos habituales.
- Presentan gran diversidad en el número de componentes y número de niveles.

Circuito	Función	Entradas	Salidas	Puertas
c17	Puerta NAND6	7	1	7
c432	Control de interrupciones (27 canales)	36	7	160
c499	Detector/Corrector de errores de 32 bits	41	32	202
c880	ALU de 8bits	60	26	383
c1355	Detector/Corrector de errores de 32 bits	41	32	546
c1908	Detector/Corrector de errores de 16 bits	33	25	880
c2670	ALU de 12 bits y controlador	233	140	1193
c3540	ALU de 8 bits	50	22	1669
c5315	ALU de 9 bits	178	123	2307
c6288	Multiplicador 16 bits	32	32	2416
c7552	Sumador comparador de 32 bits	207	108	5312

Tabla 5.7. Circuitos combinacionales de prueba propuestos en ISCAS'85.

Como se indicó anteriormente, los resultados de actividad de conmutación están directamente relacionados con el conjunto de patrones que se utilicen [NAJM94]. Para minimizar la variación bastaría con utilizar unos patrones aleatorios muy largos, obteniéndose así unos resultados medios. El principal problema al aplicar este método, es la simulación eléctrica de circuitos complejos, que hace inviable el proceso de simulación al tener gran cantidad de patrones. En [BELL06] se propone un método basado en los trabajos [CIRIT87], [NAJM91] y [BURCH93] que permite escoger un conjunto de patrones reducido a partir de simulaciones lógicas de 1000 patrones aleatorios, de los que se miden las transiciones globales en cada simulación. Con estos resultados se obtiene una estimación de la actividad de conmutación con un error por debajo del 3%, y por tanto, puede obtenerse una secuencia de menor tamaño (50 patrones), procesando los 1000 patrones iniciales, tales que, generen una actividad de conmutación similar a la media obtenida con estos 1000 patrones iniciales. Una vez escogidos los patrones se han convertido al formato HSPICE y HALOTIS mediante dos programas.

Por otro lado, los *netlist* se han obtenido partiendo de una descripción VERILOG, utilizando la herramienta DFWII[9] con una tecnología de AMS CMOS 0.35μm. Para llevar a cabo la simulación con HALOTIS no basta con tener la descripción VERILOG de los circuitos, además, es necesario caracterizar cada una de las celdas lógicas que aparecen en dichos *netlists*.

Cada una de las celdas lógicas de los que se componen estos *benchmarks,* ha sido extraída por separado con DFWII y después, caracterizada con las herramientas AUTODDM y AUTOTPN. Tras esto, con el conjunto de parámetros obtenidos se puede crear la librería HALOTIS capaz de simular todos los *benchmarks.* Finalmente esta librería consta de un total de 21 puertas diferentes

9. *Design Frame Work II*

mostradas en la tabla 5.8 y, su definición completa puede consultarse en el anexo 1.

Con los conjuntos de 50 patrones para cada circuito, los *netlists* en formato VERILOG y HSPICE y la librería de celdas de HALOTIS caracterizada, se han realizado las simulaciones con las tres herramientas: VERILOG, HSPICE y HALOTIS, obteniéndose los resultados de la tabla 5.9.

Tipo	Número de entradas	Total de puertas
AND	2, 3, 4, 5, 8	5
BUFFER	1	1
XOR	2	1
INV	1	1
NAND	2, 3, 4, 5, 8	5
NOR	2, 3, 4, 8	4
OR	2, 3, 4, 5	4

Tabla 5.8. Puertas lógicas del benchmark ISCAS'85.

Circuito	HSPICE	VERILOG		HALOTIS	
		Transiciones	Error	Transiciones	Error
c432	4517	4735	4,8%	4599	1,8%
c499	6196	6417	3,6%	5336	-13,9%
c880	11033	11337	2,8%	11189	1,4%
c1355	13960	16190	16%	13948	-0,8%
c1908	25873	32411	25,3%	28123	8,7%
c2670	38655	44979	16,4%	36774	-4,9%
c3540	52303	60920	16,5%	56880	8,7%
c5315	79803	100295	25,7%	81852	2,5%
c6288	194784	418815	115%	242014	24,2%
c7552	144535	174292	20,9%	146251	1,2%

Tabla 5.9. Comparación de la actividad de conmutación con HSPICE, VERILOG y HALOTIS.

El análisis de los resultados concluye que HALOTIS tiene mayor precisión que VERILOG en todos los casos, llegando incluso a bajar el gran error cometido en el circuito multiplicador 16 bits (c6288) del 115% de VERILOG, al 24,2% de HALOTIS. En tres casos (c499, c1375 y c2670) HALOTIS obtiene menor actividad de conmutación debido a que el modelo DDM filtra los *glitches* en los primeros niveles mientras que en la realidad sí son propagados, provocando la desaparición de las correspondientes transiciones en el resto de niveles del circuito.

Los tiempos de ejecución de la simulación en el computador también se han medido y se incluyen en la tabla 5.10. Los tiempos de VERILOG y HALOTIS no son comparables en valores absolutos, ya que VERILOG incluye en un solo paso de ejecución la lectura del *netlist,* patrones y celdas. HALOTIS en cambio, utiliza

herramientas de preprocesado (*hcells*, *hverilog)* para estas tareas, que consumen tiempo de CPU que no se muestra en la tabla. Sí es comparable el aumento de tiempo de ejecución con la complejidad del circuito. Tanto VERILOG como HALOTIS siguen el mismo orden de magnitud en los diferentes circuitos, mientras HSPICE necesita tiempos muy elevados.

En la sección siguiente se analiza en profundidad el comportamiento de HALOTIS frente a los diferentes tamaños de circuitos y patrones, incluyéndose datos sobre los recursos informáticos utilizados en todos los casos.

Circuito	**Tiempo de CPU en segundos**		
	HSPICE	**VERILOG**	**HALOTIS**
c432	2714	6,4	0,05
c499	8087	7,2	0,08
c880	15240	8,3	0,12
c1355	28411	8,3	0,15
c1908	83989	9,9	0,25
c2670	260106	16,7	0,36
c3540	722935	14,8	0,44
c5315	1518849	24,1	0,78
c6288	836644	34,2	2,02
c7552	4577727	31,3	1,35

Tabla 5.10. Comparación de los tiempos de ejecución de los simuladores.

5.2.2. Resultados del modelo de intensidad

La implementación realizada del modelo de intensidad propuesto en la capítulo 3 en HALOTIS, ha sido verificada con varios circuitos de diversa complejidad y formados por diferentes tipos de puertas. El primer resultado significativo es la cadena de inversores utilizada a lo largo del capítulo como ejemplo donde se aprecian visualmente las formas de onda, los diferentes efectos, en este caso, es relevante observar las aportaciones triangulares de intensidad de cada inversor.

Seguidamente, se presentan otros dos circuitos, en los que aumenta la complejidad. El primero es el sumador completo en el que aparecen resultados del modelo de intensidad para puertas de más de una entrada y, el segundo, el multiplicador de 4 bits presentado en este capítulo con anterioridad, pero ahora, desarrollado en tecnología AMS CMOS 0.35μm.

5.2.2.1. Cadena de inversores

El primer ejemplo que se presenta es la cadena de inversores de la figura 5.1

que, a pesar de su relativa simplicidad, describe de forma destacada las capacidades y posibilidades del modelo de intensidad desarrollado. Esta cadena de inversores representa una posible ruta activa en un circuito combinacional multinivel pudiéndose, por tanto, ver el efecto de la suma de cada aportación a la intensidad global producida por las transiciones, al superponerse en algunos instantes de tiempo. También es un buen ejemplo para comparar las diferencias en los resultados cuando se combina el modelo de degradación con el de intensidad.

En el circuito propuesto se han utilizado dos patrones diferentes: el primero intenta mostrar la respuesta del modelo en ausencia de degradación, en el segundo el efecto de degradación tiene lugar durante la simulación y, se aprecia la mejora en los resultados al combinar ambos modelos.

El primer patrón es un pulso ancho en la entrada de la cadena de inversores. Este pulso es completamente propagado a través de la cadena, siendo lo suficientemente ancho como para que no ocurra degradación. La figura 5.25 y 5.26 muestran las formas de onda obtenidas en los nodos internos simuladas con HSPICE y HALOTIS respectivamente. La simulación lógica se ajusta con la simulación eléctrica con mucha precisión.

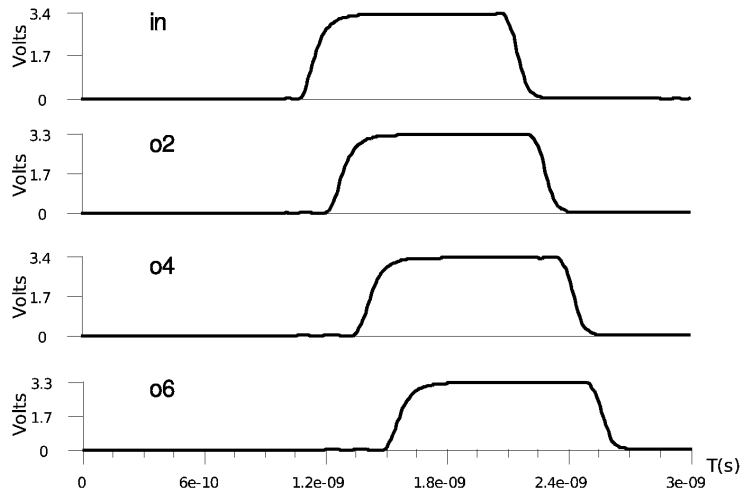


Figura 5.25. Formas de onda para un pulso ancho con HSPICE.

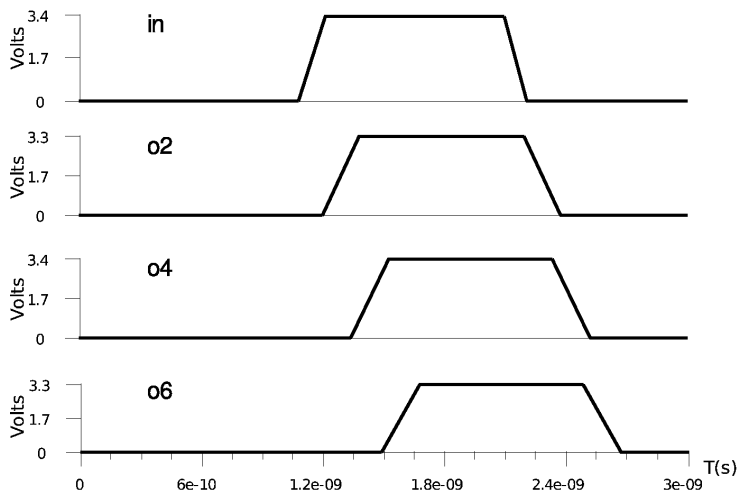


Figura 5.26. Formas de onda para un pulso ancho con HALOTIS.

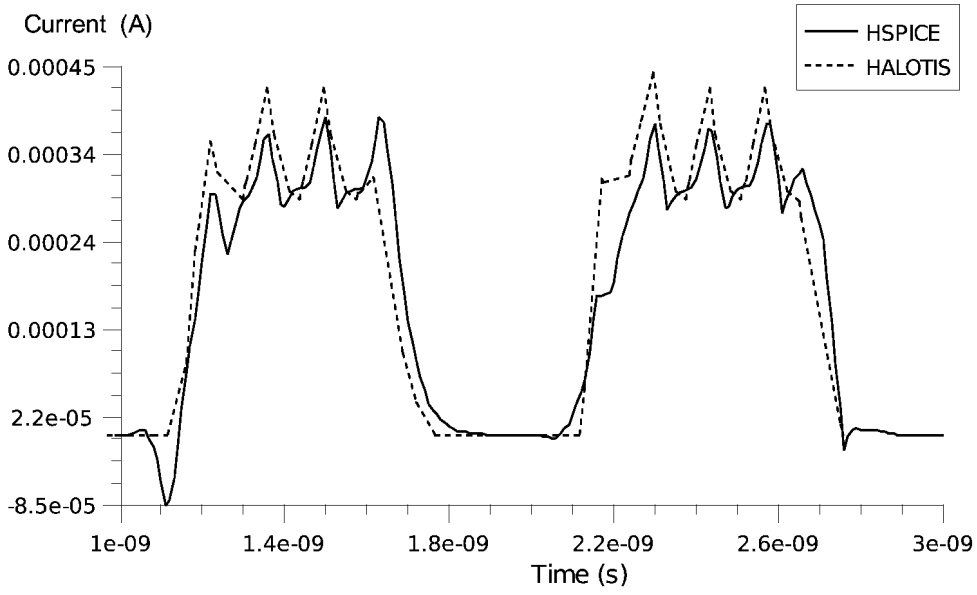


Figura 5.27. Curva de intensidad para un pulso ancho en la cadena de inversores.

Cuando se realiza la simulación incluyendo el modelo de intensidad, el simulador lógico calcula las aportaciones a la curva de intensidad producidas por cada una de esas transiciones y realiza la suma de todas para obtener una curva de intensidad global. La curva de intensidad resultante calculada por HALOTIS se muestra en la figura 5.27, superpuesta a los resultados obtenidos por HSPICE. Ambas curvas encajan con bastante precisión y se puede observar que los picos de corriente son reproducidos a nivel lógico con precisión.

El segundo patrón es un tren de pulsos aplicado en el primer inversor. Los pulsos son lo suficientemente estrechos para ser degradados siendo algunos filtrados debido al efecto de degradación a lo largo de la cadena. En esta simulación se comparan tres resultados: HSPICE, HALOTIS utilizando el modelo DDM y HALOTIS utilizando el modelo convencional que no incluye efecto de degradación (figuras 5.28, 5.29 y 5.30).

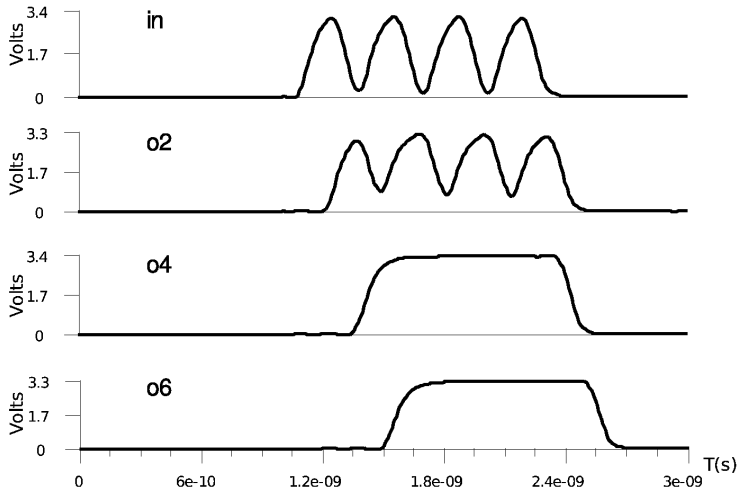


Figura 5.28. Formas de onda del tren de pulsos obtenidas con HSPICE.

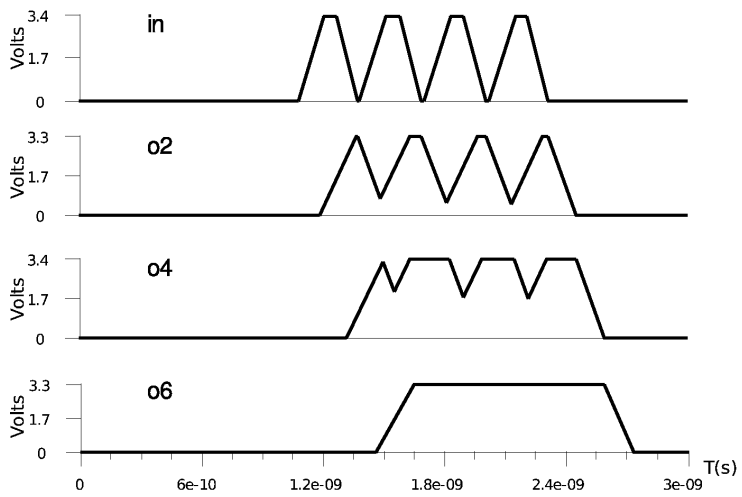


Figura 5.29. Formas de onda del tren de pulsos obtenidas con HALOTIS-DDM.

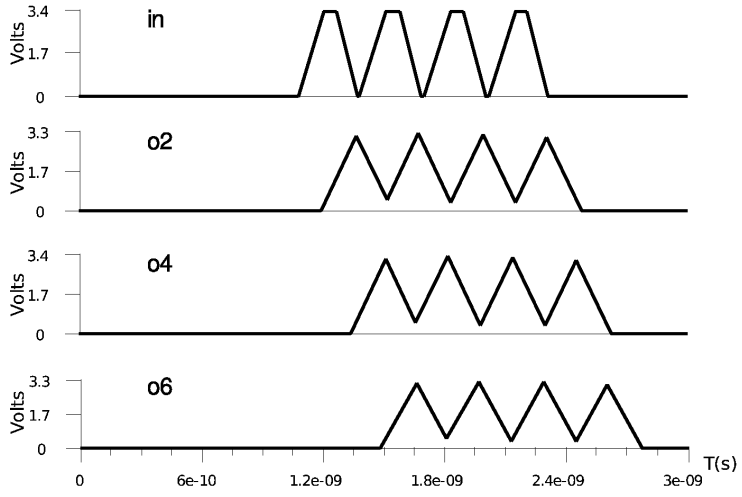


Figura 5.30. Formas de onda del tren de pulsos obtenidas con HALOTIS-CDM.

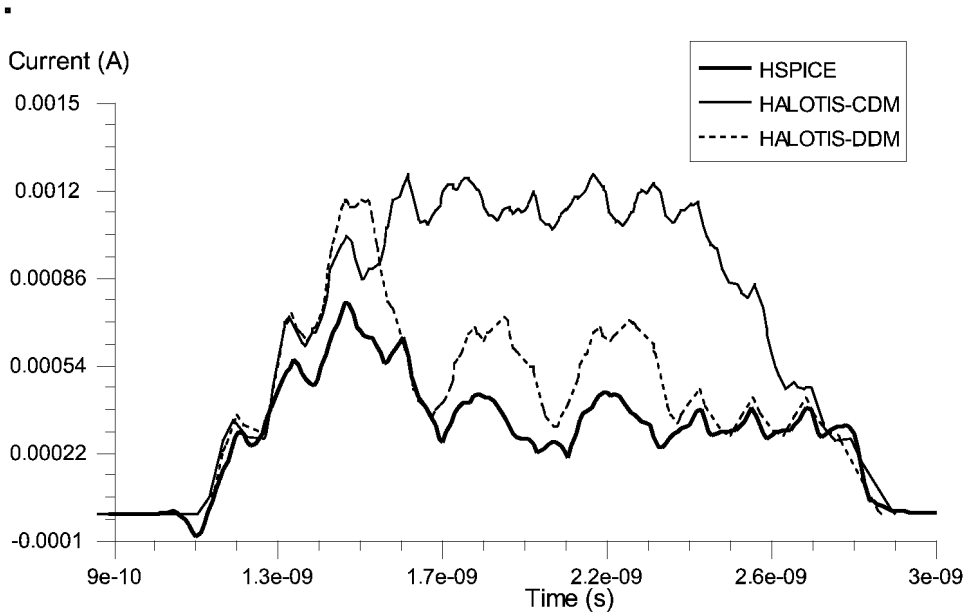


Figura 5.31. Comparación de curvas de intensidad para el tren de pulsos con degradación.

Este ejemplo muestra la aportación del modelo DDM a la precisión en el cálculo de la curva de intensidad. La figura 5.31 representa las tres curvas de intensidad quedando por encima la que se ha calculado con un modelo convencional. Al propagarse los pulsos en todos los niveles, el cálculo global es superior al obtenido con HSPICE, mientras que el modelo DDM tiene mucho mejor ajuste.

El consumo de energía se obtiene calculando el área total que abarca cada curva de intensidad. En la tabla 5.11 se ha tomado como referencia HSPICE donde se ha fijado éste área como área unitaria, calculando así el error relativo cometido por los otros dos modelos. Tal y como muestran estos resultados, obviar el efecto de degradación conlleva cometer un gran error en el consumo de energía, para el ejemplo, hasta un 129% frente al 30% que se obtiene al incluirlo.

Consumo de energía normalizado		
HSPICE	1	0%
DDM	1,3	+30%
CDM	2,29	+129%

Tabla 5.11. Consumo de energía normalizado en la cadena de inversores.

5.2.2.2. Sumador completo

El segundo circuito es el sumador completo de la figura 5.32 (presentado anteriormente en la figura 5.18), el cual, está formado en su totalidad por puertas NAND de dos entradas. Se considera un buen circuito para verificar el modelo de intensidad en puertas de más de una entrada.

Con este objetivo se ha simulado un patrón simple con HSPICE, HALOTIS-CDM, HALOTIS-DDM. El patrón contiene una primera transición en la que cambian todas las entradas, originando gran actividad de conmutación. Esta primera transición muestra claramente, en la figura 5.33, la diferencia entre los modelos DDM y CDM (la forma de onda CDM se prolonga más tiempo). El resto del patrón contiene una secuencia *gray,* la cual genera poca actividad de conmutación, observándose la superposición de las formas de onda de HSPICE a ambos modelos de comportamiento.

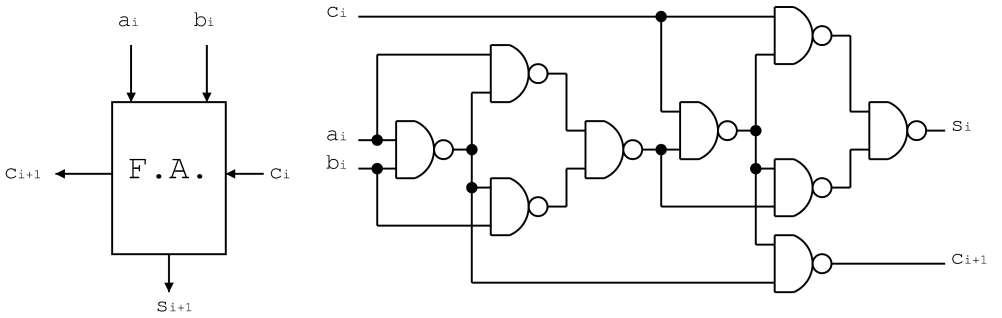


Figura 5.32. Sumador completo a nivel de puertas.

