Dedico el proyecto realizado a mis padres, Juan y Carmen, a mis hermanos, Carlos y Carmen, y al resto de mi familia.

Quiero agradecer la inspiración que me han proporcionado la plantilla de la UD Las Palmas, el tenis español, Javier Reverte, Joseph Conrad, Vargas Llosa, U2, Garbage, Charlize Theron, Laetitia Casta, Patrick Bateman, el Nota,....

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE
INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Entorno gráfico de simulación de tiempos de propagación basado en el análisis de la sensibilidad

Presidente: Roberto Esper-Chaín

Secretario: Pedro Hernández

Vocal:

Tutores: JAVIER GARCIA F. JAVIER DEL PINO

Autor: JUAN PULIDO

NOTA : SOBRESALIENTE 10. MH.

ESPECIALIDAD : EQUIPOS ELECTRÓNICOS
TUTOR/ES : Javier García García

Fco. Javier del Pino Suárez

AUTOR : Juan Marcos Pulido González

FECHA : Junio 2001

Índice

1. Introducción

En esta memoria se describe el programa diseñado para la simulación de tiempos de propagación basado en el análisis de la sensibilidad. A la aplicación se le denomina SiDSen, Simulate Delay based on Sensitivity Analysis. Este nombre se usará a lo largo de la memoria para hacer referencia al programa. El modelo temporal utilizado en el programa se aplica a la familia lógica directamente acoplada *DCFL*, *Direct Coupled FET Logic*, implementada con *HFETs*, *Heteroestructure Field Effect Transistors*, en Arseniuro de Galio, GaAs.

En el diseño de circuitos electrónicos digitales ha cobrado gran importancia el uso de modelos que simulen el funcionamiento de los mismos, y que realicen una estimación de los tiempos de propagación. Para evaluar la efectividad de los modelos de análisis temporal se pueden utilizar dos factores: el tiempo de computación consumido y la precisión de los resultados de la simulación.

El modelo que representa SPICE proporciona los resultados más exactos en la estimación de los tiempos de retardo en circuitos lógicos. Sin embargo, requiere del empleo de una gran cantidad de tiempo de CPU.

El modelo temporal basado en el análisis de la sensibilidad propone unas ecuaciones de modelado cuya aplicación en la simulación de circuitos lógicos supone un considerable ahorro en el tiempo de computación empleado. Consiste en un modelo de funcionamiento de la familia lógica DCFL implementada con HFETs explotando el concepto de sensibilidad. A través de simulaciones SPICE se determina la sensibilidad de los tiempos de propagación con respecto a los parámetros de modelado de tipo geométrico, tecnológico y eléctrico. De esta forma se obtiene un modelo que va a permitir al diseñador optimizar el circuito que simula.

Se ha dotado al programa de la posibilidad de introducir para la simulación de los tiempos de retardo ecuaciones externamente. Esta capacidad permite seguir desarrollando el modelo temporal con la incorporación de parámetros no considerados hasta ahora, como

puede ser la temperatura a la que se encuentran los dispositivos. Actualmente se considera que el circuito está operando a temperatura ambiente (27°C).

Se ha añadido a la aplicación herramientas de traducción de ficheros a, y desde, formato SPICE, en particular para la versión IAFSPICE, pero fácilmente aplicable a otras versiones. Estas utilidades representan elementos que van a permitir comparar ambos modelos. Un circuito lógico cuyo diseño se haya desarrollado mediante el formato del programa puede ser finalmente optimizado y ajustado trasladándolo a SPICE mediante la herramienta de traducción.

El programa se ha creado bajo el entorno de programación Delphi3, con lo que se obtiene una aplicación con soporte e integración completas para Windows, y que presenta las características distintivas de los programas ejecutables bajo Windows. Para crear una aplicación bajo este entorno es necesario conocer la base Pascal del mismo y emplear sus componentes predefinidos. Delphi3 contiene herramientas de diseño visuales que permiten crear los elementos visuales de una aplicación de forma rápida y sencilla. Colocando los componentes y controles sobre la ficha, ventana para el futuro usuario, se genera la interfaz de usuario de la aplicación. Automáticamente Delphi3 añade el código Pascal necesario para soportar tales elementos. La apariencia de los controles y componentes se establece por parte del programador fijando sus propiedades, y el código se diseña para dar respuesta a los eventos producidos por la acción del usuario sobre ellos.