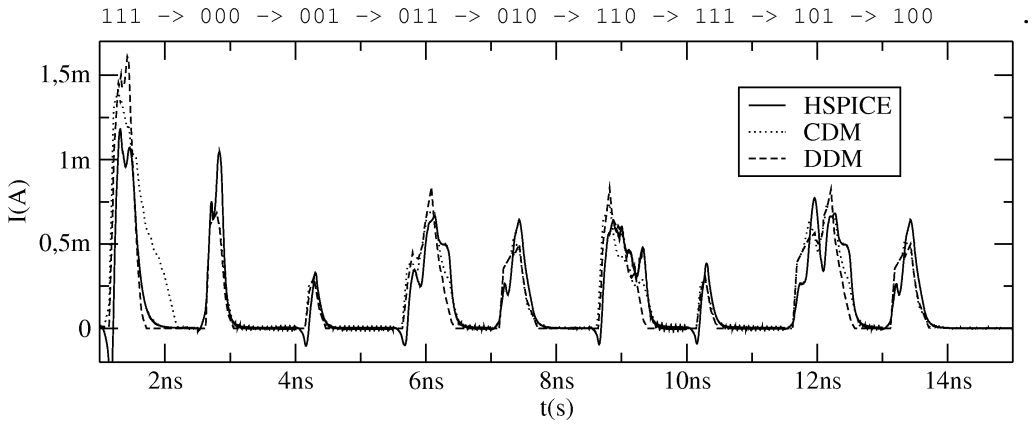


Figura 5.33. Comparación de curvas de intensidad del sumador completo.

Calculando el área de la curva de intensidad, se comprobó en la sección 3.4.1 que se obtiene una estimación del consumo de potencia para en un intervalo de tiempo, simplemente dividiendo el área entre dicho intervalo de tiempo. HSPICE

calcula dicha estimación mediante una sentencia de medida, dando el resultado para el intervalo de tiempo ya calculado. En caso de HALOTIS, devuelve la energía, de forma que, dividiendo entre el intervalo de tiempo de simulación, se obtiene la estimación en *Watios,* la misma representación de la potencia que ofrece HSPICE.

Estos cálculos se han realizado para el patrón completo y se incluyen en la tabla 5.12.

	Potencia	**Error**
HSPICE	0.50248mW	0%
DDM	0.498777mW	+0.74%
CDM	0.589504mW	+17.32%

Tabla 5.12. Consumo de energía desde 1ns a 14ns en el sumador completo.

5.2.2.3. Multiplicador de 4 bits

El último circuito simulado es el multiplicador de 4 bits, utilizado anteriormente (figura 5.17), para el cual se han obtenido las curvas de intensidad de diferentes transiciones. Este circuito, del mismo modo que el anterior, ha sido implementado con tecnología AMS CMOS 0.35μm, el cual contiene tres tipos de puertas: inversores, NAND y AND. Todas han sido caracterizadas para todos los modelos necesarios para simular con HALOTIS.

Desde la figura 5.34 hasta la figura 5.39 presentan sucesivamente la simulación del patrón: 0x0,7x7, 5xA, Ex6, FxF, 0x0. Cada una de las transiciones se ha separado en una figura y se han incluido las formas de onda de intensidad generadas por HSPICE, HALOTIS-CDM y HALOTIS-DDM para poder realizar una comparación visual.

A pesar del cambio de tecnología respecto a la sección 5.1.4, los resultados a nivel lógico son similares por lo que no se incluyen aquí, quedando los resultados centrados en el modelo de intensidad.

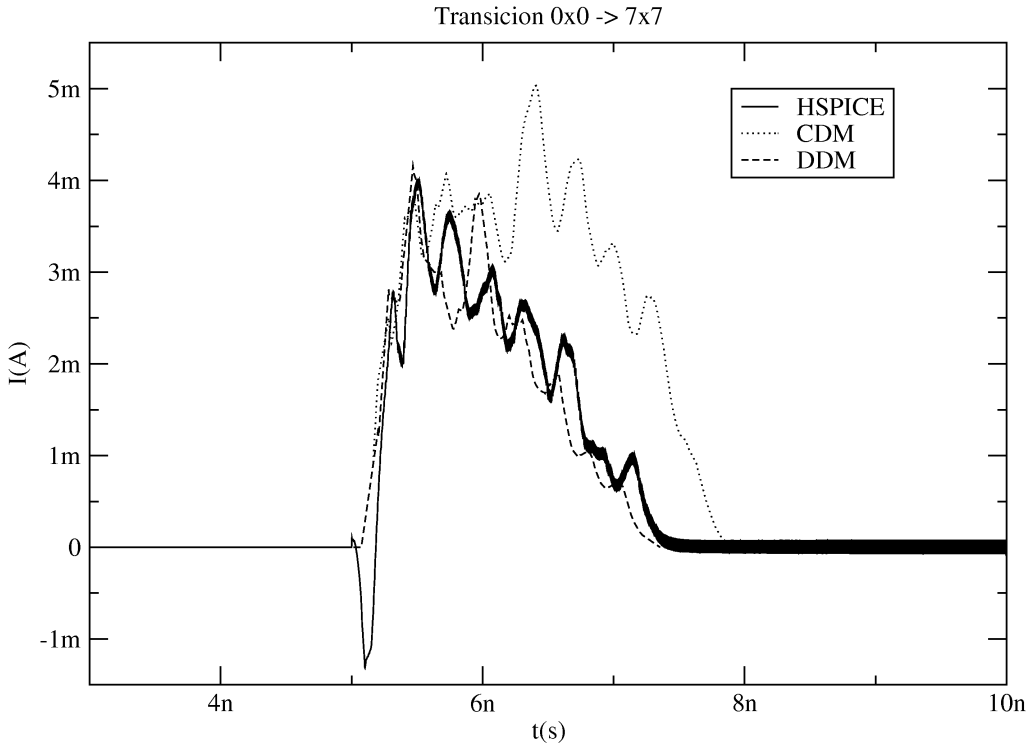


Figura 5.34. Comparación de curvas de intensidad para la transición de 0x0 a 7x7.

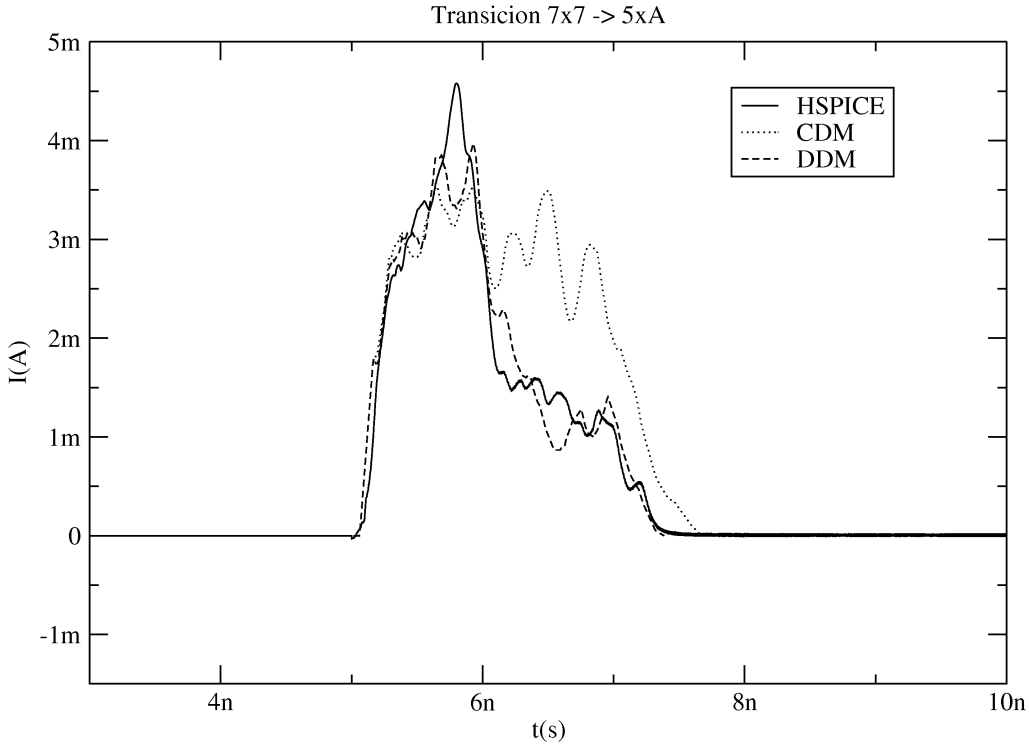


Figura 5.35. Comparación de curvas de intensidad para la transición de 7x7 a 5xA.

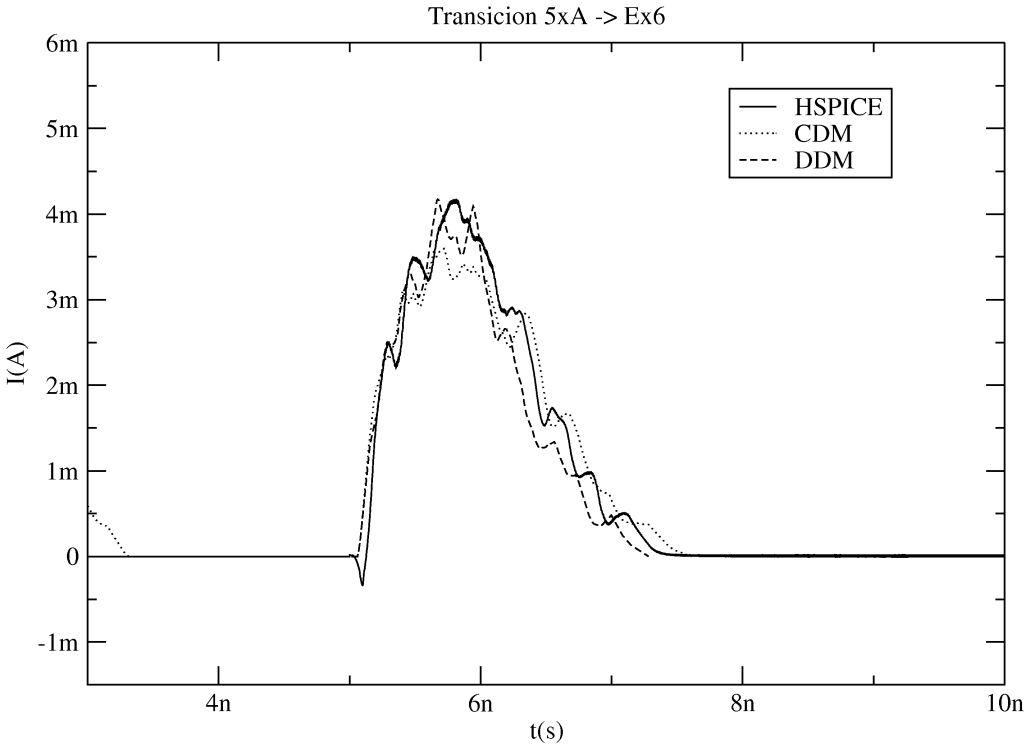


Figura 5.36. Comparación de curvas de intensidad para la transición de 5xA a Ex6.

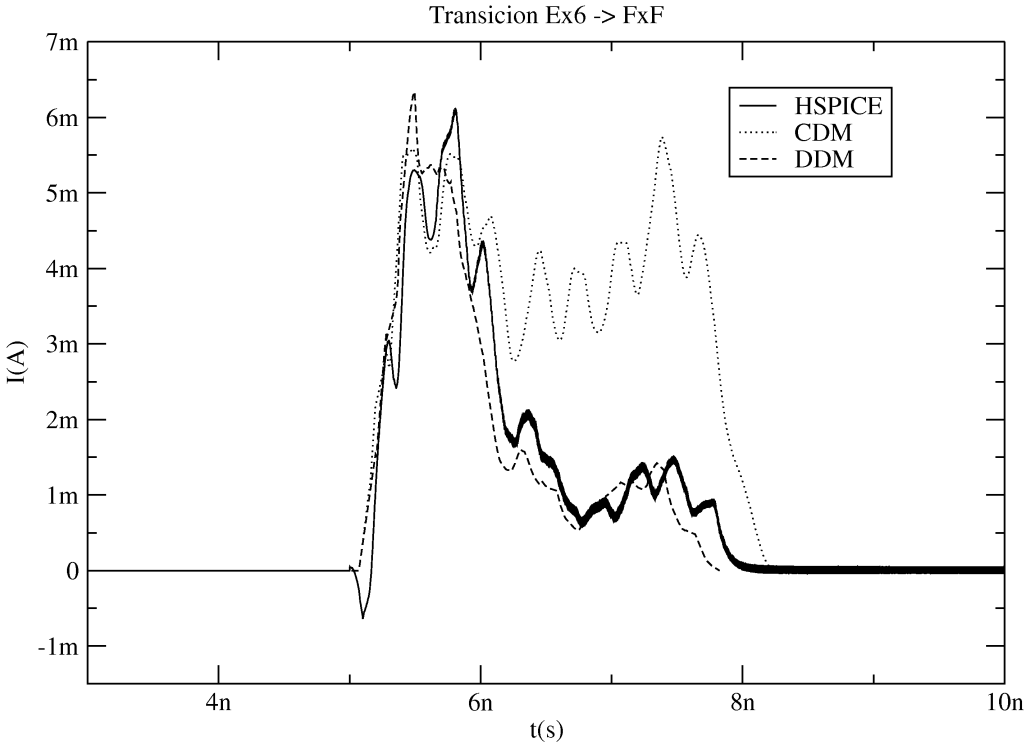


Figura 5.37. Comparación de curvas de intensidad para la transición de Ex6 a FxF.

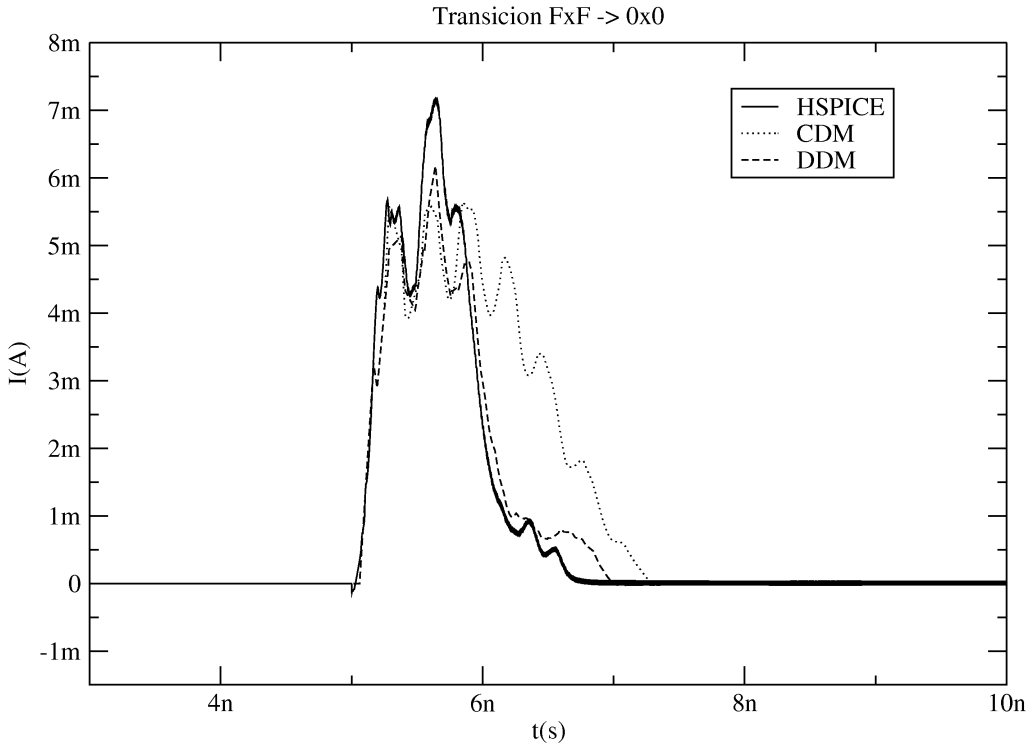


Figura 5.38. Comparación de curvas de intensidad para la transición de FxF a 0x0.

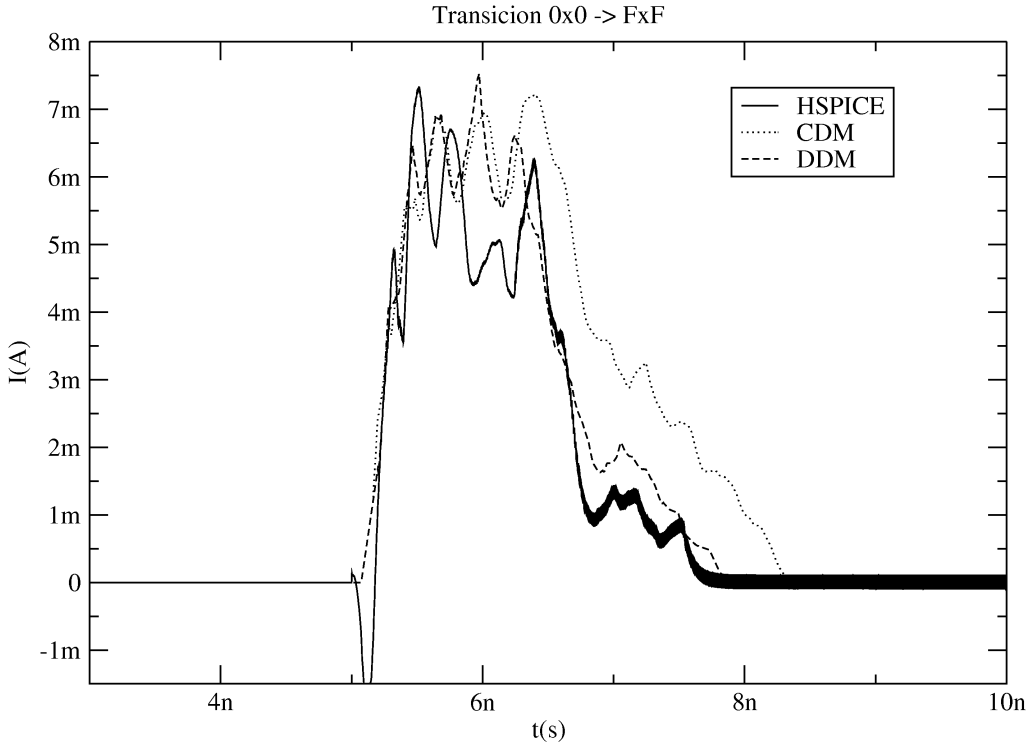


Figura 5.39. Comparación de curvas de intensidad para la transición de 0x0 a FxF.

En todas las figuras anteriores se observa cualitativamente la sobreestimación producida por el modelo sin degradación (CDM). Siempre la forma de onda de intensidad se prolonga en el tiempo respecto a la de HSPICE, obteniéndose una estimación de la potencia superior a la real. Esta potencia se ha medido en todos los casos y se resume en la tabla 5.13.

Transición	HSPICE	DDM	CDM	Error DDM	Error CDM
0x0 a 7x7	2.9069mW	2.83966mW	5.29372mW	2.3%	82%
7x7 a 7xA	2.9072mW	2.8967mW	3.92686mW	0.4%	35%
5xA a Ex6	3.1264mW	2.85124mW	3.10778mW	8,8%	0.5%
Ex6 a FxF	3.9576mW	3.66334mW	7.51052mW	7.4%	89%
FxF a 0x0	3.3368mW	3.21318mW	4.64958mW	2.8%	39%
0x0 a FxF	5.4064mW	6.0098mW	8.1574mW	11%	51%

Tabla 5.13. Consumo de potencia en diferentes transiciones del multiplicador de 4 bits.

Con estos resultados se puede concluir que con el modelo DDM además de obtenerse mejores resultados en la estimación de potencia/energía, el error cometido presenta una mayor estabilidad frente a la disparidad de los errores cuando no se contempla la degradación durante la simulación. DDM mantiene el error alrededor del 10%, mientras que un modelo de comportamiento convencional oscila desde un 1% hasta un 90%.

Resultados parecidos se obtuvieron en la sección 5.2.1 donde se utilizaron circuitos de complejidad similar al multiplicador. En ese caso se utilizaron los *benchmarks* midiendo las transiciones como representación de la actividad de conmutación. La magnitud de los errores que se obtuvieron en la tabla 5.9 son del mismo orden de magnitud que los obtenidos en la tabla 5.13, es decir, con modelos y simuladores tradicionales (VERILOG en el caso anterior) el error en la estimación del consumo de potencia/actividad de conmutación es variable en un rango del 100%.

5.3. Rendimiento de HALOTIS

Un aspecto clave para evaluar el software es poder estimar los recursos necesarios para la ejecución según la carga que trabajo. Los procesos de simulación eléctrica (digital o analógica) son de naturaleza secuencial, es decir, durante el proceso de simulación no existe posibilidad de interacción con el usuario. Esto facilita la estimación de recursos informáticos, ya que dependen únicamente de los datos de entrada en cada herramienta.

Anteriormente en el capítulo 1, se enumeraron una serie de parámetros que

se usan para tomar medidas de rendimiento en una simulación guiada por eventos, como es la productividad, medida en eventos procesados por segundo y tiempos de realización de determinadas tareas. No obstante, el análisis de los resultados en esta sección se realiza estableciendo tres bloques de ejecución claramente diferenciados en el proceso de simulación: (1) Lectura del *netlist, (*2) Procesado de patrones y, (3) Proceso de simulación. Además, se mide el tiempo de ejecución de cada bloque desde dos puntos de vista, por un lado, en función del tamaño del circuito (número de puertas y conexiones) y, por otro lado, en función de los estímulos iniciales (patrones y eventos). A partir del número de patrones iniciales se obtiene información sobre el rendimiento de la cola de eventos implementada en HALOTIS, puesto que, al tener mayor número de patrones, obtendremos mayor número de eventos inicialmente encolados.

Por otro lado, el banco de pruebas utilizado debe ser significativo. Por los motivos expuestos en la sección anterior, se consideran el *benchmark* del ISCAS'85 un conjunto bastante completo para realizar las pruebas del rendimiento del entorno HALOTIS. Del conjunto de programas que incluye el entorno HALOTIS, el que se analiza profundamente es el motor de simulación. Aunque también se muestran resultados del procesamiento de *netlist* (ejecutable *hverilog),* el ejecutable *halotis* es el que contiene el motor de simulación y es el responsable de manejar todas las estructuras representativas del *netlist*, librería de celdas, patrones y algoritmo de simulación.

El objetivo perseguido con estas pruebas es comprobar la ausencia de errores en el orden de los tiempos de ejecución de los algoritmos implementados en HALOTIS. Principalmente, el tiempo de ejecución se consume en el acceso a los datos de las diferentes estructuras accedidas reiteradamente durante la simulación (cola de eventos, *netlist*, librería de celdas, etc.). En el capítulo de implementación se presentaban los órdenes de tiempo de ejecución para las estructuras de datos, en función del tamaño de los datos. Todos estos tiempos tienen órdenes lineales o logarítmicos, no siendo en ningún caso de órdenes superiores, ni por supuesto exponenciales. Si así fuese, existirían determinados circuitos que no se podrían simular en un tiempo razonable.

5.3.1. Tiempo de ejecución y productividad

El primer análisis ha consistido en medir el tiempo de ejecución del procesado del *netlist* de cada uno de los circuitos con la herramienta *hverilog*. Estos resultados se incluyen en la tabla 5.14 y, se han representado en la figura 5.40 mostrando un aumento del tiempo de procesado del circuito, lineal con el número de puertas. La regresión marcada en esta figura con línea de puntos se aproxima por:

$$T_{netlist}(ms)=(K_{setup}+0{,}1995\times N_{puertas})\times \nu$$

$$T_{netlist}(ms)=(0{,}0335+0{,}0001995\times N_{puertas})\times \nu \quad (5.1)$$

con un coeficiente de correlación de 0.992, siendo K_{setup} el tiempo de inicialización del simulador y v una constante que dependerá de la velocidad de la máquina sobre la que se simule, siendo 1 para la máquina sobre la que se han realizado las pruebas. Este resultado afirma que el módulo *hverilog* no contiene ningún error en los algoritmos que controlan las estructuras de tratamiento de los componentes del *netlist*, y que cumple las expectativas planteadas durante la fase de diseño e implementación.

Circuito	Puertas	Tiempo de procesado
c17	17	10ms
c432	160	50ms
c499	202	70ms
c880	383	90ms
c1355	546	150ms
c1908	880	220ms
c5315	2307	600ms
c7552	5312	1050ms

Tabla 5.14. Tiempos de lectura y procesado del netlist con hverilog.

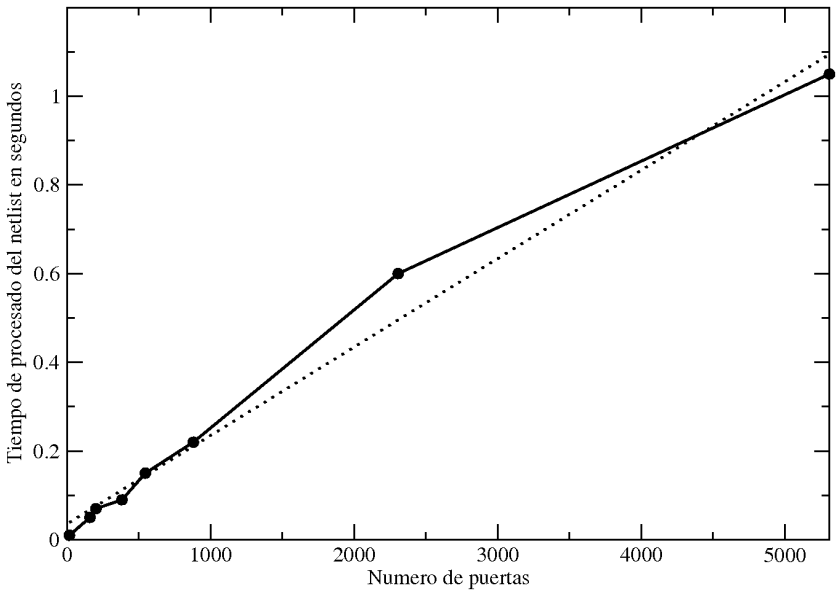


Figura 5.40. Tiempo de procesado del netlist frente al número de puertas.

Una segunda batería de pruebas consiste en realizar simulaciones con un gran número de patrones observando el comportamiento del motor de simulación ante el incremento de los estímulos iniciales. En estas segundas pruebas, se realiza un primer análisis sobre el número de componentes del circuito (figura

5.41) concluyendo que, el tiempo de procesado de los patrones no depende significativamente del número de componentes, sino más bien de la estructura interna del mismo (*fanout*, número de conexiones, número de entradas, etc.). Por ello, se obtienen valores tan dispares en las representaciones.

En cambio, en un segundo análisis respecto al tiempo de procesado de los patrones, separando cada circuito, sí concluye con una dependencia. La dependencia entre el número de patrones y el tiempo de procesado de los mismos es de orden *nlog(n)*. Efectivamente, las estructuras de datos que controlan este procesado forman dos bucles anidados tales que:

- El bucle exterior realiza la búsqueda del componente por nombre y, es un algoritmo de orden $log_2 C$ (donde C es el número de componentes). Es así por ser el índice de nombres un árbol binario.
- Tras encontrar el componente, el bucle interno implementa la inserción de cada evento en la cola, donde también el tiempo de ejecución es de orden logarítmico respecto al número de eventos. El número de eventos en la cola es lineal respecto al número de patrones, puesto que, la simulación no ha comenzado y, por tanto, el procesado de un patrón en un componente, es de orden $log_2 P$, siendo P el número de patrones.

Al ser un bucle anidado estos órdenes se multiplican entre sí, y al tener que procesar P patrones todo debe multiplicarse por P quedando un orden final en la operación de procesado de patrones:

$$T_{procesado} = o(P \times \log_2 C \times \log_2 P) \tag{5.2}$$

En cada simulación el número de componentes es una constante, quedando la ecuación 5.2 como:

$$T_{procesado} = K_{setup} + A \times P \times \log_2 P \tag{5.3}$$

donde A engloba la dependencia con el número de componentes y K_{setup} es una constante de tiempo que hace referencia a la inicialización del motor de simulación. Se concluye por tanto que el orden del tiempo de ejecución encaja con los resultados empíricos: *O(nlog(n))*.

Para concluir el análisis, el aumento de tiempo en función de los patrones se ha representado por separado en la figura 5.42. Se han llegado a simular hasta un total de 150000 patrones en unos tiempos razonables. La verificación del comportamiento temporal de HALOTIS consiste en realizar el ajuste de las curvas de la figura 5.42 a la ecuación 5.2, pero haciendo constante el número de componentes C en cada una de ellas. El error en el cálculo del ajuste se presenta en la tabla 5.15, concluyendo que, al ser tan pequeño, el funcionamiento del algoritmo de procesado de patrones, de control del *netlist* y de la cola de eventos es el esperado,

por tanto, la implementación es correcta.

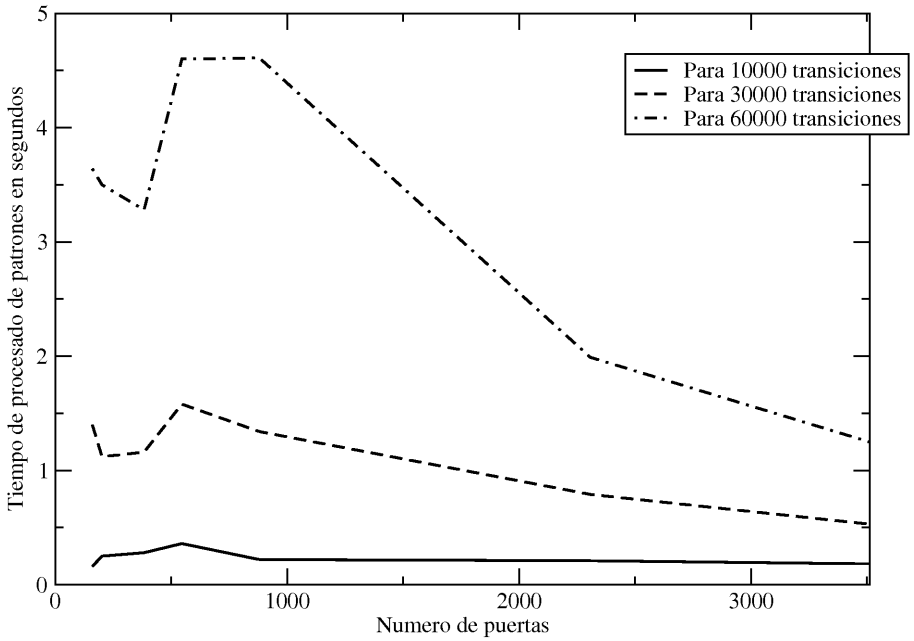


Figura 5.41. Tiempo de procesado de los patrones respecto al número de puertas.

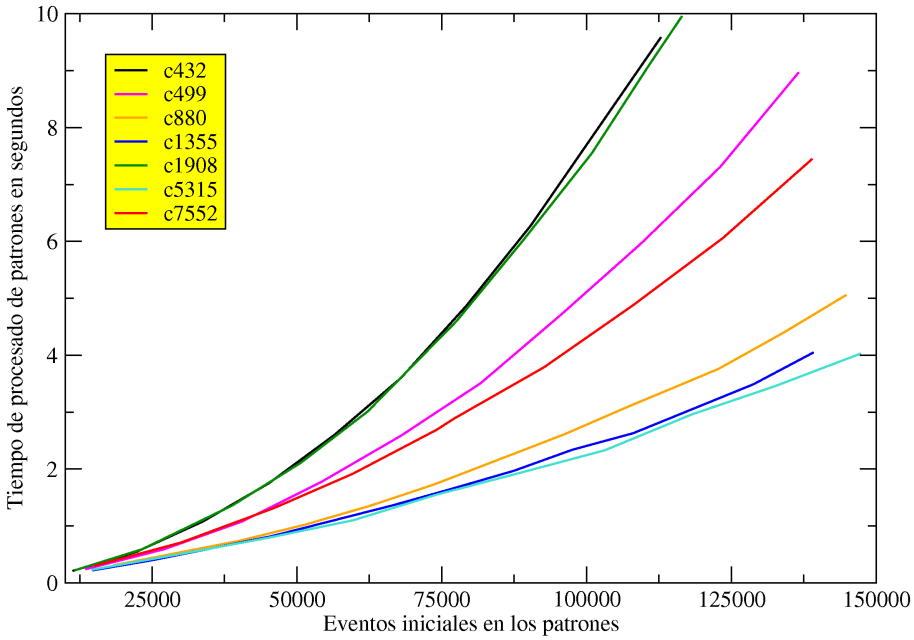


Figura 5.42. Tiempo de procesado de los patrones en los diferentes circuitos.

Circuito	Error de Ajuste
c432	0,0052%
c499	0,0093%
c880	0,0069%
c1355	0,0115%
c1908	0,0086%
c5315	0,0097%
c7552	0,0039%

Tabla 5.15. Ajuste del tiempo de procesado de patrones para cada circuito.

En el último análisis se miden los tiempos totales de simulación, comprobando de nuevo que la evolución sigue el orden de los algoritmos programados en HALOTIS. Una segunda aplicación de estas simulaciones es la medida de la velocidad de la herramienta. Formalmente, la velocidad se mide a partir de la productividad, es decir, tareas que es capaz de desarrollar por unidad de tiempo. Para una simulación guiada por eventos, se establece con medida válida de productividad el número de eventos simulado en cada segundo.

Como segundo objetivo en estas simulaciones se calcula la influencia en la productividad de HALOTIS (por tanto en la velocidad), del tamaño de los circuitos y del número de patrones a simular. Para todos los circuitos se han realizado simulaciones con más de 1 millón de eventos, resultando los tiempos de simulación mostrados en la figura 5.43. En todos los casos, la respuesta del motor de simulación es lineal frente al tiempo y, además, el hecho de que todas las rectas se superpongan corrobora que el número de componentes del circuito apenas influye en el tiempo de procesado de los eventos. Todas estas rectas tienen un buen coeficiente de correlación (tabla 5.16) y las pendientes presentan una variación de no más del 10%.

Las pendientes de la figura 5.43 muestran la productividad de HALOTIS, existiendo un descenso al aumentar el tamaño del circuito. La variación es muy pequeña y, además, no es lineal con el número de componentes, tiende a estabilizarse. Si se mide la productividad respecto al número de patrones se obtiene la representación de la figura 5.44, viéndose la escasa influencia en la productividad, que tiene la cantidad de estímulos iniciales, siendo además, independiente del tamaño del circuito.

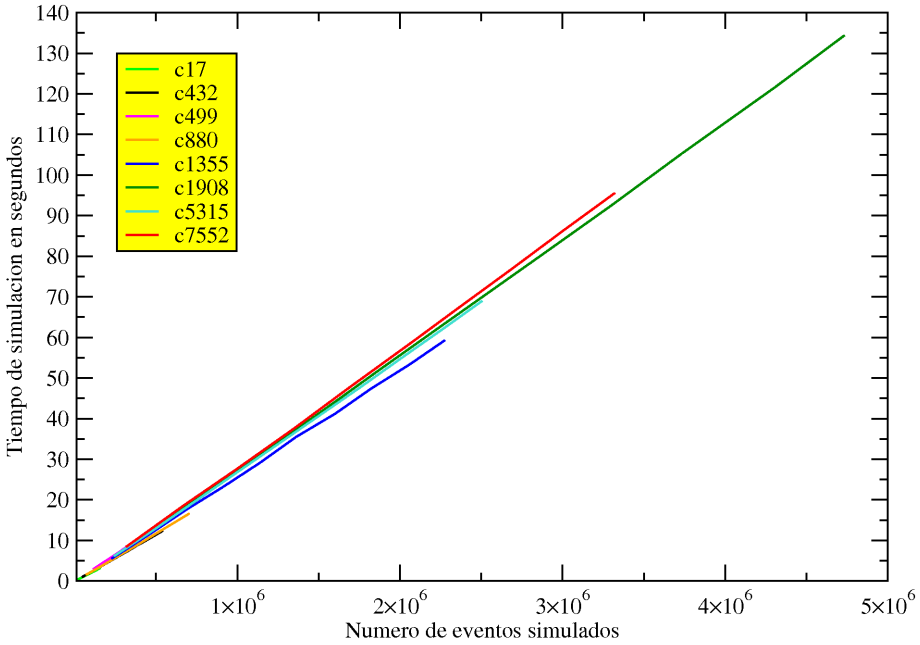


Figura 5.43. Tiempo de simulación en función del número de eventos.

Circuito	Coeficiente	Recta
c17	0,9999292	$y=-0{,}022583+2{,}0233\times10^{-5}x$
c432	0,9999824	$y=-0{,}10115+2{,}2985\times10^{-5}x$
c499	0,9999732	$y=-0{,}3302+2{,}7447\times10^{-5}x$
c880	0,9999537	$y=-0{,}216625+2{,}3818\times10^{-5}x$
c1355	0,9999422	$y=-0{,}53677+2{,}6223\times10^{-5}x$
c1908	0,9999697	$y=-1{,}4318+2{,}8579\times10^{-5}x$
c5315	0,9999678	$y=-0{,}891+2{,}7787\times10^{-5}x$
c7552	0,9999535	$y=-1{,}1658+2{,}9043\times10^{-5}x$

Tabla 5.16. Regresiones del tiempo de simulación frente a los eventos totales.

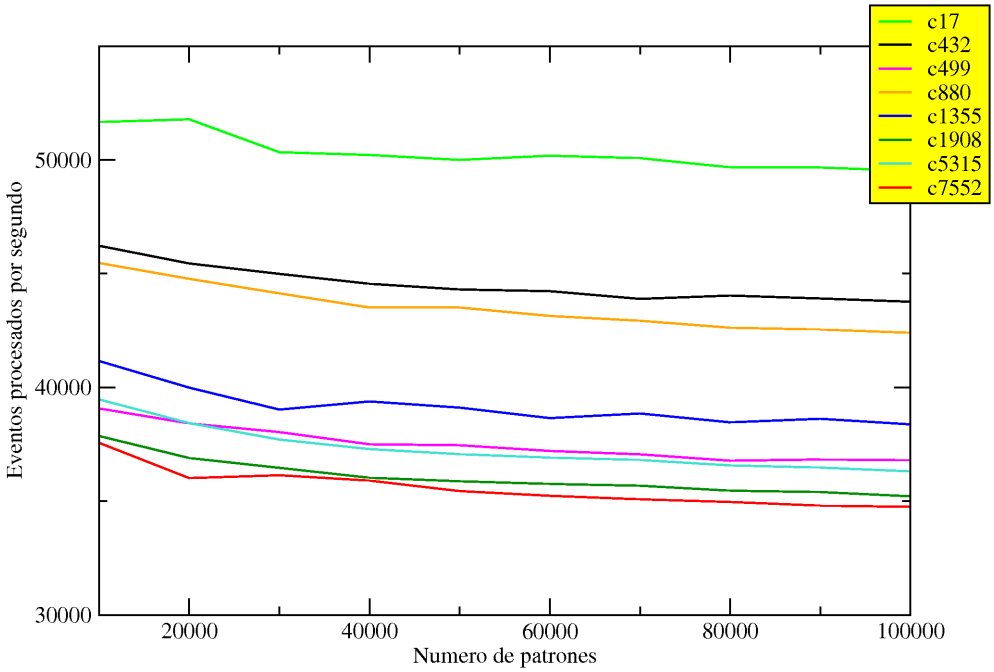


Figura 5.44. Productividad frente al número de patrones totales procesados.

Tras el análisis de esta sección, se concluye que todos los órdenes de tiempo de ejecución de los algoritmos implementados en HALOTIS, son menores del cuadrático, respecto al tamaño del circuito y a la cantidad de estímulos iniciales. Por tanto, se puede simular circuitos de gran tamaño siempre que se disponga de una máquina de la velocidad adecuada.

5.3.2. Utilización de memoria

Todo el análisis realizado en la sección anterior quedaría invalidado si no se consigue acotar el uso de la memoria del motor de simulación. En todo sistema operativo cuando la memoria utilizada por los procesos supera la cantidad de memoria física disponible, el sistema comienza a paginar, aumentando el tiempo de ejecución de cualquier proceso. En estas circunstancias, las medidas de tiempo de ejecución de los procesos son irreales.

Ha habido que cerciorarse que todas las ejecuciones anteriores cabían en la memoria física de la computadora sobre la que se hacían las pruebas. Se han tomado medidas del uso de memoria para todas las simulaciones anteriores, y como resumen, se muestran resultados de los tres circuitos mayores en la tabla 5.17.

La evolución del consumo de memoria durante el proceso de simulación se

representa en la figura 5.45, en la que se han separado tres zonas, dos crecientes correspondientes a la lectura del *netlist* y de patrones. Tras esto, el motor de simulación tiene todos los eventos iniciales en la cola, y por tanto es cuando empieza a decrecer el uso, al ir procesando cada evento y eliminándolo. El valor del máximo de memoria es fácil de obtener, es la suma del tamaño del *netlist* y los eventos iniciales resumiéndose como sigue:

$$M_{máx} = (N_{puertas} \times M_{puerta}) + (N_{conexiones} \times M_{conexión}) + (N_{eventos} \times M_{evento}) \qquad (5.4)$$

donde M_i son constantes conocidas, concretamente es el tamaño que ocupa una instancia de un objeto en memoria (puerta conexión o evento, según el valor de i), siendo del orden de 20-50 bytes. Usando la ecuación 5.4 el uso de memoria es fácil de aproximar, a partir del *netlist* y de los patrones.

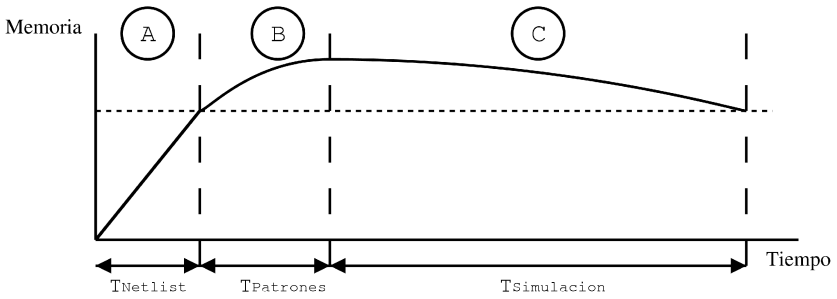


Figura 5.45. Evolución del consumo de memoria del motor de simulación.

Circuito	Eventos iniciales	Máxima memoria usada
c1908	77600	8,3Mbytes
c5315	147300	14,5Mbytes
c7552	154500	18,8Mbytes

Tabla 5.17. Picos de utilización de memoria durante el proceso de simulación.

No todos los circuitos encajan en este comportamiento. Existen simulaciones no convergentes, es decir, simulaciones en las que la cola de eventos no nunca llega a vaciarse, generalmente por la existencia de realimentaciones. Un ejemplo de esta situación es el oscilador en anillo presentado en la sección 5.1.3. Este tipo de simulaciones ha permitido realizar una prueba de estabilidad del consumo de memoria del motor de simulación. Consiste en dejar la simulación no acotada en el tiempo, de forma que nunca termina. Tras varias horas se compara el uso total de memoria, concluyéndose que al no aumentar, no existen pérdidas de memoria en el algoritmo de simulación.

Capítulo 6. Conclusiones

En octubre del año 2000 uno de los codirectores de este trabajo defendió su tesis doctoral, trabajo que presentaba muy exhaustivamente el modelo DDM. A partir de ese momento se planteó la necesidad de desarrollar un simulador lógico temporal que no solo incorporara el modelo DDM, sino que además desarrollaran las aplicaciones más importantes que se derivan de su empleo en la Simulación Lógica. Con esta idea comenzó este trabajo y, a lo largo de estos años, se ha ido enriqueciendo con otros aspectos de interés culminando en el trabajo presentado en esta Tesis doctoral. A continuación se enumeran los logros más importantes alcanzados con este trabajo:

1. Se ha llevado a cabo un estudio de los fundamentos de la verificación temporal centrándose en la simulación lógico-temporal de circuitos digitales. En este estudio se han analizado las causas del grado de precisión de los resultados de la simulación lógica temporal y las características informáticas de estas herramientas CAD.
2. Se ha analizado detalladamente el modelo de retraso DDM así como el algoritmo de efecto inercial asociado al mismo. Con este análisis se ha llegado a establecer el conjunto de ecuaciones y parámetros posteriormente incorporados en el simulador lógico desarrollado.

3. Se ha verificado la validez del modelo DDM en un estilo de celdas distintas de las estándar CMOS denominadas CBL (*Current Balanced Logic*). Este nuevo tipo de celdas es especialmente adecuado en el diseño de la parte digital de circuitos mixtos. El estudio del comportamiento dinámico de dichas celdas ha mostrado como el modelo DDM es válido para el comportamiento temporal exhibido por estas celdas.

4. Se ha propuesto un nuevo modelo de intensidad para las celdas lógicas CMOS y su aplicación en la Simulación Lógica Temporal. Se ha estudiado la validez del modelo en las celdas CMOS, desarrollándose un proceso de caracterización de parámetros para el mismo.

5. Como objetivo principal de la Tesis se ha desarrollado un entorno de Simulación Lógica Temporal al que se ha denominado HALOTIS. HALOTIS se ha desarrollado siguiendo la metodología UML. Las características más importantes de HALOTIS son la inclusión del modelo DDM y el algoritmo para el cálculo de la intensidad consumida en los circuitos digitales, empleando para ello, el modelo de intensidad propuesto.