El contenido de esta memoria se ha dividido en nueve capítulos, incluyendo este primer capítulo de introducción, y un anexo o apéndice. En el capítulo 2 se presenta el modelo temporal utilizado, fundamento de la aplicación. Contiene una breve descripción de la familia lógica DCFL y del concepto de sensibilidad, presentándose la metodología seguida para obtener las ecuaciones de modelado.

El capítulo 3 se dedica a describir de forma general el programa creado. Se muestran los formatos establecidos para los distintos tipos de ficheros con que trabaja la aplicación, y se informa al lector de las capacidades y herramientas del programa.

En el capítulo 4 se hace una descripción detallada del código implementado para realizar el programa. Se analiza el código diseñado y se presentan diagramas de flujo del mismo.

En el capítulo 5 se realiza un estudio comparativo entre el programa creado e IAFSPICE ejecutando diversas simulaciones. Se utilizan como elementos de comparación la precisión de los resultados y el tiempo de CPU empleado en la estimación de los tiempos de retardo.

El capítulo 6 se dedica a la presentación de conclusiones, y se proponen posibles trabajos futuros.

El capítulo 7 se dedica a la memoria económica. El capítulo 8 contiene la bibliografía empleada.

Por último, el anexo recoge el código diseñado para crear el programa.

2. Modelo temporal

2.1. Introducción

En éste capítulo se describe la metodología que ha conducido al desarrollo del modelo temporal basado en el análisis de la sensibilidad [1], [2], aplicado a la familia lógica *DCFL, Direct Coupled FET Logic,* implementada con *HFETs, Heteroestructure Field Effect Transistors.*

El aumento en complejidad de los circuitos electrónicos ha hecho necesaria la utilización de modelos temporales que simulen su funcionamiento. Se han desarrollado modelos temporales basados en diversas estrategias de modelado como la aproximación a redes RC de los transistores, modelos constituidos por tablas de datos, modelos conformados por expresiones basadas en aspectos físicos, etc.

Un modelo que incorpore expresiones derivadas de aspectos tecnológicos, geométricos y eléctricos cuenta con la ventaja de permitir al diseñador optimizar el circuito que simula. Es deseable que el modelo emplee el número mínimo de parámetros necesario para simular de forma simple y precisa la respuesta temporal de un circuito. El modelo temporal basado en el análisis de la sensibilidad desarrolla ecuaciones que cumplen con estos preceptos. El procedimiento para obtener las ecuaciones de modelado consiste en la determinación, por medio de simulaciones SPICE, de la sensibilidad de ciertas variables dependientes, tiempos de propagación en este caso, con respecto a las variaciones de variables independientes, parámetros de modelado geométricos, tecnológicos y eléctricos. La sensibilidad permite detectar qué variables independientes deben ser incorporadas al modelo, siendo el fundamento de las ecuaciones del mismo.

En el punto 2.2 se realiza una descripción de la familia lógica DCFL. En los dos apartados siguientes se introduce el análisis de la sensibilidad y se presenta el desarrollo del modelo temporal. En el quinto punto del capítulo se expone el estudio realizado para aplicar las ecuaciones de modelado a la puerta lógica NOR de dos entradas de la familia lógica DCFL.

2.2. Familia lógica DCFL con HFETs

La familia lógica DCFL es la más simple y su uso es el más extendido en la tecnología de Arseniuro de Galio, GaAs. El inversor básico DCFL es similar al inversor NMOS [3]. Está formado por dos transistores *HFET*, siendo de enriquecimiento el transistor que conmuta, y de empobrecimiento la carga activa. En la siguiente figura se muestra dicho inversor:

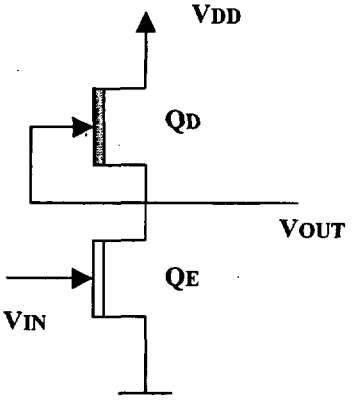


Figura 2.1. Inversor DCFL.