6. Se han obtenido un conjunto de resultados de simulación que muestran las mejoras de HALOTIS comparadas con otras herramientas de simulación lógica y con el simulador analógico HSPICE. Los resultados podemos clasificarlos en dos grupos: los referentes a la precisión en cuanto a la forma de onda de la señal y, los referentes a la estimación del consumo de potencia e intensidad.

   a) Respecto a los primeros, se han llevado a cabo simulaciones que muestran el buen funcionamiento de HALOTIS así como el alto grado de precisión conseguido al utilizar el modelo DDM. También, se ha comprobado la precisión de los resultados de simulación cuando se simulan circuitos lógicos diseñados con celdas CBL.

   b) Respecto a los segundos, se ha dotado a HALOTIS de un mecanismo para medir la actividad de conmutación de los circuitos simulados. Empleando el *benchmark* de circuitos combinacionales ISCAS'85 se ha comprobado como la medida de la actividad de conmutación mediante simulación lógica alcanza cotas suficientes de precisión y fiabilidad sólo cuando el modelo de retraso incluye el efecto de degradación. Además, se ha incorporado en HALOTIS un algoritmo de cálculo de intensidad que, unido al modelo de intensidad desarrollado dota a HALOTIS de la capacidad de generar curvas de intensidad en la simulación de los circuitos digitales. Nuevamente, los resultados muestran la necesidad de incluir la degradación, en el modelo de retraso empleado, para obtener resultados fiables.

7. Se ha llevado a cabo un estudio sobre la eficiencia de la herramienta en

cuanto a los recursos computacionales empleados en la simulación de circuitos digitales. Los resultados indican que HALOTIS se mueve dentro de los parámetros de velocidad de computación y uso de memoria típica de las herramientas de Simulación Lógica.

Anexo 1. Descripción de las librerías de celdas

Librería AMS 0.6μm utilizada en la versión inicial de HALOTIS (versión 0), la cual, estaba escrita directamente en código C++ e incrustada en el motor de simulación:

```
#include"halotiscells.h"
#include"../gate.h"
#include"../error.h"
#include<ctype.h>
#include<assert.h>
////////////////////////////////////////////////////////////
// Singleton construction
HALOTIS_Cells* HALOTIS_Cells::instance=NULL;
HALOTIS_Cells::HALOTIS_Cells()
{
  LibName="Halotis standart CMOS Gates v1.0";
  TimeScale=SLTAP_Time::ps;
  assert(TimeScale==SLTAP_Time::GetGlobalTimeScale());
}

HALOTIS_Cells::~HALOTIS_Cells()
{

}
////////////////////////////////////////////////////////////
SLTAP_Gate *HALOTIS_Cells::CreateGate(string type,string name)
{
  if(type=="inv")
    return new HALOTIS_Cell_inv(name);
  else if(type=="nand2")
    return new HALOTIS_Cell_nand2(name);
  else if(type=="nor2")
    return new HALOTIS_Cell_nor2(name);
  else if(type=="nand3")
    return new HALOTIS_Cell_nand3(name);
  else if(type=="nor3")
    return new HALOTIS_Cell_nor3(name);
  else
    return NULL;
}

////////////////////////////////////////////////////////////
//
// Library cells
//

////////////// INV GATE ////////////////////////////
SLTAP_DDMModel HALOTIS_Cell_inv::model;
char *HALOTIS_Cell_inv::wires[]={"a","q"};
HALOTIS_Cell_inv::HALOTIS_Cell_inv(string
name):SLTAP_Gate(name,1,wires)
{
```

```
  model.threshold=2.3718;
  model.cin=19.56;
  model.coutf=0.795189*model.cin;
  model.coutr=0.789113*model.cin;
  model.vtn=1.1568;
  model.vtp=1.6805;
  model.cmf=6.22;
  model.cmr=3.51;
  model.k1f=24.65;
  model.k1r=24.17;
  model.k2f=0.0;
  model.k2r=0.0;
  model.k3f=1;
  model.k3r=1;
  model.taucof=1.0/0.7;
  model.taucor=1.0/0.7;
  model.af=95.7;
  model.bf=4.903;
  model.cf=1.77;
  model.ar=41.27;
  model.br=3.02;
  model.cr=1.61;

  SetInputModel(0,&model);
}

////////////////// NAND2 GATE ////////////////////
SLTAP_DDMModel HALOTIS_Cell_nand2::model[2];
char *HALOTIS_Cell_nand2::wires[]={"a","b","q"};
HALOTIS_Cell_nand2::HALOTIS_Cell_nand2(string
name):SLTAP_Gate(name,2,wires)
{
  model[0].threshold=2.1072;
  model[0].cin=6.5196;
  model[0].coutf=1.48201*model[0].cin;
  model[0].coutr=1.35292*model[0].cin;
  model[0].vtn=1.01118;
  model[0].vtp=1.91384;
  model[0].cmf=0.296319*model[0].cin;
  model[0].cmr=0.192156*model[0].cin;
  model[0].k1f=24.6781;
  model[0].k1r=34.2615;
  model[0].k2f=0;
  model[0].k2r=0;
  model[0].k3f=1;
  model[0].k3r=1;
  model[0].taucof=1.0/0.7;
  model[0].taucor=1.0/0.7;
  model[0].af=208.623;
  model[0].bf=20.1537;
  model[0].cf=1.32955;
  model[0].ar=90.5916;
  model[0].br=9.84602;
  model[0].cr=1.98706;

  model[1].threshold=1.9165  ;
  model[1].cin=6.5196;
  model[1].coutf=1.56915*model[1].cin;
  model[1].coutr=2.15318*model[1].cin;
  model[1].vtn=0.25008;
  model[1].vtp=2.58311;
  model[1].cmf=0.591444*model[1].cin;
  model[1].cmr=0.553764*model[1].cin;
  model[1].k1f=24.5585;
```

```
  model[1].k1r=34.31;
  model[1].k2f=0;
  model[1].k2r=0;
  model[1].k3f=1;
  model[1].k3r=1;
  model[1].taucof=1.0/0.7;
  model[1].taucor=1.0/0.7;
  model[1].af=252.685;
  model[1].bf=20.2306;
  model[1].cf=1.13477;
  model[1].ar=126.194;
  model[1].br=11.4375;
  model[1].cr=2.54086;

  SetInputModel(0,&model[0]);
  SetInputModel(1,&model[1]);

}

//////////////////// NOR2 GATE ////////////////////////////
SLTAP_DDMModel HALOTIS_Cell_nor2::model[2];
char *HALOTIS_Cell_nor2::wires[]={"a","b","q"};

HALOTIS_Cell_nor2::HALOTIS_Cell_nor2(string
name):SLTAP_Gate(name,2,wires)
{
  model[0].threshold=2.4284;
  model[0].cin=29.34;
  model[0].coutf=0.93695*model[0].cin;
  model[0].coutr=1.18121*model[0].cin;
  model[0].vtn=1.25046;
  model[0].vtp=1.45493;
  model[0].cmf=0.395916;
  model[0].cmr=0.173535;
  model[0].k1f=36.9682;
  model[0].k1r=38.3439;
  model[0].k2f=0;
  model[0].k2r=0;
  model[0].k3f=1;
  model[0].k3r=1;
  model[0].taucof=1.0/0.7;
  model[0].taucor=1.0/0.7;
  model[0].af=250.258;
  model[0].bf=4.84526;
  model[0].cf=1.91189;
  model[0].ar=64.0181;
  model[0].br=2.85134;
  model[0].cr=1.48774;

  model[1].threshold=2.6894;
  model[1].cin=29.34;
  model[1].coutf=2.07841*model[1].cin;
  model[1].coutr=1.19362*model[1].cin;
  model[1].vtn=1.38201;
  model[1].vtp=0.66668;
  model[1].cmf=0.109628*model[1].cin;
  model[1].cmr=0.428121*model[1].cin;
  model[1].k1f=37.6041;
  model[1].k1r=39.024;
  model[1].k2f=0;
  model[1].k2r=0;
  model[1].k3f=1;
  model[1].k3r=1;
  model[1].taucof=1.0/0.7;
```

```
  model[1].taucor=1.0/0.7;
  model[1].af=272.206;
  model[1].bf=5.26297;
  model[1].cf=2.37859;
  model[1].ar=142.057;
  model[1].br=2.43184;
  model[1].cr=1.45051;

  SetInputModel(0,&model[0]);
  SetInputModel(1,&model[1]);

}

////////////// NAND3 GATE ///////////////////////////////////
SLTAP_DDMModel HALOTIS_Cell_nand3::model[3];
char *HALOTIS_Cell_nand3::wires[]={"a","b","c","q"};

HALOTIS_Cell_nand3::HALOTIS_Cell_nand3(string
name):SLTAP_Gate(name,3,wires)
{
  model[0].threshold=2.1499  ;
  model[0].cin=8.964;
  model[0].coutf=1.71892*model[0].cin;
  model[0].coutr=1.5766*model[0].cin;
  model[0].vtn=0.94751;
  model[0].vtp=1.85381;
  model[0].cmf=0.248145*model[0].cin;
  model[0].cmr=0.23745*model[0].cin;
  model[0].k1f=30.1112;
  model[0].k1r=37.0899;
  model[0].k2f=0;
  model[0].k2r=0;
  model[0].k3f=1;
  model[0].k3r=1;
  model[0].taucof=1.0/0.7;
  model[0].taucor=1.0/0.7;
  model[0].af=274.293;
  model[0].bf=15.66;
  model[0].cf=1.3775;
  model[0].ar=176.654;
  model[0].br=10.3423;
  model[0].cr=1.85757;

  model[1].threshold=2.0037  ;
  model[1].cin=8.964;
  model[1].coutf=1.96578*model[1].cin;
  model[1].coutr=2.3889*model[1].cin;
  model[1].vtn=0.32133;
  model[1].vtp=1.97546;
  model[1].cmf=0.542558*model[1].cin;
  model[1].cmr=0.114375*model[1].cin;
  model[1].k1f=29.7051;
  model[1].k1r=37.1177;
  model[1].k2f=0;
  model[1].k2r=0;
  model[1].k3f=1;
  model[1].k3r=1;
  model[1].taucof=1.0/0.7;
  model[1].taucor=1.0/0.7;
  model[1].af=312.815;
  model[1].bf=15.6922;
  model[1].cf=1.2401;
  model[1].ar=200.834;
  model[1].br=11.2297;
```

```
  model[1].cr=2.49735;

  model[2].threshold=1.8212  ;
  model[2].cin=8.964;
  model[2].coutf=1.97213*model[2].cin;
  model[2].coutr=3.1201*model[2].cin;
  model[2].vtn=-0.2117;
  model[2].vtp=2.11813;
  model[2].cmf=0.707119*model[2].cin;
  model[2].cmr=-0.0817936*model[2].cin;
  model[2].k1f=29.7341;
  model[2].k1r=37.4536;
  model[2].k2f=0;
  model[2].k2r=0;
  model[2].k3f=1;
  model[2].k3r=1;
  model[2].taucof=1.0/0.7;
  model[2].taucor=1.0/0.7;
  model[2].af=392.024;
  model[2].bf=15.307;
  model[2].cf=1.01648;
  model[2].ar=244.442;
  model[2].br=11.8572;
  model[2].cr=2.90585;

  SetInputModel(0,&model[0]);
  SetInputModel(1,&model[1]);
  SetInputModel(2,&model[2]);
}

////////////// NOR3 GATE ///////////////////////////////////
SLTAP_DDMModel HALOTIS_Cell_nor3::model[3];
char *HALOTIS_Cell_nor3::wires[]={"a","b","c","q"};

HALOTIS_Cell_nor3::HALOTIS_Cell_nor3(string name) :
SLTAP_Gate(name,3,wires)
{
  model[0].threshold=2.4660  ;
  model[0].cin=39.12;
  model[0].coutf=1.00727*model[0].cin;
  model[0].coutr=1.01859*model[0].cin;
  model[0].vtn=1.33604;
  model[0].vtp=1.37108;
  model[0].cmf=0.421664*model[0].cin;
  model[0].cmr=0.109228*model[0].cin;
  model[0].k1f=49.2928;
  model[0].k1r=53.6968;
  model[0].k2f=0;
  model[0].k2r=0;
  model[0].k3f=1;
  model[0].k3r=1;
  model[0].taucof=1.0/0.7;
  model[0].taucor=1.0/0.7;
  model[0].af=439.88;
  model[0].bf=5.34891;
  model[0].cf=1.85453;
  model[0].ar=90.5908;
  model[0].br=2.91873;
  model[0].cr=1.57227;

  model[1].threshold=2.6501  ;
  model[1].cin=39.12;
  model[1].coutf=2.28268*model[1].cin;
  model[1].coutr=1.44415*model[1].cin;
```

```
  model[1].vtn=1.40156;
  model[1].vtp=0.68716;
  model[1].cmf=0.206168*model[1].cin;
  model[1].cmr=0.427643*model[1].cin;
  model[1].k1f=49.9255;
  model[1].k1r=53.7282;
  model[1].k2f=0;
  model[1].k2r=0;
  model[1].k3f=1;
  model[1].k3r=1;
  model[1].taucof=1.0/0.7;
  model[1].taucor=1.0/0.7;
  model[1].af=484.18;
  model[1].bf=5.48288;
  model[1].cf=2.37431;
  model[1].ar=144.228;
  model[1].br=2.79995;
  model[1].cr=1.46403;

  model[2].threshold=2.8815  ;
  model[2].cin=39.12;
  model[2].coutf=3.05201*model[2].cin;
  model[2].coutr=1.48856*model[2].cin;
  model[2].vtn=1.55258;
  model[2].vtp=0.18365;
  model[2].cmf=-0.0705008*model[2].cin;
  model[2].cmr=0.653006*model[2].cin;
  model[2].k1f=51.7082;
  model[2].k1r=53.75;
  model[2].k2f=0;
  model[2].k2r=0;
  model[2].k3f=1;
  model[2].k3r=1;
  model[2].taucof=1.0/0.7;
  model[2].taucor=1.0/0.7;
  model[2].af=505.569;
  model[2].bf=5.73299;
  model[2].cf=2.66644;
  model[2].ar=317.492;
  model[2].br=2.55793;
  model[2].cr=1.30738;

  SetInputModel(0,&model[0]);
  SetInputModel(1,&model[1]);
  SetInputModel(2,&model[2]);
```

Librería AMS 0.35μm en formato de texto correspondiente a la versión 1 de HALOTIS:

```
/*----------------------------------------
 AMS_035
  ----------------------------------------*/

#define LMIN 0.3e-6
#define COX 4.480461e-3
#define VDD 3.3
```

```
 BEGIN;
 IDD_MODEL();

 STATIC_PARAMETER(Vddh,3.3);
 STATIC_PARAMETER(Vddl,0);
 STATIC_PARAMETER(Vtn,0.81804);
 STATIC_PARAMETER(Vtp,2.7622);

 /*----------------------------------------
 AMS_035 BU1
 Last update: 13-5-04
 ----------------------------------------*/
BEGIN_CELL("bu1",1,1);
     INPUT_NAME(0,"a");
     OUT_NAME(0,"q");

     EXPRESION(0,INPUT(0));

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+2e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+2e-6)*LMIN*COX);
     PARAMETER(Af,18.0025e-12);
     PARAMETER(Bf,5.7802e3);
     PARAMETER(Cf,1.40305);
     PARAMETER(Ar,44.0191e-12);
     PARAMETER(Br,12.2354e3);
     PARAMETER(Cr,2.07436);
     PARAMETER(Dr,3.36427e3);
     PARAMETER(Er,0.135381);
     PARAMETER(Fr,120.062e-12);
     PARAMETER(Df,3.81402e3);
     PARAMETER(Ef,0.0351446);
     PARAMETER(Ff,121.492e-12);
     PARAMETER(Gr,9974.26);
     PARAMETER(Hr,8.6876e-11);
     PARAMETER(Gf,7011.94);
     PARAMETER(Hf,6.9397e-11);
     ASSIGN_MODEL(0,0);

 END_CELL;

/*----------------------------------------
 AMS_035 IN1
 Last update: 1-3-04
 ----------------------------------------*/
 BEGIN_CELL("in1",1,1);
     INPUT_NAME(0,"a");
     OUT_NAME(0,"q");

     EXPRESION(0,NOT(INPUT(0)));

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+2e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+2e-6)*LMIN*COX);
     PARAMETER(Af,94.1408e-12);
     PARAMETER(Bf,10.1678e3);
     PARAMETER(Cf,0.926382);
     PARAMETER(Ar,32.005e-12);
     PARAMETER(Br,5.24642e3);
     PARAMETER(Cr,1.05815);
     PARAMETER(Dr,4.33892e3);
     PARAMETER(Er,0.176655);
     PARAMETER(Fr,30.4103e-12);
```

```
      PARAMETER(Df,3.64233e3);
      PARAMETER(Ef,0.0946134);
      PARAMETER(Ff,32.1833e-12);
      PARAMETER(Gr,9916.13);
      PARAMETER(Hr,8.86367e-11);
      PARAMETER(Gf,7005.88);
      PARAMETER(Hf,7.01589e-11);
      ASSIGN_MODEL(0,0);

 END_CELL;

/*-------------------------------------------
      NA2 AMS_035
      Last Update: 9-4-04
-------------------------------------------*/
      BEGIN_CELL("na2",2,1);

      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      OUT_NAME(0,"q");

      // Expresi□n de la salida
      EXPRESION(0,NAND(INPUT(0),INPUT(1)));

      // Parametros de la entrada 0
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(2e-6+2e-6)*LMIN*COX);
      PARAMETER(Cinf,(2e-6+2e-6)*LMIN*COX);
      PARAMETER(Af,177.033e-12);
      PARAMETER(Bf,10.3333e3);
      PARAMETER(Cf,0.882377);
      PARAMETER(Ar,64.6769e-12);
      PARAMETER(Br,4.41094e3);
      PARAMETER(Cr,1.31169);
      PARAMETER(Dr,4.20692e3);
      PARAMETER(Er,0.201717);
      PARAMETER(Fr,55.4206e-12);
      PARAMETER(Df,2.77464e3);
      PARAMETER(Ef,0.0825482);
      PARAMETER(Ff,42.3562e-12);
      PARAMETER(Gr,10340.8);
      PARAMETER(Hr,1.33747e-10);
      PARAMETER(Gf,6054.05);
      PARAMETER(Hf,8.64447e-11);
      // Assign Model (out,in)
      ASSIGN_MODEL(0,0);

      // Modelo para la segunda entrada
      PARAMETER(Af,239.906e-12);
      PARAMETER(Bf,9.48001e3);
      PARAMETER(Cf,0.816024);
      PARAMETER(Ar,86.6308e-12);
      PARAMETER(Br,5.17971e3);
      PARAMETER(Cr,1.63747);
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(2e-6+2e-6)*LMIN*COX);
      PARAMETER(Cinf,(2e-6+2e-6)*LMIN*COX);
      PARAMETER(Dr,4.15532e3);
      PARAMETER(Er,0.213299);
      PARAMETER(Fr,93.0931e-12);
      PARAMETER(Df,2.7454e3);
      PARAMETER(Ef,0.0184577);
      PARAMETER(Ff,61.0572e-12);
      PARAMETER(Gr,10319.5);
```

```
      PARAMETER(Hr,2.15881e-10);
      PARAMETER(Gf,6012.03);
      PARAMETER(Hf,8.8294e-11);

      ASSIGN_MODEL(0,1);

 END_CELL;

/*-------------------------------------------
      AND2 AMS_035
      Last Update: 13-4-04
      NOTAS: Una entrada con parametros 0
-------------------------------------------*/
BEGIN_CELL("and2",2,1);

      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      OUT_NAME(0,"q");

      // Expresi?n de la salida
      EXPRESION(0,AND(INPUT(0),INPUT(1)));

      // Parametros de la entrada 0
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,113.786e-12);
      PARAMETER(Bf,9.53203e3);
      PARAMETER(Cf,1.7397);
      PARAMETER(Ar,251.418e-12);
      PARAMETER(Br,-3.45613e3);
      PARAMETER(Cr,1.13628);
      PARAMETER(Dr,4.33608e3);
      PARAMETER(Er,0.193527);
      PARAMETER(Fr,177.902e-12);
      PARAMETER(Df,4.22064e3);
      PARAMETER(Ef,0.0335817);
      PARAMETER(Ff,152.826e-12);
      PARAMETER(Gr,9820.06);
      PARAMETER(Hr,1.06063e-10);
      PARAMETER(Gf,7468.03);
      PARAMETER(Hf,9.40487e-11);
      // Assign Model (out,in)
      ASSIGN_MODEL(0,0);

      // Modelo para la segunda entrada
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0); // No degradation effect
      PARAMETER(Bf,0);
      PARAMETER(Cf,0);
      PARAMETER(Ar,299.929e-12);
      PARAMETER(Br,-4.81925e3);
      PARAMETER(Cr,0.955825);
      PARAMETER(Dr,4.67356e3);
      PARAMETER(Er,0.211201);
      PARAMETER(Fr,225.425e-12);
      PARAMETER(Df,4.39545e3);
      PARAMETER(Ef,-0.00360159);
      PARAMETER(Ff,168.564e-12);
      PARAMETER(Gr,9816.81);
      PARAMETER(Hr,1.06125e-10);
      PARAMETER(Gf,7333.44);
```

```
      PARAMETER(Hf,1.12689e-10);
      ASSIGN_MODEL(0,1);

 END_CELL;

  /*-------------------------------------------
      NO2 AMS_035
      Last Update: 13-4-04
-------------------------------------------*/
BEGIN_CELL("no2",2,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      OUT_NAME(0,"q");

      EXPRESION(0,NOR(INPUT(0),INPUT(1)));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Af,280.795e-12);
      PARAMETER(Bf,9.81578e3);
      PARAMETER(Cf,1.47707);
      PARAMETER(Ar,583.966e-12);
      PARAMETER(Br,-5.73553e3);
      PARAMETER(Cr,0.699558);
      PARAMETER(Dr,4.0707e3);
      PARAMETER(Er,0.0871037);
      PARAMETER(Fr,99.6883e-12);
      PARAMETER(Df,3.60752e3);
      PARAMETER(Ef,0.134644);
      PARAMETER(Ff,114.916e-12);
      PARAMETER(Gr,9612.79);
      PARAMETER(Hr,1.80261e-10);
      PARAMETER(Gf,13265.9);
      PARAMETER(Hf,4.20325e-10);
      // Assign Model (out,in)
      ASSIGN_MODEL(0,0);

      // Modelo para la entrada 1
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Af,260.776e-12);
      PARAMETER(Bf,9.35901e3);
      PARAMETER(Cf,1.29811);
      PARAMETER(Ar,64.7805e-12);
      PARAMETER(Br,5.37045e3);
      PARAMETER(Cr,1.05284);
      PARAMETER(Dr,3.94827e3);
      PARAMETER(Er,0.160375);
      PARAMETER(Fr,43.9069e-12);
      PARAMETER(Df,3.47118e3);
      PARAMETER(Ef,0.12462);
      PARAMETER(Ff,59.7304e-12);
      PARAMETER(Gr,10216);
      PARAMETER(Hr,1.56534e-10);
      PARAMETER(Gf,14838.6);
      PARAMETER(Hf,1.78293e-10);
      ASSIGN_MODEL(0,1);

 END_CELL;

/* ----------------------------------------
 AMS_035 AND3
```

```
 Last update: 13-5-04
 ----------------------------------------*/
BEGIN_CELL("and3",3,1);
     INPUT_NAME(0,"a");
     INPUT_NAME(1,"b");
     INPUT_NAME(2,"c");
     OUT_NAME(0,"q");

     EXPRESION(0,AND(AND(INPUT(0),INPUT(1)),INPUT(2)));

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1.5e-6+1e-6)*LMIN*COX);
     PARAMETER(Cinf,(1.5e-6+1e-6)*LMIN*COX);
     PARAMETER(Af,154.013e-12);
     PARAMETER(Bf,7.7502e3);
     PARAMETER(Cf,1.75096);
     PARAMETER(Ar,342.589e-12);
     PARAMETER(Br,-3.78526e3);
     PARAMETER(Cr,0.911094);
     PARAMETER(Dr,4.33509e3);
     PARAMETER(Er,0.204248);
     PARAMETER(Fr,213.7e-12);
     PARAMETER(Df,4.31819e3);
     PARAMETER(Ef,0.0102685);
     PARAMETER(Ff,174.045e-12);
     PARAMETER(Gr,10006.6);
     PARAMETER(Hr,1.16891e-10);
     PARAMETER(Gf,7544.86);
     PARAMETER(Hf,1.06065e-10);
     ASSIGN_MODEL(0,0);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1.5e-6+1e-6)*LMIN*COX);
     PARAMETER(Cinf,(1.5e-6+1e-6)*LMIN*COX);
     PARAMETER(Af,0e-12);
     PARAMETER(Bf,0e3);
     PARAMETER(Cf,0);
     PARAMETER(Ar,413.139e-12);
     PARAMETER(Br,-6.25836e3);
     PARAMETER(Cr,0.819249);
     PARAMETER(Dr,4.691e3);
     PARAMETER(Er,0.219112);
     PARAMETER(Fr,282.925e-12);
     PARAMETER(Df,4.44964e3);
     PARAMETER(Ef,-0.0133218);
     PARAMETER(Ff,202.265e-12);
     PARAMETER(Gr,10000.8);
     PARAMETER(Hr, 1.17299e-10);
     PARAMETER(Gf,7415);
     PARAMETER(Hf,1.32687e-10);
     ASSIGN_MODEL(0,1);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1.5e-6+1e-6)*LMIN*COX);
     PARAMETER(Cinf,(1.5e-6+1e-6)*LMIN*COX);
     PARAMETER(Af,0e-12);
     PARAMETER(Bf,0e3);
     PARAMETER(Cf,0);
     PARAMETER(Ar, 588.363e-12);
     PARAMETER(Br,-11.1797e3);
     PARAMETER(Cr,0.749395);
     PARAMETER(Dr,5.0562e3);
     PARAMETER(Er,0.232544);
     PARAMETER(Fr,338.684e-12);
```

```
      PARAMETER(Df,4.5151e3);
      PARAMETER(Ef,-0.0457065);
      PARAMETER(Ff,219.001e-12);
      PARAMETER(Gr, 9993.7);
      PARAMETER(Hr, 1.17387e-10);
      PARAMETER(Gf,7566.72);
      PARAMETER(Hf,1.57151e-10);
      ASSIGN_MODEL(0,2);

END_CELL;

 /* ----------------------------------------
 AMS_035 AND4
 Last update: 15-5-04
 ----------------------------------------*/
BEGIN_CELL("and4",4,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      INPUT_NAME(2,"c");
      INPUT_NAME(3,"d");

      OUT_NAME(0,"q");

      EXPRESION(0,AND(AND(INPUT(0),INPUT(1)),AND(INPUT(2),INPUT(3))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,231.14e-12);
      PARAMETER(Br, 0.418437e3);
      PARAMETER(Cr,1.19788);
      PARAMETER(Dr,3.70665e3);
      PARAMETER(Er,0.204958);
      PARAMETER(Fr,391.627e-12);
      PARAMETER(Df,3.92893e3);
      PARAMETER(Ef,-0.0374726);
      PARAMETER(Ff,264.838e-12);
      PARAMETER(Gr,9156.62);
      PARAMETER(Hr,2.06081e-10);
      PARAMETER(Gf, 6819.64);
      PARAMETER(Hf,2.80294e-10 );
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,220.458e-12);
      PARAMETER(Br,1.33824e3);
      PARAMETER(Cr, 1.30545);
      PARAMETER(Dr,3.66028e3);
      PARAMETER(Er,0.185817);
      PARAMETER(Fr,337.537e-12);
      PARAMETER(Df,3.89823e3);
      PARAMETER(Ef,-0.00128473);
      PARAMETER(Ff,248.719e-12);
      PARAMETER(Gr, 9156.75);
      PARAMETER(Hr,2.06049e-10);
      PARAMETER(Gf,7003.16);
```

```
    PARAMETER(Hf,2.66137e-10);
    ASSIGN_MODEL(0,1);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,0e-12);
    PARAMETER(Bf,0e3);
    PARAMETER(Cf,0);
    PARAMETER(Ar, 240.478e-12);
    PARAMETER(Br, -1.91325e3);
    PARAMETER(Cr, 0.96696);
    PARAMETER(Dr,3.98309e3);
    PARAMETER(Er,0.207326);
    PARAMETER(Fr,319.864e-12);
    PARAMETER(Df,3.79036e3);
    PARAMETER(Ef,-0.0190383);
    PARAMETER(Ff,203.396e-12);
    PARAMETER(Gr,9967.09);
    PARAMETER(Hr,1.79879e-10);
    PARAMETER(Gf, 7183.15);
    PARAMETER(Hf, 1.54347e-10);
    ASSIGN_MODEL(0,2);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,0e-12);
    PARAMETER(Bf,0e3);
    PARAMETER(Cf,0);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,3.835e3);
    PARAMETER(Er,0.189598);
    PARAMETER(Fr,268.188e-12);
    PARAMETER(Df,3.67249e3);
    PARAMETER(Ef,0.0173518);
    PARAMETER(Ff,188.369e-12);
    PARAMETER(Gr,9967);
    PARAMETER(Hr,1.79851e-10);
    PARAMETER(Gf,7310.35);
    PARAMETER(Hf,1.39129e-10);
    ASSIGN_MODEL(0,3);

END_CELL;
/* ----------------------------------------
AMS_035 AND8
Last update: 15-5-04
----------------------------------------*/
BEGIN_CELL("and8",8,1);
    INPUT_NAME(0,"a");
    INPUT_NAME(1,"b");
    INPUT_NAME(2,"c");
    INPUT_NAME(3,"d");
    INPUT_NAME(4,"e");
    INPUT_NAME(5,"f");
    INPUT_NAME(6,"g");
    INPUT_NAME(7,"h");

    OUT_NAME(0,"q");
```

```
EXPRESION(0,AND(AND(AND(INPUT(0),INPUT(1)),AND(INPUT(2),INPUT(3))),AN
D(AND(INPUT(4),INPUT(5)),AND(INPUT(6),INPUT(7)))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,619.078e-12);
      PARAMETER(Br, -2.23056e3);
      PARAMETER(Cr, 1.5489);
      PARAMETER(Dr,3.74489e3);
      PARAMETER(Er,0.198267);
      PARAMETER(Fr,560.363e-12);
      PARAMETER(Df,3.90749e3);
      PARAMETER(Ef,-0.032934);
      PARAMETER(Ff,385.044e-12);
      PARAMETER(Gr,8787.24);
      PARAMETER(Hr, 3.62419e-10);
      PARAMETER(Gf,7397.89);
      PARAMETER(Hf, 5.34881e-10);
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,614.717e-12);
      PARAMETER(Br,-1.91733e3);
      PARAMETER(Cr,1.4187);
      PARAMETER(Dr,5.74337e3);
      PARAMETER(Er,0.00450594);
      PARAMETER(Fr,945.297e-12);
      PARAMETER(Df,4.01398e3);
      PARAMETER(Ef,-0.0569372);
      PARAMETER(Ff,409.538e-12);
      PARAMETER(Gr,8787.24);
      PARAMETER(Hr, 3.62419e-10);
      PARAMETER(Gf,7397.89);
      PARAMETER(Hf,5.34881e-10);
      ASSIGN_MODEL(0,1);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,686.119e-12);
      PARAMETER(Br,-4.8646e3);
      PARAMETER(Cr,1.7264);
      PARAMETER(Dr,6.42716e3);
      PARAMETER(Er,-0.111246);
      PARAMETER(Fr,1168.92e-12);
      PARAMETER(Df,4.01672e3);
      PARAMETER(Ef,-0.0872119);
      PARAMETER(Ff,425.292e-12);
      PARAMETER(Gr,8787.24);
      PARAMETER(Hr,3.62419e-10);
      PARAMETER(Gf,7397.89);
```

```
PARAMETER(Hf, 5.34881e-10);
ASSIGN_MODEL(0,2);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af, 622.374e-12);
PARAMETER(Bf, 9.65819e3);
PARAMETER(Cf,2.40066);
PARAMETER(Ar,235.497e-12);
PARAMETER(Br, 3.18782e3);
PARAMETER(Cr, 1.23948);
PARAMETER(Dr,3.47932e3);
PARAMETER(Er,0.185427);
PARAMETER(Fr,458.425e-12);
PARAMETER(Df,3.87425e3);
PARAMETER(Ef,-0.00745495);
PARAMETER(Ff,304.477e-12);
PARAMETER(Gr, 8787.24);
PARAMETER(Hr,3.62419e-10);
PARAMETER(Gf,7397.89);
PARAMETER(Hf, 5.34881e-10);
ASSIGN_MODEL(0,3);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,245.966e-12);
PARAMETER(Br,3.22057e3);
PARAMETER(Cr,1.09387 );
PARAMETER(Dr,3.50514e3);
PARAMETER(Er,0.203938);
PARAMETER(Fr,512.46e-12);
PARAMETER(Df,3.95038e3);
PARAMETER(Ef,-0.0425826);
PARAMETER(Ff,317.737e-12);
PARAMETER(Gr,8787.24);
PARAMETER(Hr,3.62419e-10);
PARAMETER(Gf,7397.89);
PARAMETER(Hf,5.34881e-10);
ASSIGN_MODEL(0,4);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af, 617.764e-12);
PARAMETER(Bf, 8.29603e3);
PARAMETER(Cf,2.36757);
PARAMETER(Ar,313.049e-12);
PARAMETER(Br,-1.22843e3);
PARAMETER(Cr,1.07053);
PARAMETER(Dr,3.81231e3);
PARAMETER(Er,0.204418);
PARAMETER(Fr,386.814e-12);
PARAMETER(Df,3.30586e3);
PARAMETER(Ef,0.00418857);
PARAMETER(Ff,216.259e-12);
PARAMETER(Gr,8787.24);
PARAMETER(Hr,3.62419e-10);
PARAMETER(Gf,7397.89);
PARAMETER(Hf, 5.34881e-10);
```

```
      ASSIGN_MODEL(0,5);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,332.317e-12);
      PARAMETER(Br,-1.9701e3);
      PARAMETER(Cr,0.915276);
      PARAMETER(Dr,3.92611e3);
      PARAMETER(Er,0.217937);
      PARAMETER(Fr,465.742e-12);
      PARAMETER(Df,3.4219e3);
      PARAMETER(Ef,-0.0223023);
      PARAMETER(Ff,244.52e-12);
      PARAMETER(Gr,8787.24);
      PARAMETER(Hr, 3.62419e-10);
      PARAMETER(Gf,7397.89);
      PARAMETER(Hf,5.34881e-10);
      ASSIGN_MODEL(0,6);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,401.039e-12);
      PARAMETER(Br,-3.90276e3);
      PARAMETER(Cr,0.808817);
      PARAMETER(Dr,-12.3926e3);
      PARAMETER(Er,0.111837);
      PARAMETER(Fr,968.52e-12);
      PARAMETER(Df,3.52009e3);
      PARAMETER(Ef,-0.0559078);
      PARAMETER(Ff,260.226e-12);
      PARAMETER(Gr,8787.24);
      PARAMETER(Hr, 3.62419e-10);
      PARAMETER(Gf, 7397.89);
      PARAMETER(Hf,5.34881e-10);
      ASSIGN_MODEL(0,7);

END_CELL;

/* ----------------------------------------
 AMS_035 NAND3
 Last update: 13-5-04
 ----------------------------------------*/
 BEGIN_CELL("na3",3,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      INPUT_NAME(2,"c");
      OUT_NAME(0,"q");

      EXPRESION(0,NOT(AND(AND(INPUT(0),INPUT(1)),INPUT(2))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(3e-6+2e-6)*LMIN*COX);
      PARAMETER(Cinf,(3e-6+2e-6)*LMIN*COX);
      PARAMETER(Af,574.956e-12);
      PARAMETER(Bf,6.07396e3);
      PARAMETER(Cf,0.897781);
```

```
      PARAMETER(Ar,164.02e-12);
      PARAMETER(Br,5.13678e3);
      PARAMETER(Cr,1.83571);
      PARAMETER(Dr,4.13132e3);
      PARAMETER(Er,0.230991);
      PARAMETER(Fr,184.711e-12);
      PARAMETER(Df,2.46507e3);
      PARAMETER(Ef,-0.0287867);
      PARAMETER(Ff,99.121e-12);
      PARAMETER(Gr,10350.6);
      PARAMETER(Hr,4.24161e-10);
      PARAMETER(Gf,12423);
      PARAMETER(Hf,3.45625e-10);
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(3e-6+2e-6)*LMIN*COX);
      PARAMETER(Cinf,(3e-6+2e-6)*LMIN*COX);
      PARAMETER(Af, 325.874e-12);
      PARAMETER(Bf, 9.57398e3);
      PARAMETER(Cf,0.774685);
      PARAMETER(Ar,144.735e-12);
      PARAMETER(Br,4.79631e3);
      PARAMETER(Cr, 1.66868);
      PARAMETER(Dr,4.12395e3);
      PARAMETER(Er,0.220792);
      PARAMETER(Fr,137.495e-12);
      PARAMETER(Df,2.5421e3);
      PARAMETER(Ef,0.0185453);
      PARAMETER(Ff,78.0013e-12);
      PARAMETER(Gr,10337.3);
      PARAMETER(Hr,3.07808e-10);
      PARAMETER(Gf,12426);
      PARAMETER(Hf,3.45645e-10);
      ASSIGN_MODEL(0,1);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(3e-6+2e-6)*LMIN*COX);
      PARAMETER(Cinf,(3e-6+2e-6)*LMIN*COX);
      PARAMETER(Af,257.311e-12);
      PARAMETER(Bf, 10.3912e3);
      PARAMETER(Cf,0.903597);
      PARAMETER(Ar,134.76e-12);
      PARAMETER(Br,3.68394e3);
      PARAMETER(Cr,1.41142);
      PARAMETER(Dr,4.17268e3);
      PARAMETER(Er,0.211886);
      PARAMETER(Fr,80.1064e-12);
      PARAMETER(Df,2.48734e3);
      PARAMETER(Ef,0.0744736);
      PARAMETER(Ff,48.6546e-12);
      PARAMETER(Gr,10343.8);
      PARAMETER(Hr,1.90983e-10);
      PARAMETER(Gf,12661.3);
      PARAMETER(Hf,3.34477e-10);
      ASSIGN_MODEL(0,2);

  END_CELL;

 /* ----------------------------------------
  AMS_035 NAND4
  Last update: 15-5-04
  NOTA: Entradas con par?metros 0
  ----------------------------------------*/
```

```
BEGIN_CELL("na4",4,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      INPUT_NAME(2,"c");
      INPUT_NAME(3,"d");

      OUT_NAME(0,"q");

      EXPRESION(0,NOT(AND(INPUT(0),AND(INPUT(1),INPUT(2)))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,148.426e-12);
      PARAMETER(Bf,2.06116e3);
      PARAMETER(Cf,1.22981);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,3.8377e3);
      PARAMETER(Er,-0.0468231);
      PARAMETER(Fr,354.294e-12);
      PARAMETER(Df,4.22852e3);
      PARAMETER(Ef,0.200285);
      PARAMETER(Ff,495.836e-12);
      PARAMETER(Gr,9388.37);
      PARAMETER(Hr,1.2003e-10);
      PARAMETER(Gf,14786.2);
      PARAMETER(Hf,1.53698e-10);
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,131.938e-12);
      PARAMETER(Bf, 2.81598e3);
      PARAMETER(Cf,1.34962 );
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,3.8479e3);
      PARAMETER(Er,-0.0126987);
      PARAMETER(Fr,338.814e-12);
      PARAMETER(Df,4.19697e3);
      PARAMETER(Ef,0.179331);
      PARAMETER(Ff,442.43e-12);
      PARAMETER(Gr,9401.13);
      PARAMETER(Hr,1.19501e-10);
      PARAMETER(Gf,14793.8);
      PARAMETER(Hf,1.53721e-10);
      ASSIGN_MODEL(0,1);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
      PARAMETER(Af, 173.881e-12);
      PARAMETER(Bf,1.10117e3);
      PARAMETER(Cf,1.04939);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,3.78927e3);
      PARAMETER(Er,-0.0336743);
      PARAMETER(Fr,292.956e-12);
```

```
    PARAMETER(Df,4.04674e3);
    PARAMETER(Ef,0.203138);
    PARAMETER(Ff,406.107e-12);
    PARAMETER(Gr,9537.48);
    PARAMETER(Hr,1.02115e-10);
    PARAMETER(Gf,14871);
    PARAMETER(Hf,1.48975e-10);
    ASSIGN_MODEL(0,2);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,156.56e-12);
    PARAMETER(Bf,1.46733e3);
    PARAMETER(Cf,1.2462);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,3.74246e3);
    PARAMETER(Er,0.000284318);
    PARAMETER(Fr,278.335e-12);
    PARAMETER(Df,4.00894e3);
    PARAMETER(Ef,0.183599);
    PARAMETER(Ff,353.768e-12);
    PARAMETER(Gr,9567.24);
    PARAMETER(Hr,1.00845e-10);
    PARAMETER(Gf,14880.9);
    PARAMETER(Hf,1.48919e-10);
    ASSIGN_MODEL(0,3);

END_CELL;

/* ----------------------------------------
AMS_035 NAND8
Last update: 15-5-04
----------------------------------------*/
BEGIN_CELL("na8",8,1);
    INPUT_NAME(0,"a");
    INPUT_NAME(1,"b");
    INPUT_NAME(2,"c");
    INPUT_NAME(3,"d");
    INPUT_NAME(4,"e");
    INPUT_NAME(5,"f");
    INPUT_NAME(6,"g");
    INPUT_NAME(7,"h");

    OUT_NAME(0,"q");

    EXPRESION(0,
    NOT(AND(
            AND(
                    AND(INPUT(0),INPUT(1)),
                    AND(INPUT(2),INPUT(3))
            ),
            AND(
                    AND(INPUT(4),INPUT(5)),
                    AND(INPUT(6),INPUT(7))
            )
    )));

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,444.896e-12);
```

```
PARAMETER(Bf,-4.49293e3);
PARAMETER(Cf,1.3447);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.30611e3);
PARAMETER(Er,-0.0444162);
PARAMETER(Fr,489.439e-12);
PARAMETER(Df,-11.7032e3);
PARAMETER(Ef,0.0860099);
PARAMETER(Ff,1056.58e-12);
PARAMETER(Gr,9636.93);
PARAMETER(Hr,1.50529e-10);
PARAMETER(Gf,15277.6);
PARAMETER(Hf,1.89781e-10);
ASSIGN_MODEL(0,0);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,420.434e-12);
PARAMETER(Bf,-3.10114e3);
PARAMETER(Cf,1.41865);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.41486e3);
PARAMETER(Er,-0.066022);
PARAMETER(Fr,513.602e-12);
PARAMETER(Df,-5.83899e3);
PARAMETER(Ef,-0.0875689);
PARAMETER(Ff,1317.37e-12);
PARAMETER(Gr,9626.87);
PARAMETER(Hr,1.50611e-10);
PARAMETER(Gf,15281.3);
PARAMETER(Hf,1.89761e-10 );
ASSIGN_MODEL(0,1);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,488.229e-12);
PARAMETER(Bf,-4.96314e3);
PARAMETER(Cf,1.07927);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.44593e3);
PARAMETER(Er,-0.0948699);
PARAMETER(Fr,528.356e-12);
PARAMETER(Df,-10.4623e3);
PARAMETER(Ef,-0.135732);
PARAMETER(Ff,1494.16e-12);
PARAMETER(Gr, 9605.95);
PARAMETER(Hr,1.51123e-10);
PARAMETER(Gf,15276.1);
PARAMETER(Hf,1.89827e-10);
ASSIGN_MODEL(0,2);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,138.393e-12);
PARAMETER(Bf,1.85392e3);
```

```
PARAMETER(Cf,1.33651);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.34353e3);
PARAMETER(Er,-0.0209743);
PARAMETER(Fr,408.724e-12);
PARAMETER(Df,4.23802e3);
PARAMETER(Ef,0.178168);
PARAMETER(Ff,519.809e-12);
PARAMETER(Gr,9380.29);
PARAMETER(Hr,1.29299e-10);
PARAMETER(Gf,15341.9);
PARAMETER(Hf,1.86333e-10);
ASSIGN_MODEL(0,3);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,162.942e-12);
PARAMETER(Bf,0.926349e3);
PARAMETER(Cf,1.24738);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.38968e3);
PARAMETER(Er,-0.0532629);
PARAMETER(Fr,421.537e-12);
PARAMETER(Df,4.27005e3);
PARAMETER(Ef,0.19868);
PARAMETER(Ff,572.599e-12);
PARAMETER(Gr, 9366.13);
PARAMETER(Hr, 1.29799e-10);
PARAMETER(Gf,15343.4 );
PARAMETER(Hf,1.86294e-10);
ASSIGN_MODEL(0,4);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,221.583e-12);
PARAMETER(Bf,0.922602e3);
PARAMETER(Cf,1.16988);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.05044e3);
PARAMETER(Er,-0.0168499);
PARAMETER(Fr,318.787e-12);
PARAMETER(Df,4.07645e3);
PARAMETER(Ef,0.198498);
PARAMETER(Ff,442.964e-12);
PARAMETER(Gr,9561.44);
PARAMETER(Hr, 1.07488e-10);
PARAMETER(Gf,15657.6);
PARAMETER(Hf,1.66659e-10);
ASSIGN_MODEL(0,5);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,241.196e-12);
PARAMETER(Bf, 0.321273e3);
PARAMETER(Cf,0.965147);
```

```
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,4.08854e3);
    PARAMETER(Er,-0.0392041);
    PARAMETER(Fr,347.179e-12);
    PARAMETER(Df,4.10297e3);
    PARAMETER(Ef,0.213601);
    PARAMETER(Ff,521.813e-12);
    PARAMETER(Gr, 9525.49);
    PARAMETER(Hr, 1.09161e-10);
    PARAMETER(Gf,15648);
    PARAMETER(Hf,1.66777e-10);
    ASSIGN_MODEL(0,6);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,289.653e-12);
    PARAMETER(Bf,-0.565863e3);
    PARAMETER(Cf,0.903924);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,4.13417e3);
    PARAMETER(Er,-0.0699182);
    PARAMETER(Fr,361.994e-12);
    PARAMETER(Df,6.37322e3);
    PARAMETER(Ef,0.0102985);
    PARAMETER(Ff,883.525e-12);
    PARAMETER(Gr,9492.42);
    PARAMETER(Hr, 1.11168e-10 );
    PARAMETER(Gf,15661.3);
    PARAMETER(Hf,1.66625e-10);
    ASSIGN_MODEL(0,7);

END_CELL;

/* ----------------------------------------
EO1_035 EO1
Last update: 15-5-04
----------------------------------------*/
BEGIN_CELL("eo1",2,1);
    INPUT_NAME(0,"a");
    INPUT_NAME(1,"b");
    OUT_NAME(0,"q");

    EXPRESION(0,OR(
            AND(INPUT(0),NOT(INPUT(1))),
            AND(NOT(INPUT(0)),INPUT(1))
    ));

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(3e-6+8e-6)*LMIN*COX);
    PARAMETER(Cinf,(3e-6+8e-6)*LMIN*COX);
    PARAMETER(Af,158.076e-12);
    PARAMETER(Bf,5.46158e3);
    PARAMETER(Cf,1.7891);
    PARAMETER(Ar,165.854e-12);
    PARAMETER(Br,11.4816e3);
    PARAMETER(Cr,1.53512);
    PARAMETER(Dr,3.57941e3);
    PARAMETER(Er,0.0942382);
    PARAMETER(Fr,291.721e-12);
```

```
      PARAMETER(Df,3.38322e3);
      PARAMETER(Ef,0.0703304);
      PARAMETER(Ff,263.039e-12);
      PARAMETER(Gr,9861.7);
      PARAMETER(Hr,2.77883e-10);
      PARAMETER(Gf,7060.74);
      PARAMETER(Hf,1.40968e-10);
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(3e-6+8e-6)*LMIN*COX);
      PARAMETER(Cinf,(3e-6+8e-6)*LMIN*COX);
      PARAMETER(Af, 94.52e-12);
      PARAMETER(Bf,5.84067e3);
      PARAMETER(Cf,1.7085);
      PARAMETER(Ar,262.26e-12);
      PARAMETER(Br,10.0848e3);
      PARAMETER(Cr,2.22407);
      PARAMETER(Dr,3.64409e3);
      PARAMETER(Er,0.0538591);
      PARAMETER(Fr,188.315e-12);
      PARAMETER(Df,3.31971e3);
      PARAMETER(Ef,0.109484);
      PARAMETER(Ff,216.276e-12);
      PARAMETER(Gr,10072.3);
      PARAMETER(Hr, 1.81501e-10);
      PARAMETER(Gf,7064.72);
      PARAMETER(Hf,1.40092e-10);
      ASSIGN_MODEL(0,1);

 END_CELL;

 /*-----------------------------------------
 AMS_035 NO3
 Last update: 13-5-04
 -----------------------------------------*/
BEGIN_CELL("no3",3,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      INPUT_NAME(2,"c");
      OUT_NAME(0,"q");

      EXPRESION(0,NOT(OR(OR(INPUT(0),INPUT(1)),INPUT(2))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Af,527.608e-12);
      PARAMETER(Bf, 8.95091e3);
      PARAMETER(Cf,1.47266);
      PARAMETER(Ar,113.866e-12);
      PARAMETER(Br,5.38921e3);
      PARAMETER(Cr,1.01642);
      PARAMETER(Dr,3.63826e3);
      PARAMETER(Er,0.158536);
      PARAMETER(Fr,40.5978e-12);
      PARAMETER(Df,3.43069e3);
      PARAMETER(Ef,0.134941);
      PARAMETER(Ff,87.9478e-12);
      PARAMETER(Gr,10895.8);
      PARAMETER(Hr,2.25327e-10);
      PARAMETER(Gf,14840.8);
      PARAMETER(Hf,2.5598e-10);
      ASSIGN_MODEL(0,0);
```

```
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Af,542.968e-12);
      PARAMETER(Bf,9.25063e3);
      PARAMETER(Cf,1.55716);
      PARAMETER(Ar,644.944e-12);
      PARAMETER(Br,-3.4006e3);
      PARAMETER(Cr,0.916424);
      PARAMETER(Dr,4.10288e3);
      PARAMETER(Er,0.0858778);
      PARAMETER(Fr,135.897e-12);
      PARAMETER(Df,3.53471e3);
      PARAMETER(Ef,0.141252);
      PARAMETER(Ff,181.095e-12);
      PARAMETER(Gr,9480.32);
      PARAMETER(Hr,3.23326e-10);
      PARAMETER(Gf,13707.9);
      PARAMETER(Hf,5.64531e-10);
      ASSIGN_MODEL(0,1);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Af, 583.071e-12);
      PARAMETER(Bf,9.37351e3);
      PARAMETER(Cf,1.76315);
      PARAMETER(Ar, 1507.59e-12);
      PARAMETER(Br,-7.25125e3);
      PARAMETER(Cr,1.31042);
      PARAMETER(Dr,4.06515e3);
      PARAMETER(Er,0.0379025);
      PARAMETER(Fr,196.828e-12);
      PARAMETER(Df,3.73248e3);
      PARAMETER(Ef,0.153299);
      PARAMETER(Ff,233.249e-12);
      PARAMETER(Gr,9372.77);
      PARAMETER(Hr,3.28939e-10);
      PARAMETER(Gf,10683.2);
      PARAMETER(Hf,1.07031e-09);
      ASSIGN_MODEL(0,2);

 END_CELL;

 /* ----------------------------------------
 AMS_035 NO4
 Last update: 15-5-04
 NOTA: No funciona AUTODDM
 ----------------------------------------*/
BEGIN_CELL("no4",4,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      INPUT_NAME(2,"c");
      INPUT_NAME(3,"d");

      OUT_NAME(0,"q");


EXPRESION(0,NOT(OR(OR(INPUT(0),INPUT(1)),OR(INPUT(2),INPUT(3)))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
```

```
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,3.5068e3);
PARAMETER(Er,0.0856259);
PARAMETER(Fr,462.168e-12);
PARAMETER(Df,3.95094e3);
PARAMETER(Ef,0.0364723);
PARAMETER(Ff,326.658e-12);
PARAMETER(Gr, 9817.5);
PARAMETER(Hr, 1.08192e-10);
PARAMETER(Gf,13958.1);
PARAMETER(Hf,1.89091e-10);
ASSIGN_MODEL(0,0);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,3.49817e3);
PARAMETER(Er,0.043343);
PARAMETER(Fr,385.353e-12);
PARAMETER(Df,3.94524e3);
PARAMETER(Ef,0.0772384);
PARAMETER(Ff,275.424e-12);
PARAMETER(Gr, 9818.61);
PARAMETER(Hr,1.08065e-10);
PARAMETER(Gf,13978.5);
PARAMETER(Hf,1.87288e-10);
ASSIGN_MODEL(0,1);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,3.40892e3);
PARAMETER(Er,0.0882182);
PARAMETER(Fr,402.959e-12);
PARAMETER(Df,3.9495e3);
PARAMETER(Ef,0.0413143);
PARAMETER(Ff,318.191e-12);
PARAMETER(Gr,9832.81);
PARAMETER(Hr, 1.07954e-10);
PARAMETER(Gf,14225.8);
PARAMETER(Hf, 1.66217e-10);
ASSIGN_MODEL(0,2);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
```

```
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,3.39844e3);
      PARAMETER(Er,0.0458158);
      PARAMETER(Fr,328.088e-12);
      PARAMETER(Df,3.93588e3);
      PARAMETER(Ef,0.0811938);
      PARAMETER(Ff,266.705e-12);
      PARAMETER(Gr,9832.7);
      PARAMETER(Hr,1.07856e-10);
      PARAMETER(Gf,14258);
      PARAMETER(Hf,1.63883e-10);
      ASSIGN_MODEL(0,3);

 END_CELL;

/* ----------------------------------------
AMS_035 OR2
Last update: 13-5-04
----------------------------------------*/
 BEGIN_CELL("or2",2,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      OUT_NAME(0,"q");

      EXPRESION(0,OR(INPUT(0),INPUT(1)));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Af,17.7535e-12);
      PARAMETER(Bf,12.1241e3);
      PARAMETER(Cf,1.56426);
      PARAMETER(Ar,215.31e-12);
      PARAMETER(Br,0.965249e3);
      PARAMETER(Cr,1.91567);
      PARAMETER(Dr,3.70426e3);
      PARAMETER(Er,0.114831);
      PARAMETER(Fr,141.609e-12);
      PARAMETER(Df,3.92994e3);
      PARAMETER(Ef,0.077138);
      PARAMETER(Ff,153.231e-12);
      PARAMETER(Gr,9945.78);
      PARAMETER(Hr, 9.3186e-11);
      PARAMETER(Gf,14511.1);
      PARAMETER(Hf,1.49543e-10);
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Af,205.078e-12);
      PARAMETER(Bf,5.50566e3);
      PARAMETER(Cf,0.904671);
      PARAMETER(Ar,318.335e-12);
      PARAMETER(Br,-1.09867e3);
      PARAMETER(Cr,1.96767);
      PARAMETER(Dr,3.86164e3);
      PARAMETER(Er,0.0704065);
      PARAMETER(Fr,190.702e-12);
      PARAMETER(Df,4.03554e3);
```

```
     PARAMETER(Ef,0.112354);
     PARAMETER(Ff,223.797e-12);
     PARAMETER(Gr,9787.85);
     PARAMETER(Hr,1.11842e-10);
     PARAMETER(Gf,14461.1);
     PARAMETER(Hf,1.5256e-10);
     ASSIGN_MODEL(0,1);


END_CELL;

/* ----------------------------------------
AMS_035 or3
Last update: 13-5-04
----------------------------------------*/
BEGIN_CELL("or3",3,1);
     INPUT_NAME(0,"a");
     INPUT_NAME(1,"b");
     INPUT_NAME(2,"c");
     OUT_NAME(0,"q");

     EXPRESION(0,OR(OR(INPUT(0),INPUT(1)),INPUT(2)));

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+6e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+6e-6)*LMIN*COX);
     PARAMETER(Af,34.1089e-12);
     PARAMETER(Bf,11.1257e3);
     PARAMETER(Cf,1.63224);
     PARAMETER(Ar, 547.971e-12);
     PARAMETER(Br,-2.59492e3);
     PARAMETER(Cr,2.24143);
     PARAMETER(Dr,3.87064e3);
     PARAMETER(Er,0.101591);
     PARAMETER(Fr,153.597e-12);
     PARAMETER(Df,3.92967e3);
     PARAMETER(Ef,0.0906772);
     PARAMETER(Ff,195.923e-12);
     PARAMETER(Gr,9933.69);
     PARAMETER(Hr,9.9524e-11);
     PARAMETER(Gf,14957.1);
     PARAMETER(Hf,1.67405e-10);
     ASSIGN_MODEL(0,0);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+6e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+6e-6)*LMIN*COX);
     PARAMETER(Af,251.439e-12);
     PARAMETER(Bf,5.24299e3);
     PARAMETER(Cf,0.853934);
     PARAMETER(Ar, 563.43e-12);
     PARAMETER(Br,-1.10045e3);
     PARAMETER(Cr,2.05054);
     PARAMETER(Dr,4.12739e3);
     PARAMETER(Er,0.0672776);
     PARAMETER(Fr,243.668e-12);
     PARAMETER(Df,4.0149e3);
     PARAMETER(Ef,0.121072);
     PARAMETER(Ff,311.915e-12);
     PARAMETER(Gr,9727.62);
     PARAMETER(Hr,1.24924e-10);
     PARAMETER(Gf,14765.9);
     PARAMETER(Hf,1.85867e-10);
     ASSIGN_MODEL(0,1);
```

```
      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+6e-6)*LMIN*COX);
      PARAMETER(Af,951.912e-12);
      PARAMETER(Bf,-2.72035e3);
      PARAMETER(Cf,3.67134);
      PARAMETER(Ar,599.588e-12);
      PARAMETER(Br,-0.0295967e3);
      PARAMETER(Cr, 2.01094);
      PARAMETER(Dr,4.17077e3);
      PARAMETER(Er,0.0268944);
      PARAMETER(Fr,300.211e-12);
      PARAMETER(Df,4.26474e3);
      PARAMETER(Ef,0.142996);
      PARAMETER(Ff,380.555e-12);
      PARAMETER(Gr,9793.7);
      PARAMETER(Hr,1.52921e-10);
      PARAMETER(Gf,14751.4);
      PARAMETER(Hf,1.87041e-10);
      ASSIGN_MODEL(0,2);

 END_CELL;

 /* ----------------------------------------
 AMS_035 OR4
 Last update: 15-5-04
 Nota: 3 entradas no tienen parametros DDM-Xr
 ----------------------------------------*/
BEGIN_CELL("or4",4,1);
      INPUT_NAME(0,"a");
      INPUT_NAME(1,"b");
      INPUT_NAME(2,"c");
      INPUT_NAME(3,"d");

      OUT_NAME(0,"q");

      EXPRESION(0,OR(OR(INPUT(0),INPUT(1)),OR(INPUT(2),INPUT(3))));

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Af,87.852e-12);
      PARAMETER(Bf,12.6185e3);
      PARAMETER(Cf,1.62772);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,2.69628e3);
      PARAMETER(Er,0.0523455);
      PARAMETER(Fr,218.538e-12);
      PARAMETER(Df,3.82631e3);
      PARAMETER(Ef,0.0965442);
      PARAMETER(Ff,325.452e-12);
      PARAMETER(Gr,9994.26);
      PARAMETER(Hr,2.4235e-10);
      PARAMETER(Gf,12679.7);
      PARAMETER(Hf,2.44165e-10);
      ASSIGN_MODEL(0,0);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
      PARAMETER(Af,142.148e-12);
```

```
    PARAMETER(Bf,11.8867e3);
    PARAMETER(Cf,2.12398);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,2.61292e3);
    PARAMETER(Er,0.0989263);
    PARAMETER(Fr,168.919e-12);
    PARAMETER(Df,3.83359e3);
    PARAMETER(Ef,0.0595226);
    PARAMETER(Ff,248.223e-12);
    PARAMETER(Gr,10116.1);
    PARAMETER(Hr,2.28119e-10);
    PARAMETER(Gf,12726.7);
    PARAMETER(Hf, 2.41653e-10);
    ASSIGN_MODEL(0,1);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
    PARAMETER(Af,23.6843e-12);
    PARAMETER(Bf,12.4845e3);
    PARAMETER(Cf,1.44371);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,2.94307e3);
    PARAMETER(Er,0.0604697);
    PARAMETER(Fr,209.945e-12);
    PARAMETER(Df,3.85886e3);
    PARAMETER(Ef,0.0989165);
    PARAMETER(Ff,282.875e-12);
    PARAMETER(Gr,9921.69);
    PARAMETER(Hr, 1.62925e-10);
    PARAMETER(Gf,12817.4);
    PARAMETER(Hf,2.51713e-10);
    ASSIGN_MODEL(0,2);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+4e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+4e-6)*LMIN*COX);
    PARAMETER(Af,58.9788e-12);
    PARAMETER(Bf,12.5824e3);
    PARAMETER(Cf,1.95951);
    PARAMETER(Ar,226.228e-12);
    PARAMETER(Br,3.03949e3);
    PARAMETER(Cr,2.0869);
    PARAMETER(Dr,2.80556e3);
    PARAMETER(Er,0.104432);
    PARAMETER(Fr,161.876e-12);
    PARAMETER(Df,3.84797e3);
    PARAMETER(Ef,0.0623502);
    PARAMETER(Ff,207.438e-12);
    PARAMETER(Gr,10056.6);
    PARAMETER(Hr, 1.46883e-10);
    PARAMETER(Gf,12861.3);
    PARAMETER(Hf,2.48903e-10);
    ASSIGN_MODEL(0,3);

END_CELL;

/*----------------------------------------
AMS_035 NAND5
Last update: 30-05-04
```

```
 CUIDADO los par□metros de la pendiente est□n mal..revisar puerta
completa
 ----------------------------------------*/
 BEGIN_CELL("na5",5,1);
     INPUT_NAME(0,"a");
     INPUT_NAME(1,"b");
     INPUT_NAME(2,"c");
     INPUT_NAME(3,"d");
     INPUT_NAME(4,"e");
     OUT_NAME(0,"q");

     EXPRESION(0,NOT(AND(
                     INPUT(0),
                     AND(
                             AND(INPUT(1),INPUT(2)),
                             AND(INPUT(2),INPUT(3)))
                     )
             ));

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1.3e-6+1e-6)*LMIN*COX);
     PARAMETER(Cinf,(1.3e-6+1e-6)*LMIN*COX);
     PARAMETER(Af,247.115e-12);
     PARAMETER(Bf, 0.728928e3);
     PARAMETER(Cf,0.982359);
     PARAMETER(Ar,0e-12);
     PARAMETER(Br,0e3);
     PARAMETER(Cr,0e3);
     PARAMETER(Dr,3.79917e3);
     PARAMETER(Er,-0.0818712);
     PARAMETER(Fr,398.443e-12);
     PARAMETER(Df,-11.299e3);
     PARAMETER(Ef,-0.0552099);
     PARAMETER(Ff,1279e-12);
     PARAMETER(Gr,9319.85);
     PARAMETER(Hr,1.23003e-10);
     PARAMETER(Gf,14733.6);
     PARAMETER(Hf,1.55603e-10);
     ASSIGN_MODEL(0,0);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,( 1.3e-6+1e-6)*LMIN*COX);
     PARAMETER(Cinf,( 1.3e-6+1e-6)*LMIN*COX);
     PARAMETER(Af,204.259e-12);
     PARAMETER(Bf, 1.17288e3);
     PARAMETER(Cf,1.14972);
     PARAMETER(Ar,0e-12);
     PARAMETER(Br,0e3);
     PARAMETER(Cr,0e3);
     PARAMETER(Dr,3.82174e3);
     PARAMETER(Er,-0.0514129);
     PARAMETER(Fr,380.982e-12);
     PARAMETER(Df,-8.77835e3);
     PARAMETER(Ef,0.153401);
     PARAMETER(Ff,851.559e-12);
     PARAMETER(Gr,9358.43);
     PARAMETER(Hr,1.21501e-10);
     PARAMETER(Gf,14736.9);
     PARAMETER(Hf,1.55563e-10);
     ASSIGN_MODEL(0,1);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,( 1.3e-6+1e-6)*LMIN*COX);
     PARAMETER(Cinf,( 1.3e-6+1e-6)*LMIN*COX);
```

```
    PARAMETER(Af,180.563e-12);
    PARAMETER(Bf,2.30714e3);
    PARAMETER(Cf,1.33495);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0e3);
    PARAMETER(Dr,3.77395e3);
    PARAMETER(Er,-0.0285033);
    PARAMETER(Fr,353.212e-12);
    PARAMETER(Df,4.16642e3);
    PARAMETER(Ef,0.194967);
    PARAMETER(Ff,482.813e-12);
    PARAMETER(Gr,9384.82);
    PARAMETER(Hr,1.20533e-10);
    PARAMETER(Gf,14738);
    PARAMETER(Hf,1.55519e-10);
    ASSIGN_MODEL(0,2);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,( 1.3e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,( 1.3e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,160.019e-12);
    PARAMETER(Bf,2.13272e3);
    PARAMETER(Cf,1.04955);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,3.72146e3);
    PARAMETER(Er,-0.0348763);
    PARAMETER(Fr,295.426e-12);
    PARAMETER(Df,4.00637e3);
    PARAMETER(Ef,0.202624);
    PARAMETER(Ff,407.62e-12);
    PARAMETER(Gr,9563.53);
    PARAMETER(Hr,1.01972e-10 );
    PARAMETER(Gf,14807.1);
    PARAMETER(Hf,1.49348e-10);
    ASSIGN_MODEL(0,3);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,( 1.3e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,( 1.3e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,154.712e-12);
    PARAMETER(Bf,1.90367e3);
    PARAMETER(Cf,1.44091);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,3.67596e3);
    PARAMETER(Er,-0.0011683);
    PARAMETER(Fr,281.011e-12);
    PARAMETER(Df,3.97133e3);
    PARAMETER(Ef,0.182886);
    PARAMETER(Ff,355.448e-12);
    PARAMETER(Gr,9592.58);
    PARAMETER(Hr,1.0072e-10);
    PARAMETER(Gf,14807);
    PARAMETER(Hf,1.49364e-10 );
    ASSIGN_MODEL(0,4);

END_CELL;

/*----------------------------------------
AMS_035 OR5
```

```
 Last update: 30-5-04
 ----------------------------------------*/
BEGIN_CELL("or5",5,1);
     INPUT_NAME(0,"a");
     INPUT_NAME(1,"b");
     INPUT_NAME(2,"c");
     INPUT_NAME(3,"d");
     INPUT_NAME(4,"e");
     OUT_NAME(0,"q");

     EXPRESION(0,OR(
                     INPUT(0),
                     OR(
                             OR(INPUT(1),INPUT(2)),
                             OR(INPUT(3),INPUT(4)))
             ));

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+5.2e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+5.2e-6)*LMIN*COX);
     PARAMETER(Af,258.901e-12);
     PARAMETER(Bf,9.23864e3);
     PARAMETER(Cf,1.16635);
     PARAMETER(Ar,0e-12);
     PARAMETER(Br,0e3);
     PARAMETER(Cr,0);
     PARAMETER(Dr,2.82112e3);
     PARAMETER(Er,0.0101045);
     PARAMETER(Fr,322.303e-12);
     PARAMETER(Df,3.91949e3);
     PARAMETER(Ef,0.132499);
     PARAMETER(Ff,497.033e-12);
     PARAMETER(Gr, 9982.96);
     PARAMETER(Hr,2.7743e-10);
     PARAMETER(Gf,12672.8);
     PARAMETER(Hf,2.71275e-10);
     ASSIGN_MODEL(0,0);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+5.2e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+5.2e-6)*LMIN*COX);
     PARAMETER(Af,27.0145e-12);
     PARAMETER(Bf,12.5601e3);
     PARAMETER(Cf,1.41023);
     PARAMETER(Ar,0e-12);
     PARAMETER(Br,0e3);
     PARAMETER(Cr,0);
     PARAMETER(Dr,2.80962e3);
     PARAMETER(Er,0.0525053);
     PARAMETER(Fr,262.234e-12);
     PARAMETER(Df,3.79484e3);
     PARAMETER(Ef,0.107777);
     PARAMETER(Ff,418.859e-12);
     PARAMETER(Gr,9919.92);
     PARAMETER(Hr,2.52069e-10);
     PARAMETER(Gf,12690.6);
     PARAMETER(Hf,2.70093e-10);
     ASSIGN_MODEL(0,1);

     PARAMETER(Threshold,VDD/2);
     PARAMETER(Cinr,(1e-6+5.2e-6)*LMIN*COX);
     PARAMETER(Cinf,(1e-6+5.2e-6)*LMIN*COX);
     PARAMETER(Af,142.619e-12);
     PARAMETER(Bf,11.5873e3);
```

```
    PARAMETER(Cf,1.88843);
    PARAMETER(Ar,200.92e-12);
    PARAMETER(Br,1.9275e3);
    PARAMETER(Cr,1.3372);
    PARAMETER(Dr,2.6513e3);
    PARAMETER(Er,0.089391);
    PARAMETER(Fr,174.654e-12);
    PARAMETER(Df,3.82656e3);
    PARAMETER(Ef,0.0754809);
    PARAMETER(Ff,293.452e-12);
    PARAMETER(Gr,10054.3);
    PARAMETER(Hr,2.33216e-10);
    PARAMETER(Gf,12905);
    PARAMETER(Hf,2.53819e-10);
    ASSIGN_MODEL(0,2);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+5.2e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+5.2e-6)*LMIN*COX);
    PARAMETER(Af,19.9071e-12);
    PARAMETER(Bf,12.639e3);
    PARAMETER(Cf, 1.59154);
    PARAMETER(Ar,0e-12);
    PARAMETER(Br,0e3);
    PARAMETER(Cr,0);
    PARAMETER(Dr,2.87026e3);
    PARAMETER(Er,0.0594848);
    PARAMETER(Fr,213.528e-12);
    PARAMETER(Df,3.85568e3);
    PARAMETER(Ef,0.0956135);
    PARAMETER(Ff,288.553e-12);
    PARAMETER(Gr,9971.58);
    PARAMETER(Hr,1.62177e-10);
    PARAMETER(Gf,12819.4);
    PARAMETER(Hf,2.51557e-10);
    ASSIGN_MODEL(0,3);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1e-6+5.2e-6)*LMIN*COX);
    PARAMETER(Cinf,(1e-6+5.2e-6)*LMIN*COX);
    PARAMETER(Af,65.8431e-12);
    PARAMETER(Bf,12.2737e3);
    PARAMETER(Cf,1.80808);
    PARAMETER(Ar,141.831e-12);
    PARAMETER(Br,4.78646e3);
    PARAMETER(Cr,1.94248);
    PARAMETER(Dr,2.74898e3);
    PARAMETER(Er,0.101828);
    PARAMETER(Fr,166.363e-12);
    PARAMETER(Df,3.85677e3);
    PARAMETER(Ef,0.0583599);
    PARAMETER(Ff,212.504e-12);
    PARAMETER(Gr,10109.7);
    PARAMETER(Hr,1.46077e-10);
    PARAMETER(Gf,12858.4);
    PARAMETER(Hf,2.48809e-10);
    ASSIGN_MODEL(0,4);

END_CELL;

/*----------------------------------------
AMS_035 AND5
Last update: 31-5-04
----------------------------------------*/
```

```
BEGIN_CELL("and5",5,1);
    INPUT_NAME(0,"a");
    INPUT_NAME(1,"b");
    INPUT_NAME(2,"c");
    INPUT_NAME(3,"d");
    INPUT_NAME(4,"e");
    OUT_NAME(0,"q");

    EXPRESION(0,AND(INPUT(0),
                    AND(
                            AND(INPUT(1),INPUT(2)),
                            AND(INPUT(3),INPUT(4)))
                            ));

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,0e-12);
    PARAMETER(Bf,0e3);
    PARAMETER(Cf,0);
    PARAMETER(Ar,356.932e-12);
    PARAMETER(Br,-3.02798e3);
    PARAMETER(Cr,1.06885);
    PARAMETER(Dr,-9.16167e3);
    PARAMETER(Er,0.168347);
    PARAMETER(Fr,815.53e-12);
    PARAMETER(Df,3.95072e3);
    PARAMETER(Ef,-0.0739445);
    PARAMETER(Ff,308.466e-12);
    PARAMETER(Gr,9198.83);
    PARAMETER(Hr,2.13215e-10);
    PARAMETER(Gf,6594.75);
    PARAMETER(Hf,3.28635e-10);
    ASSIGN_MODEL(0,0);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,0e-12);
    PARAMETER(Bf,0e3);
    PARAMETER(Cf,0);
    PARAMETER(Ar,272.271e-12);
    PARAMETER(Br,1.02085e3);
    PARAMETER(Cr,1.01007);
    PARAMETER(Dr,3.82461e3);
    PARAMETER(Er,0.215497);
    PARAMETER(Fr,456.689e-12);
    PARAMETER(Df,3.96411e3);
    PARAMETER(Ef,0.0422245);
    PARAMETER(Ff,290.216e-12);
    PARAMETER(Gr,9188.7);
    PARAMETER(Hr,2.13361e-10);
    PARAMETER(Gf,6761.82);
    PARAMETER(Hf,3.02843e-10);
    ASSIGN_MODEL(0,1);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,0e-12);
    PARAMETER(Bf,0e3);
    PARAMETER(Cf,0);
    PARAMETER(Ar,274.029e-12);
    PARAMETER(Br,0.124319e3);
```

```
    PARAMETER(Cr,1.15255);
    PARAMETER(Dr,3.77452e3);
    PARAMETER(Er,0.199849);
    PARAMETER(Fr,376.85e-12);
    PARAMETER(Df,3.88715e3);
    PARAMETER(Ef,-0.0173278);
    PARAMETER(Ff,262.127e-12);
    PARAMETER(Gr,9208.63);
    PARAMETER(Hr,2.12851e-10);
    PARAMETER(Gf,7030.45);
    PARAMETER(Hf,2.80969e-10);
    ASSIGN_MODEL(0,2);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,0e-12);
    PARAMETER(Bf,0e3);
    PARAMETER(Cf,0);
    PARAMETER(Ar,215.853e-12);
    PARAMETER(Br,-0.250551e3);
    PARAMETER(Cr,1.17028);
    PARAMETER(Dr,3.91242e3);
    PARAMETER(Er,0.206878);
    PARAMETER(Fr,321.421e-12);
    PARAMETER(Df,3.77215e3);
    PARAMETER(Ef,-0.0206217);
    PARAMETER(Ff,205.229e-12);
    PARAMETER(Gr,9929.11);
    PARAMETER(Hr,1.80193e-10);
    PARAMETER(Gf,7125.95);
    PARAMETER(Hf,1.54696e-10 );
    ASSIGN_MODEL(0,3);

    PARAMETER(Threshold,VDD/2);
    PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
    PARAMETER(Af,371.49e-12);
    PARAMETER(Bf,7.66479e3);
    PARAMETER(Cf,2.35037);
    PARAMETER(Ar,209.808e-12);
    PARAMETER(Br,-0.322556e3);
    PARAMETER(Cr,1.10685);
    PARAMETER(Dr,3.77865e3);
    PARAMETER(Er,0.188956);
    PARAMETER(Fr,269.719e-12);
    PARAMETER(Df,3.6582e3);
    PARAMETER(Ef,0.0151353);
    PARAMETER(Ff,190.59e-12);
    PARAMETER(Gr,9930.09);
    PARAMETER(Hr, 1.80155e-10);
    PARAMETER(Gf,7256.18);
    PARAMETER(Hf,1.39441e-10);
    ASSIGN_MODEL(0,4);

END_CELL;

/*----------------------------------------
AMS_035 NO8
Last update: 1-6-04
----------------------------------------*/
BEGIN_CELL("no8",8,1);
    INPUT_NAME(0,"a");
    INPUT_NAME(1,"b");
```

```
INPUT_NAME(2,"c");
INPUT_NAME(3,"d");
INPUT_NAME(4,"e");
INPUT_NAME(5,"f");
INPUT_NAME(6,"g");
INPUT_NAME(7,"h");

OUT_NAME(0,"q");

EXPRESION(0,NOT(
        OR(
        OR(
                OR(INPUT(0),INPUT(1)),
                OR(INPUT(2),INPUT(3))),
        OR(
                OR(INPUT(4),INPUT(5)),
                OR(INPUT(6),INPUT(7)))
        )
));

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.66634e3);
PARAMETER(Er,0.063842);
PARAMETER(Fr,537.757e-12);
PARAMETER(Df,4.35415e3);
PARAMETER(Ef,0.0704472);
PARAMETER(Ff,311.581e-12);
PARAMETER(Gr,10009.2);
PARAMETER(Hr,1.20035e-10);
PARAMETER(Gf,13722.4);
PARAMETER(Hf,2.49525e-10);
ASSIGN_MODEL(0,0);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.66433e3);
PARAMETER(Er,0.0994645);
PARAMETER(Fr,663.798e-12);
PARAMETER(Df,4.4796e3);
PARAMETER(Ef,0.0374669);
PARAMETER(Ff,397.926e-12);
PARAMETER(Gr,10000.5);
PARAMETER(Hr,1.20857e-10 );
PARAMETER(Gf,13695);
PARAMETER(Hf,2.51419e-10);
ASSIGN_MODEL(0,1);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
```

```
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.73448e3);
PARAMETER(Er,0.1268);
PARAMETER(Fr,746.968e-12);
PARAMETER(Df,4.45759e3);
PARAMETER(Ef,-0.00412814);
PARAMETER(Ff,459.428e-12);
PARAMETER(Gr,10001.2);
PARAMETER(Hr,1.20882e-10);
PARAMETER(Gf,13603.1);
PARAMETER(Hf,2.56689e-10);
ASSIGN_MODEL(0,2);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.33082e3);
PARAMETER(Er,0.0667528);
PARAMETER(Fr,472.713e-12);
PARAMETER(Df,4.37977e3);
PARAMETER(Ef,0.0759679);
PARAMETER(Ff,297.586e-12);
PARAMETER(Gr,9996.73);
PARAMETER(Hr,1.20849e-10);
PARAMETER(Gf,14148.6);
PARAMETER(Hf,2.1267e-10);
ASSIGN_MODEL(0,3);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Af,0e-12);
PARAMETER(Bf,0e3);
PARAMETER(Cf,0);
PARAMETER(Ar,0e-12);
PARAMETER(Br,0e3);
PARAMETER(Cr,0);
PARAMETER(Dr,4.33304e3);
PARAMETER(Er,0.10235);
PARAMETER(Fr,596.734e-12);
PARAMETER(Df,4.4482e3);
PARAMETER(Ef,0.043175);
PARAMETER(Ff,389.031e-12);
PARAMETER(Gr,9985.68);
PARAMETER(Hr,1.21849e-10);
PARAMETER(Gf,14115.9);
PARAMETER(Hf, 2.15013e-10);
ASSIGN_MODEL(0,4);

PARAMETER(Threshold,VDD/2);
PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
```

```
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,4.40655e3);
      PARAMETER(Er,0.129048);
      PARAMETER(Fr,677.659e-12);
      PARAMETER(Df,4.42556e3);
      PARAMETER(Ef,0.00147022);
      PARAMETER(Ff,450.221e-12);
      PARAMETER(Gr,9985.95);
      PARAMETER(Hr,1.21891e-10 );
      PARAMETER(Gf,14029.8);
      PARAMETER(Hf,2.2051e-10);
      ASSIGN_MODEL(0,5);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,4.02781e3);
      PARAMETER(Er,0.051218);
      PARAMETER(Fr,353.137e-12);
      PARAMETER(Df,4.36323e3);
      PARAMETER(Ef,0.086809);
      PARAMETER(Ff,266.582e-12);
      PARAMETER(Gr,10078.9);
      PARAMETER(Hr,1.17223e-10);
      PARAMETER(Gf,14650.9);
      PARAMETER(Hf,1.76579e-10 );
      ASSIGN_MODEL(0,6);

      PARAMETER(Threshold,VDD/2);
      PARAMETER(Cinr,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Cinf,(1.375e-6+1e-6)*LMIN*COX);
      PARAMETER(Af,0e-12);
      PARAMETER(Bf,0e3);
      PARAMETER(Cf,0);
      PARAMETER(Ar,0e-12);
      PARAMETER(Br,0e3);
      PARAMETER(Cr,0);
      PARAMETER(Dr,4.04923e3);
      PARAMETER(Er,0.0917761);
      PARAMETER(Fr,428.74e-12);
      PARAMETER(Df,4.3806e3);
      PARAMETER(Ef,0.0450458);
      PARAMETER(Ff,318.638e-12);
      PARAMETER(Gr,10075.1);
      PARAMETER(Hr,1.17328e-10);
      PARAMETER(Gf,14624.5);
      PARAMETER(Hf,1.78596e-10 );
      ASSIGN_MODEL(0,7);

END_CELL;
```