La tensión de alimentación, V_{DD}, es de 1.5 V. La longitud de ambos transistores, que está prefijada por la tecnología [4], [5], es de 0.3 μm. Las anchuras de los transistores pueden ser modificadas dentro de cierto intervalo. Se define la relación de aspecto, β, como el cociente entre la anchura del transistor de enriquecimiento, W_E, y la anchura del transistor de deplexión, W_D, en el inversor DCFL.

$$\beta = \frac{W_E}{W_D} \qquad (1)$$

Esta variable es un parámetro de gran importancia en el funcionamiento de la puerta lógica. Se ha comprobado que la curva característica posee una fuerte dependencia con el valor de la relación de aspecto, siendo independiente de los distintos valores de anchuras [6].

En cuanto a su funcionamiento, cuando a la entrada haya una tensión que corresponda al nivel lógico alto, el transistor de enriquecimiento se encuentra en conducción, y el nudo de salida se sitúa en el nivel lógico bajo. Si lo que se tiene a la entrada es una tensión correspondiente al nivel lógico bajo, menor que la tensión umbral, V_{TE}, el transistor se encuentra en corte, y la salida está en nivel alto.

2.3. Introducción al análisis de la sensibilidad

El modelo se desarrolla empleando el análisis de la sensibilidad observando la variación de la magnitud de interés, en este caso tiempo de propagación, con respecto a variaciones porcentuales de diversos parámetros.

Sea m_j la magnitud de interés, por ejemplo: tensiones de salida, tiempos de propagación, o potencia disipada; se tiene entonces m_j como variable dependiente de una serie de parámetros o variables independientes, p_i, de tipo geométrico, tecnológico o eléctrico, como pueden ser la relación de aspecto, β, tensión de alimentación, tensión umbral de los transistores, entre otros.

La sensibilidad de una variable dependiente con respecto al valor nominal de un parámetro se define como,

$$S_{pi}^{mj} = \frac{\frac{m_j - m_{j0}}{m_{j0}}}{\frac{p_i - p_{i0}}{p_{i0}}} \qquad (2)$$

donde p_i representa un valor del parámetro de modelado, p_{i0} es el correspondiente valor nominal, m_j es el valor de la magnitud objeto de estudio, y m_{j0} es el valor de dicha magnitud cuando el parámetro toma su valor nominal. La sensibilidad es el tanto por uno de la variación relativa de m_j con la variación relativa de p_i.

El valor de m_{j0} se determina por simulación cuando el parámetro es p_{i0}, y tras hacerlo variar hasta p_i se obtiene m_j.

Si la magnitud permanece constante ante las variaciones en el parámetro se obtiene que $S_p^m = 0$, y se dice que la magnitud mj es estrictamente insensible al parámetro p. Si por contra, S_p^m presenta un valor no nulo, el valor de la sensibilidad puede encontrarse en uno de los siguientes tres intervalos definidos por dos valores de sensibilidad crítica, Sc y SM. Se define Sc como la sensibilidad crítica mínima, y se considera que la magnitud mj es insensible a pi si $S_p^m \leq S_C$. El valor de Sc se selecciona atendiendo a un compromiso entre sencillez y precisión del modelo resultante, con objeto de minimizar el número de parámetros necesarios para determinar el valor de la magnitud. Si mj es insensible a pi puede considerarse que mj no depende significativamente del parámetro pi.

Si S_p^m es un valor alto, mayor o igual que SM, sensibilidad crítica máxima, se considera que la magnitud mj es muy sensible a pi. El valor de SM se escoge como el caso en que variaciones porcentuales en el parámetro producen una variación equivalente en la magnitud, esto es, SM = 1.

Este análisis se presenta en modo esquemático en la Tabla 2.I:

Tabla 2.I. Intervalos de sensibilidad.

$S_p^m = 0$	m es estrictamente insensible a p
$S_p^m \leq S_C$	m es insensible a p
$S_C < S_p^m < S_M$	m es sensible a p
$S_p^m \geq S_M$	m es muy sensible a p

Sea mj una magnitud sensible al conjunto de parámetros $r_1, r_2, \ldots, r_K$. En ese caso, se asume que mj se puede expresar como una función sólo en dichos parámetros, $m_j = F(r_1, r_2, \ldots, r_K)$ y se desarrolla en serie de Taylor en torno al hiperpunto de los valores nominales

($r_{10}, r_{20}, \ldots, r_{K0}$). Si solamente se toman en consideración los términos de primer orden de la serie, m_j se puede expresar como una función lineal de los parámetros con coeficientes proporcionales a la sensibilidad para el valor nominal. Esta aproximación es aceptable si la magnitud no es muy sensible al parámetro, $S_p^m < S_M$.