La siguiente librería está escrita en XML, el formato de librerías adoptado por HALOTIS versión 2. Destacar que el formato XML es genérico y puede ser utilizado por cualquier aplicación, de forma que, cada aplicación define cual es la estructura y elementos que permite para cada tipo de documento XML que utiliza. La definición del formato para una determinada aplicación de XML queda formalmente especificada en la llamada *definición de tipo de documento* (DTD). Un DTD es un fichero de texto con una sintaxis concreta que define las restricciones en la estructura y sintaxis del documento XML. A continuación se presenta el DTD de la librería de celdas de HALOTIS:

```
<!-- HALOTIS Cells library DTD
     Version of DTD: 19 Jan 2006
-->

<!ELEMENT library (name, modelname, vddh, vddl, param*,
cells) >
<!ELEMENT cells (logiccell* | sequentialcell*) >
<!ELEMENT logiccell (name, input*, out*, model*) >
<!ELEMENT sequentialcell (name, input*, statevar+, out*
,table, model*) >

<!ELEMENT out (name,expression*) >
<!ELEMENT model (param+) >

<!ATTLIST model
          input CDATA #REQUIRED
          out CDATA #REQUIRED>
<!ATTLIST param
          name CDATA #REQUIRED >

<!ELEMENT name (#PCDATA) >
<!ELEMENT modelname (#PCDATA) >
<!ELEMENT vddh (#PCDATA) >
<!ELEMENT vddl (#PCDATA) >
<!ELEMENT input (#PCDATA) >
<!ELEMENT param (#PCDATA) >
<!ELEMENT expression (#PCDATA) >

<!ELEMENT statevar (#PCDATA) >
<!ATTLIST statevar
          nextstate CDATA #REQUIRED >

<!ELEMENT table (cols,row+) >
<!ELEMENT cols (#PCDATA) >
<!ELEMENT row (#PCDATA) >
```

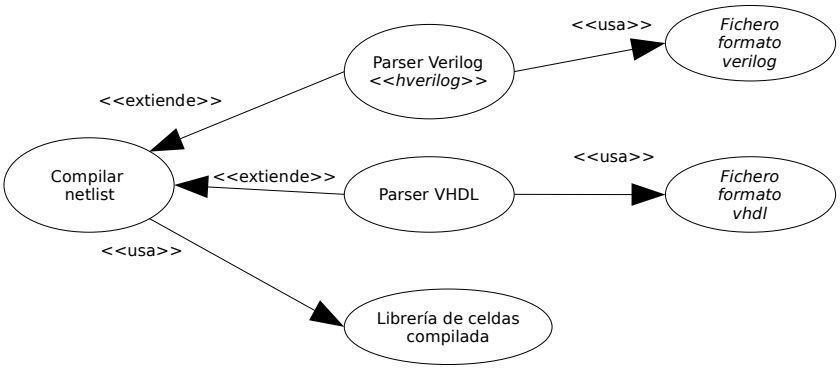
Por último se incluye la librería celdas AMS 0.35μm en formato XML con los parámetros del modelo de intensidad:

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<!DOCTYPE library PUBLIC "http://www.dte.us.es"
"hcells.dtd">
<!-- Libreria AMS 035 de prueba -->
<library>
  <name>AMS035</name>
  <modelname>iddm</modelname>
  <vddh>3.3</vddh>
  <vddl>0</vddl>
  <param name="vtn">0.81804</param>
  <param name="vtp">2.7622</param>
  <cells>
    <logiccell>
      <name>in1</name>
      <input>a</input>
      <out>
        <name>q</name>
        <expression>not(a)</expression>
      </out>
      <model input="a" out="q">
        <param name="Threshold">1.65</param>
        <param name="Cinr">4.0324149e-15</param>
        <param name="Cinf">4.0324149e-15</param>
        <param name="Af">94.1408e-12</param>
        <param name="Bf">10.1678e3</param>
        <param name="Cf">0.926382</param>
        <param name="Ar">32.005e-12</param>
        <param name="Br">5.24642e3</param>
        <param name="Cr">1.05815</param>
        <param name="Dr">4.33892e3</param>
        <param name="Er">0.176655</param>
        <param name="Fr">30.4103e-12</param>
        <param name="Df">3.64233e3</param>
        <param name="Ef">0.0946134</param>
        <param name="Ff">32.1833e-12</param>
        <param name="Gr">9916.13</param>
        <param name="Hr">8.86367e-11</param>
        <param name="Gf">7005.88</param>
        <param name="Hf">7.01589e-11</param>
        <param name="R">6.60323e-18</param>
        <param name="Q">5.20299e-08</param>
      </model>
    </logiccell>
    <!-- NAND2 -->
    <logiccell>
      <name>na2</name>
      <input>a</input>
      <input>b</input>
      <out>
        <name>q</name>
        <expression>nand(a,b)</expression>
      </out>
      <model input="a" out="q">
          <param name="Threshold">1.65</param>
          <param name="Cinr">5.3765532e-15</param>
```