Partiendo de las consideraciones previas, se escribe la ecuación general de modelado basado en la sensibilidad.

$$m_j = m_{j0}[1 + \sum_{i=1}^{k} S_{r_{i0}}^{m_{j0}} \frac{r_i - r_{i0}}{r_{i0}}] \qquad (3)$$

Esta expresión es válida si la sensibilidad pertenece al intervalo [Sc, SM), donde m_{j0} es el valor nominal de la magnitud m_j, y $S_{r_{i0}}^{m_{j0}}$ es el valor de la sensibilidad para el valor nominal del parámetro, r_{i0}.

El modelo propuesto precisa del valor de la sensibilidad nominal $S_{r_{i0}}^{m_{j0}}$ de la variable dependiente para cada uno de los parámetros a los que es sensible. Se obtiene evaluando el límite,

$$S_{r_{i0}}^{m_{j0}} = \lim_{r_i \to r_{i0}} S_{r_i}^{m_j} \qquad (4)$$

para lo cual es necesario conocer la dependencia de m_j con respecto a la variable independiente.

Si alguno de los parámetros de modelado, p_i, no cumple las condiciones citadas, se debe realizar un estudio detallado del parámetro en cuestión, por lo que la ecuación (3) pasa a ser:

$$m_j = m_{j0}[1 + \sum_{i=1}^{k} S_{r_{i0}}^{m_{j0}} \frac{r_i - r_{i0}}{r_{i0}}] F(p_x) \qquad (5)$$

donde F(px) es función de un parámetro muy sensible.

2.4. Modelo temporal

En este apartado se utilizan las ecuaciones de modelado basadas en la sensibilidad para definir el modelo temporal. En este caso la variable dependiente corresponde al tiempo de propagación t_p, el tiempo de propagación de alta a baja, t_{pHL}, y el tiempo de propagación de la transición de baja a alta, t_{pLH}. Se consideran variables independientes todos los parámetros de modelado SPICE, así como la relación de aspecto, β, el fan-out y los tiempos de subida/bajada de las señales de excitación. Dado que el desarrollo del modelo se fundamenta en la simulación, un requisito evidente es que SPICE incorpore modelos eléctricos fiables. Se pueden añadir otras variables independientes, como la temperatura, pero al no estar aún convenientemente descrita [5] se descarta su modelado. Para el resto de las variables independientes se supone tácitamente que la precisión de los modelos de SPICE es infinita.

Las variables independientes se hacen variar dentro de intervalos adecuados que es necesario fijar. Del conjunto de variables señaladas algunas no presentan grandes variaciones porcentuales, lo que da validez a la hipótesis de dependencia lineal. Sin embargo, otras pueden llegar a presentar grandes variaciones, siendo preciso un estudio previo detallado.

Los parámetros de modelado son considerados como pertenecientes al grupo de los que no presentan grandes variaciones porcentuales, permitiendo, aún así, variaciones en un intervalo amplio del ±20%. El valor nominal de la relación de aspecto se fija mediante un proceso de optimización de la familia lógica. La lógica DCFL con HFETs es rígida, de forma que, una vez definidas las dimensiones óptimas de las puertas, si éstas varían se degrada significativamente la respuesta. Las dimensiones de la puerta se mantienen en valores en el entorno del valor nominal.

Por lo que respecta a los tiempos de subida/bajada de las señales de excitación, se ha comprobado que aplicando una señal rampa muy rápida, o muy lenta, a la puerta de entrada se alcanzan valores similares después de dos o tres puertas. Consecuentemente se puede considerar el tiempo de propagación insensible a los tiempos de subida/bajada para circuitos formados por varias puertas.

El intervalo en el que varía el fan-out, FO, se fija atendiendo a los márgenes de ruido. Se considera que el máximo fan-out para la familia lógica DCFL implementada con HFETs es 4 [1].

El modelo fija el valor de la sensibilidad crítica, Sc, para los parámetros de modelado a un valor de 0.4, excepto para β, relación de aspecto, y el fan-out, ya que al ser parámetros de diseño requieren un mayor control y los modela siempre.