```
            <param name="Cinf">5.3765532e-15</param>
            <param name="Af">177.033e-12</param>
            <param name="Bf">10.3333e3</param>
            <param name="Cf">0.882377</param>
            <param name="Ar">64.6769e-12</param>
            <param name="Br">4.41094e3</param>
            <param name="Cr">1.31169</param>
            <param name="Dr">4.20692e3</param>
            <param name="Er">0.201717</param>
            <param name="Fr">55.4206e-12</param>
            <param name="Df">2.77464e3</param>
            <param name="Ef">0.0825482</param>
            <param name="Ff">42.3562e-12</param>
            <param name="Gr">10340.8</param>
            <param name="Hr">1.33747e-10</param>
            <param name="Gf">6054.05</param>
            <param name="Hf">8.64447e-11</param>
            <param name="R">4.78655e-18</param>
            <param name="Q">0.000418447</param>
        </model>
        <model input="b" out="q">
            <param name="Threshold">1.65</param>
            <param name="Cinr">5.3765532e-15</param>
            <param name="Cinf">5.3765532e-15</param>
            <param name="Af">239.906e-12</param>
            <param name="Bf">9.48001e3</param>
            <param name="Cf">0.816024</param>
            <param name="Ar">86.6308e-12</param>
            <param name="Br">5.17971e3</param>
            <param name="Cr">1.63747</param>
            <param name="Dr">4.15532e3</param>
            <param name="Er">0.213299</param>
            <param name="Fr">93.0931e-12</param>
            <param name="Df">2.7454e3</param>
            <param name="Ef">0.0184577</param>
            <param name="Ff">61.0572e-12</param>
            <param name="Gr">10319.5</param>
            <param name="Hr">2.15881e-10</param>
            <param name="Gf">6012.03</param>
            <param name="Hf">8.8294e-11</param>
            <param name="R">9.3796e-18</param>
            <param name="Q">0.000269692</param>
        </model>
      </logiccell>
    </cells>
</library>
```

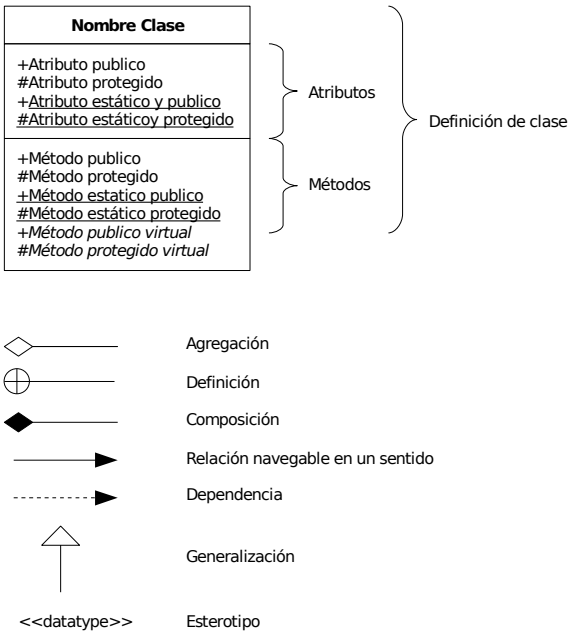
Anexo 2. Simbología de los diagramas en UML

Los diagramas de casos de usos es una técnica para capturar la funcionalidad de los sistemas.



La figura representa como ejemplo una tarea y su subdivisión en otras tareas. La tarea inicial que describe el diagrama es "*Compilar netlist*" la cual se descompone en varias. La descomposición puede ser de dos tipos "*extiende*" y "*usa*". La relaciones por extensión "*Parser Verilog*" y "*Parser VHDL*" significan que, "*Compilar netlist*" se puede hacer de estas dos formas diferentes. La relación "*usa"* asociada a la "*Librería de celdas compilada*" significa que, independientemente de usar la tarea "*Parser Verilog*" o la tarea "*Parser VHDL*", la tarea "*Compilar netlist*" usará además la "*Librería de celdas compilada*".

Por otro lado, Los diagramas de clases son la representación estática de la estructura de un programa o sistema donde, aparecen las clases representadas por cajas cuadradas y sus relaciones mediante líneas con terminadores. La forma del terminador indica el tipo de relación existente.



La figura muestra las partes existentes en la representación de una clase y los tipos de relaciones posibles. Toda clase está dividida en tres partes: nombre de clase, atributos y métodos. Los atributos y los métodos presentan diferentes modificadores de visibilidad mediante símbolos:

- **Modificador '+'**: El elemento es público, es decir, otra clase puede acceder al atributo o método.
- **Modificador '#'**: Elemento protegido, sólo las clases derivadas (hijas) pueden acceder al atributo o método.
- **Modificador '-'**: El elemento es privado, sólo la clase en la que está definida tienen acceso al mismo.

La notación también permite especificar que elementos (atributos o métodos) son estáticos, es decir, datos o métodos compartidos por todos los objetos de una determinada clase, para ello, se subraya el elemento.

Respecto a las relaciones, los extremos se decoran con números que indican la multiplicidad de la relación y, además, según el símbolo del extremo el contenido semántico de la relación en la siguiente:

- **Composición**: Relación en la que una clase forma parte de otra, es una relación fuerte, ya que al destruirse un objeto de la clase contenedora se

destruyen todos los objetos de la clase contenida. Se puede decir que la clase contenida está incrustada en la clase contenedora.

- **Agregación**: Es una relación de composición débil, tal que, una clase es contenida en otra, pero un objeto de la clase contenida puede existir aunque se destruya el objeto de la clase contenedora que lo contenía.
- **Definición**: Un tipo de datos especializado definido en la clase. Generalmente sólo usado en la clase y sus hijas, si no fuera así, hubiese sido definida globalmente.
- **Dependencia**: Relación en la que una clase usa en parte o en la totalidad un componente de software. Generalmente el componente es una librería o programa externo del cual se utiliza su funcionalidad en alguna tarea.
- **Generalización**: Relación en la que una de las clases (clase hija) se considera una especialización de la otra clase (clase padre), la clase padre se considera una generalización de la clase hija. En la práctica, esto significa que una instancia de la clase hija también es instancia de la clase padre por tanto tiene todos los atributos y métodos existentes en la clase padre, además de los propios.
- **Relación navegable**: Relación solo accesible desde una de las clases, sólo en el sentido indicado en la flecha.

Bibliografía

Bibliografía derivada del trabajo de Tesis

[BELL06][10] M. J. BELLIDO, J. JUAN, M. VALENCIA, C. J. JIMÉNEZ, D. GUERRERO, A. MILLÁN, P. RUIZ DE CLAVIJO, C. BAENA, “Logic-timming Simulation and the Degradation Delay Model”, Ed. Imperial College Press, ISBN 1-86094-589-9, 2006.

[JUAN01a] J. JUAN, M. J. BELLIDO, P. RUIZ-DE-CLAVIJO, C. BAENA, C. J. JIMENEZ, M. VALENCIA, “Switching activity evaluation of CMOS digital circuits using logic timing simulation”, Electronics Letters, Vol. 37(9), pp. 555-557, 2001.

[RUIZ01a] RUIZ-DE-CLAVIJO, P., JUAN, J., BELLIDO, M. J., ACOSTA, A. J., VALENCIA, M., "HALOTIS: High Accuracy Logic Timing Simulator with Inertial and Degradation Delay Model. ", Design, Automation and Test in Europe (DATE) Conference and Exhibition, Munich (Germany), March 2001.

[RUIZ01b] P. RUIZ-DE-CLAVIJO, M. J. BELLIDO, J. JUAN, C. BAENA, “ISS: Interactive Simulation System”, Congreso: 16th Conference on Design of Circuits and Integrated Systems (DCIS), , Porto (Portugal), pp. 410-413, 2001

[RUIZ02] P. RUIZ-DE-CLAVIJO, J. JUAN, M. J. BELLIDO, A. MILLÁN, D. GUERRERO, “Efficient and Fast Current Curve Estimation of CMOS Digital Circuits at the Logic Level”, Lecture Notes in Computer Science, Vol. 2451/2002, pp400, Springer Berlin / Heidelberg, 2002

[RUIZ03] RUIZ-DE-CLAVIJO, P. , JUAN-CHICO J. , FERNANDES, J.R , BELLIDO M. J. , MILLÁN A. , AND GUERRERO D., "Delay Degradation Effect in Current-Balanced Logic Cells", Proceedings of XVIII Conference on Design of Circuits and Integrated Systems (DCIS), , ISBN 84-87087-40-X, Ciudad Real, 2003

10. Los capítulos 6 y 7 de este libro presentan el simulador HALOTIS y los resultados obtenidos con HALOTIS respectivamente.

[RUIZ05a] P. RUIZ DE CLAVIJO VAZQUEZ, J. JUAN CHICO, M. J. BELLIDO DIAZ, A. MILLAN CALDERON, D. GUERRERO MARTOS, E. OSTUA ARANGUENA J. VIEJO CORTES, “Logic-level Fast Current Simulation for Digital CMOS Circuits“, Lecture Notes in Computer Science, Vol. 3728, pp. 425-435, Springer Berlin / Heidelberg, 2005.

[RUIZ05b] P. RUIZ-DE-CLAVIJO, M. J. BELLIDO, J. JUAN, “HALOTIS - High accurate logic timing simulator”, Proc. PhD Research In Micro-Electronics & Electronics (PRIME), Lausanne (Switzerland), 2005.

[RUIZ06] P. RUIZ DE CLAVIJO VAZQUEZ, J. JUAN CHICO, M. J. BELLIDO DIAZ, A. MILLAN CALDERON, D. GUERRERO MARTOS, E. OSTUA ARANGUENA, J. VIEJO CORTES, "Accurate logic-level Current Estimation for Digital CMOS Circuits", Journal of Low Power Electronics, Vol. 2, pp. 87-94, 2006.

Bibliografía general.

[ABBO04] A. ABBO, R. KLEIHORST, V. CHOUDHARY, AND S. L., “Power consuption of performance-scaled simd processors,” in Proc. 14th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 532–540, Springer, September 2004.

[ABRA90] M. ABRAMOVICI, M.A. BREUER, A.D. FRIEDMAN: "Digital Systems Testing and Testable Design". Ed. Computer Society press. 1990.

[ACOS00] A. J. ACOSTA, A. BARRIGA, M.J. BELLIDO, J. JUAN, M. VALENCIA, “Temporización en circuitos integrados digitales CMOS”, Ed. Marcombo, ISBN: 84-267-1246-0, 2000

[ALBU99a] EDGAR F. M. ALBUQUERQUE, MANUEL M. SILVA, “Current-Balanced Logic (CBL) Cells with Complementary Outputs”, Designs of Circuits and Integrated Systems Conference (DCIS99), Palma de Mayorca, November 1999.

[ALBU99b] EDGAR F. M. ALBUQUERQUE, MANUEL M. SILVA,

"Current-Balanced Logic for mixed signal", In Proc. IEEE International Symposium on Circuits and Systems (ISCAS99), pp. 1498-1500, December 1999.

[AUVERG90] AUVERGNE, D., AZEMARD, N., DESCHACHT, D., ROBERT, M. "Input waveform slope effects in CMOS delays. IEEE Journal of Solid-State Circuits 25 (6) (December 1990) 1588–1590

[ARTS03] B. ARTS, A. BELLU, L. BENINI, N. VAN DER ENG, M. HEIJLIGERS, E. MACII, A. MILIA, R. MARO, H. MUNK AND F. THEEUWEN, PATMOS 2003, LNCS 2799, pp196"Stadistical Power Estimation of Behavioral Descriptions", pp. 197–207, Springer, September 2003.

[ARAG99] X. ARAGONÉS, J. L. GONZÁLEZ, A. RUBIO, "Analysis and Sonlutions for Switching Noise Coupling in Mixed-Signal Ics", ISBN 0-7923-8504-7, de. Kluwer Academic Publishers, 1999.

[ARGO06] M. WULP, A. RAMIREZ, P. VANPEPERSTRAETE, A. RUECKERT, K. ODUTOLA, J. BENNETT, L. TOLKE, Copyright © 2004, 2005, 2006 , http://www.argouml.org.

[BAENA02] C. BAENA, J. JUAN, M. J. BELLIDO, P. RUIZ-DE-CLAVIJO, C. J. JIMENEZ, M. VALENCIA, "Measurement of the Switching Activity of CMOS Digital Circuits at the Gate Leve", Lecture Notes in Computer Science, Vol. 2451/2002, pp 353, Springer Berlin / Heidelberg, 2002

[BELL94] M.J. BELLIDO, "Biestables CMOS VLSI bajo entradas asíncronas: Problemas y Aplicaciones". Tesis Doctoral. Universidad de Sevilla. 1994.

[BELL00] M. J. BELLIDO DÍAZ, J. JUAN CHICO, A. J. ACOSTA, M. VALENCIA y J. L. HUERTAS, "Logical modelling of delay degradation effect in static CMOS gates". IEE Proc.-Circuits Devices Syst., Vol. 147, No. 2 April © IEE,2000.

[BELL01] M.J. BELLIDO-DÍAZ, J. JUAN-CHICO, P. RUIZ-DE-CLAVIJO, A.J. ACOSTA AND M. VALENCIA. "Gate-level Simulation of CMOS Circuits Using the IDDM Model". Proc. IEEE Int. Symposium on Circ. and Systems (ISCAS). pp. V-483-V-486. Sydney, Australia, 2001.

[BISD98] L. BISDOUNIS, S. NIKOLAIDIS, O. KOUFOPAVLOU, "Analytical Transient Response and Propagation Delay Evaluation of the CMOS Inverter for Short-Channer Devices" , IEEE J. of Solid-State Circuits, pp. 302-306, Vol. 33 no. 2, ©1998.

[BISD99] BISDOUNIS, L.; KOUFOPAVLOU, O., "Analytical modeling of short-circuit energy dissipation insubmicron CMOS structures", Proceedings of The 6th IEEE International Conference on Electronics, Circuits and Systems ICECS, Volume 3, pp. 1667-1670 vol. 3, 1999

[BRYAN85] BRYAN D., "The ISCAS '85 benchmark circuits and netlist format", The IEEE International Symposium on Circuits and Systems (ISCAS), 1985.

[BRYAN88] BRYAN, Martin. "SGML: An Author´s Guide to the Standard Generalizad Markup Languaje". Addison-Wesley, 1988, ISBN: 0-201-17535-5.

[BOGLI97] A. BOGLIOLO, L. BENINI, G. DE MICHELI, AND B. RICC, "Gate-level power and current simulation of CMOS integrated circuits", IEEE Trans. on very large scale of integration (VLSI) systems, vol. 5, pp. 473-488, December 1997

[BURCH93] R. BURCH, F. N. NAJM, P. YANG, T. N. TRICK, "A Monte Carlo Approach for Power Stimation", IEEE Transactions on Very Large Scale Integrations (VLSI) Systems, Vol. 1, No. 1, pp. 63-71, 1993

[CALV91] J. CALVO, M. VALENCIA, J.L. HUERTAS, "Metastable Operation in RS Flip-Flops". Int. J. Electronics, Vol. 70, No. 6, 1991.

[CIRIT87] M. A. CIRIT,"Estimating dynamic power consumption of CMOS circuits", Proceedings of IEEE International Conference on Computer Aided Design (ICCAD), pp. 543-537, 1987

[DAGA98] M.J. DAGA, S. TURGIS, D. AUVERGE, "Design Oriented Stardard Cell Delay Modelling", PATMOS'96., pp. 215-224, State Circuits, pp. 302-306, Vol. 33 no. 2, © Feb. 1998.

[DAGA99] DAGA, J.M.; AUVERGNE, D., "A comprehensive delay macro

modeling for submicrometer CMOS logics", IEEE Journal of Solid-State Circuits, Vol. 34, No 1, pp. 42–55, Jan 1999.

[DENG88] A.C. DENG, "Piecewise-Linear Timing Modeling for Digital CMOS Circuits", IEEE Trans. on Circuits and Systems. Vol. 35, No. 10, pp. 1330-1334, Octubre 1988.

[DESC88] D. DESCHACHT, M. ROBERT, D. AUVERGNE: "Explicit Formulation of Delays in CMOS Data Paths". IEEE J. of Solid-State Circ. Vol.23. No.5, pp. 1257-1264. Octubre 1988.

[FISH01] GEORGE S. FISHMAN, "Discrete-Event Simulation ", Springer Verlang Inc. ISBN 0-387-95160-1, 2001.

[GHOSH92] A. GHOSH, S. DEVADAS, K. KEUTZER, J. WHITE, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", Proceedings os ACM IEEE 29th Design Automayion Conference (DAC), pp. 253-259, 1992.

[GNU] GNU's Not Unix - Free Software, Free Society, http://www.gnu.org.

[HENS03] P. HENSGEN, "Umbrello UML Modeller Handbook" Copyright © 2002, 2003 Umbrello UML Modeller Authors, http://uml.sourceforge.net

[HSPICE00] Cadence Home Page, http://www.cadence.com.

[ISASI01] P. ISASI, P. MARTINEZ, D. BORRAJO, " Lenguajes, gramáticas y autómatas. Un enfoque práctico", Addison Wesley, 2001], ISBN 84-7829-014-1

[JEPP94] K.O. JEPPSON, "Modeling the Influence of the Transistor Gain Ratio and de Input-to-Output Coupling Capacitance on the CMOS Inverter Delay", IEEE J. of Solid-State Circ. Vol.29. No.6, pp. 646-654, Junio 1994.

[JIME02] R. JIMENEZ, P. PARRA, P. SANMARTIN, AND A. ACOSTA, "Analysis of high-performance flips-flops for submicron mixed-signal applications," Int. Journal of Analg Integrated Circuits ans Signal Processing, vol. 33, pp. 145–156, Nov 2002.

[JUAN97] J. JUAN-CHICO, M.J. BELLIDO, A.J. ACOSTA, A. BARRIGA, M. VALENCIA, "Delay degradation effect in submicronic CMOS

inverters", Actas Power and Timing Modelling, Optimization and Simulation (PATMOS) 1997. pp. 215-224. Louvain-la-Neuve, Bélgica, 1997.

[JUAN99] J. JUAN-CHICO, M.J. BELLIDO, A.J. ACOSTA, M. VALENCIA, "Inertial effect handling method for CMOS digital IC simulation" , Electronics Letters, pp. 2028-2030, Vol. 35, ISSN:0013-5194, © 1999 IEE.

[JUAN00a] J. JUAN-CHICO, "Degradación del Retraso de Propagación en Puertas lógicas CMOS VLSI". Tesis Doctoral. Universidad de Sevilla, 2000.

[JUAN00b] J. JUAN-CHICO, P. RUIZ-DE-CLAVIJO, M.J. BELLIDO, A.J. ACOSTA, M. VALENCIA, "Inertial and Degradation Delay Model for CMOS Logic Gates" , The IEEE International Symposium on Circuits and Systems (ISCAS), pp. 459-462, Vol. 1, ISBN:0-7803-5485-0, ©2001 IEEE.

[JUAN00c] J. JUAN, M. J. BELLIDO, P. RUIZ-DE-CLAVIJO, A. J. ACOSTA, M. VALENCIA, “Degradation delay model extension to CMOS gates”, Lecture Notes in Computer Science (ISSN 03029743), Vol. 1918 pp. 149-158, 2000.

[JUAN01b] J. JUAN-CHICO, M.J. BELLIDO, P. RUIZ-DE-CLAVIJO, C. BAENA, M. VALENCIA, “AUTODDM: AUTOmatic characterization tool for the Delay Degradation Model”, Proceedings of International Conference on Electronic, Circuits and Systems, (ICECS), pp 1631-1634, 2001.

[JUAN01c] J. JUAN, M. J. BELLIDO, P. RUIZ-DE-CLAVIJO, C. BAENA, M. VALENCIA, “DDM characterization methodology and automation”, Proc. 11th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 5.2.1-5.2.10, 2001.

[KOBR01] CRIS KOBRYN, "Introduction to UML: Structural and Use Case Modeling: Object Modeling with OMG", UML Tutorial Series, OMG and Contributions., © 1999,2001.

[LATTA04] E. LATTANZI, A. ACQUAVIVA, AND B. A., “Run-time software monitor of the power consumption of wireless network interface cards,” in Proc. 14th International Workshop on Power and

Timing Modeling, Optimization and Simulation (PATMOS), pp. 352–361, Springer, September 2004.

[LEVI92] JOHN R. LEVINE, TONY MASON, DOUG BROWN, "Lex & Yacc", 1992, ISBN: 1-56592-000-7, 2, Editado por O'Reilly & Assiciates, Inc., ©1990.

[LIBXML] D. VEILLARD, “The XML C parser and toolkit of Gnome”, http://xmlsoft.org/

[MACH00] "Mach TA User's and Reference Manual, V3.0_1.1", Mentor Graphics, 2000.

[MAUR01] P. MAURINE, R. POIRIER, N. AZÉMARD, AND D. AUVERGNE, “Switching current modeling in cmos inverter for speed and power estimation”, 16th Conference on Design of Circuits and IntegratedSystems (DCIS) ,pp. 618-622, Noviembre 2001.

[MCGE91] P.C. MCGEER, R.K. BRAYTON, "Integrating functional and temporal domains in logic design", Ed. Kluwer Academic Publishers, 1991.

[MELC92] E. MELCHER, W. RÖTHIG, M. DANA, "Multiple Input Transitions in CMOS Gates", Microprocessing and Microprogramming 35 (1992) pp. 683-690. North Holland.

[MILLAN02] A. MILLAN CALDERON; J. JUAN CHICO; M.J. BELLIDO DIAZ; P. RUIZ DE CLAVIJO VAZQUEZ; D.GUERRERO MARTOS, “Characterization of normal propagation delay for delay degradation model (DDM)”, Lecture Notes in Computer Science (LNCS) , ISSN 0302-9743, Vol. 2451, pp. 477-486, 2002.

[MILLAN03] D. GUERRERO, M. J. BELLIDO, J. JUAN, A. MILLAN, P. RUIZ-DE-CLAVIJO, “Estimation of floating cube delay using transistor path computational delay models for CMOS circuits”, 18th Conference on Design of Circuits and Integrated Systems, pp. 95-99, Ciudad Real (Spain), 2003.

[MILLAN04] A. MILLAN, J. JUAN, M. J. BELLIDO, P. RUIZ-DE-CLAVIJO, D. GUERRERO, E. OSTUA, “Signal sampling based transition modeling for digital gates characterization”, Lecture Notes in Computer Science, Vol 3254, pp 829-837, 2004.

[MIZCO86] A. MIZCO, "Digital logic testing and simulation". Ed. Harper and Row, 1986.

[NAJM94] F. N. NAJM, “A Survey of Power Estimation Techniques in VLSI Circuits”, IEEE Transactions on VLSI Systems, Vol. 13, No 10, pp. 446-465, 1994]

[NAJM91] F. N. NAJM, “Transition density, a stochastic measure of activity in digital circuits”, Proceedings od ACM IEE 28th Design Automation Conference (DAC), pp 644-649, 1991

[NIKO99] S. NIKOLAIDIS AND A. CHATZIGEORGIOU, “Analytical estimation of propagation delay and short-circuit power dissipation in CMOS gates”, International journal of circuit theory and applications, vol. 27, pp. 375-392, 1999

[RABE98] RABE, D.; JOCHENS, G.; KRUSE, L.; NEBEL, W., “Power-simulation of cell based ASICs: accuracy- and performancetrade-offs”, Proceedings of Design, Automation and Test in Europe, 1998., pp. 356 – 361, Feb 1998.

[RAO89] V.S. RAO, D.V. OERHAUSER, N.T. TIMOTHY, N.H. IBRAHIM, "Switch-level timing simulation of MOS VLSI circuits". Ed. Kluwer Academic Publishers, 1989.

[RUMB96] JAMES RUMBAUGH, MICHAEL BLAHA, WILLIAM PIEMERLANI, FREDERICK EDDY, WILLIAM LENSEN. "Modelado y Diseño Orientado a Objetos". Prentice Hall, Madrid ©1996.

[RUMB00] JAMES RUMBAUGH, IVAR JACOBSON, GRADY BOOCH. "Lenguaje Unificado de Modelado. Manual de Referencia." Addison Wesley. Primera edición ISBN: 84-7829-037-0 ©2000.

[SCHI99] H. SCHILDT, "STL Programming from the Ground Up", ISBN: 0-07-882507-5, ©1999, Editado por McGraw-Hill.

[SEEN04] E. SEEN, J. LAURENT, N. JULIEN, AND M. E., “Softexplorer: Estimation, Characterization and Optimization of the Power and Energy Consumption at the Algoritmit Level,” Proc. 14th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 342–351, Springer, September 2004.

[SWEET01] D. SWEET, M. ETTRICH, “KDE 2.0 Development”, © 2001 Sams Publishing, ISBN 0-672-31891-1.

[SYNOPSYS] SYNOPSYS, INC., “Primepower reference manual.,” in http://www.synopsys.com

[TURGIS98] S. Turgis, D. Auvergne, “A Novel Macromodel for Power Estimation in CMOS Structures”, IEE Trans. Computer-Aided-Design, vol. 17, nº 11, pp 1090-1098, Nov. 1998

[UMLN97] UML Notation Guide, Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjectTime, Platinum Technology, Ptech, Taskon, Reich Technologies, y Softeam, 1 September 1997, ©1997.

[UMLS97] UML Summary version 1.1. Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjectTime, Platinum Technology, Ptech, Taskon, Reich Technologies, y Softeam, ©September 1997.

[UNGE69] S. H. UNGER, “Asynchronous Sequential Switching Circuits”, de. Wiley-Inerscience, 1969.

[UNGE89] S. H. UNGER, “The essence of logic circuits”, Pretince-Hall Inernational Inc., 1989.

[VALE86] M. VALENCIA, "Modelado discreto, descripción lógica y simulación de fenómenos metaestables en circuitos secuenciales". Tesis Doctoral, Departamento de Electrónica y Electromagnetismo, Universidad de Sevilla, 1986.

[VERI] VERITOOLS INC., “Powertool reference manual.,” in http://www.veritools.com.

[WENG03] WENG FOOK LEE,.”Verilog Conding for Logic Synthesis”, John Wiley & Sons, Inc. 2003, ISBN-0-471-42976

[WOLF94] W. WOLF, “Modern VLSI desing: A system Approach”, Ed. Prentice-Hall, 1994. ISBN-013-588377-6

[XILINXa] XILINX, EMBARCADERO TECHNOLOGIES INC., 425 Market Street, Suite 425 San Francisco CA 94105

http://www.embarcadero.com,
http://www.uml.org/uml_success_stories/

[XILINXb] XILINX, EMBARCADERO TECHNOLOGIES INC., "Xpower reference manual.," in http://www.xilinx.com