Una vez definidos los intervalos donde pueden moverse las variables independientes, así como las sensibilidades, se obtienen mediante simulaciones SPICE los tiempos de propagación t_p, t_{pHL} y t_{pLH},

$$t_p = \frac{t_{pHL} + t_{pLH}}{2} \qquad (6)$$

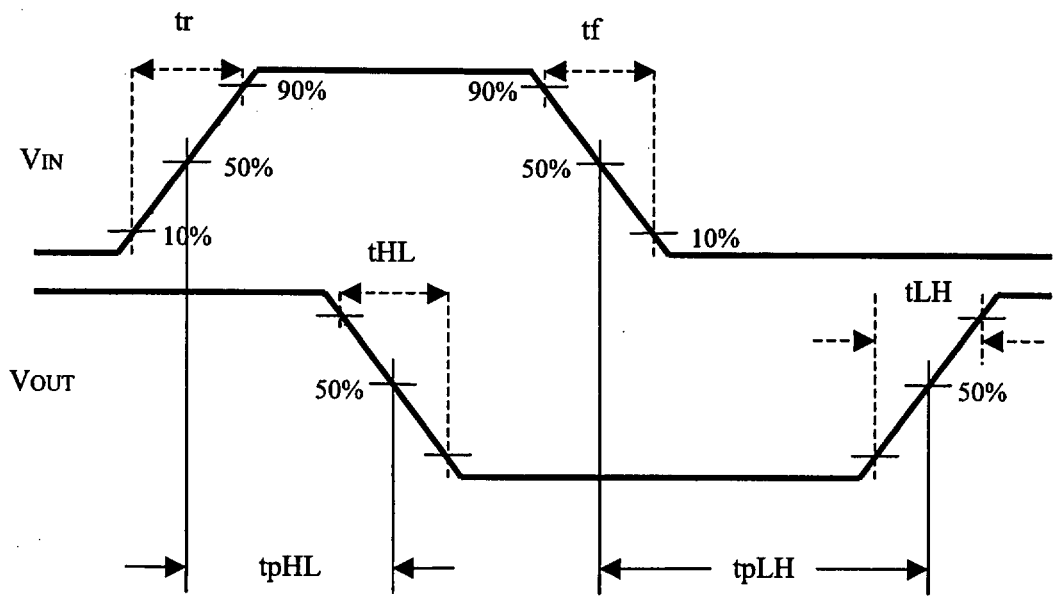


Figura 2.2. Retardos de propagación tpHL y tpLH para un inversor.

Se define el retardo en las transiciones, t_{pHL} y t_{pLH}, como el intervalo de tiempo entre el 50% de la señal de entrada y el 50% de la señal de salida según la transición sea del nivel lógico alto al bajo o viceversa respectivamente, como se puede ver en la Figura 2.2. Los tiempos de propagación de alta a baja y de baja a alta, t_{pHL} y t_{pLH}, tienen diferentes

valores. En la Figura 2.2 también se define el tiempo de subida, t_r, *rise time* y el tiempo de bajada, t_f, *fall time*, como el tiempo desde el 10% al 90% medido sobre la señal de entrada cuando sube y baja respectivamente. Análogamente en la señal de salida se obtienen los equivalentes que reciben el nombre de t_{LH} y t_{HL}.

A partir de los resultados de las simulaciones SPICE se obtiene la sensibilidad respecto a cada uno de los parámetros aplicando la ecuación (2).

Una vez conocida la sensibilidad, la sensibilidad nominal se obtiene resolviendo el límite de la ecuación (4).

El tiempo de propagación, tp, no es función de parámetros con sensibilidad crítica máxima o superior, lo que da validez a la aproximación lineal. Aplicando la ecuación (3) se obtiene.

$$tp \approx tp_0 \cdot [1 + 0.17(\beta - 2) - 0.89(V_{TE} - 0.55) - 0.18(C_E - 2.8) + 0.2(K_D - 1.6) + 0.1(G_E - 6) + 0.42(FO - 1)] \qquad (7)$$

Para los casos de t_{pHL} y t_{pLH} se han encontrado parámetros para los cuales las magnitudes resultan muy sensibles. En estos casos la dependencia de la magnitud respecto a la variable independiente da lugar a funciones no lineales.

$$t_{pHL} \approx t_{pHL0}[1 + 0.18(R_{SE} - 6) - 0.1(R_{SD} - 5) - 0.2(V_{SBD} - 1.5) - 0.2(\delta_D - 1.9) - 4.81(V_{TD} + 0.1) - 1.08(K_E - 2.5) + 2.33(n_E - 1.5) - 1.42(\beta - 2) + 0.86(FO - 1)] (-0.91K_D^2 + 4.29K_D - 3.54)(2.56V_{DD}^3 - 12.94V_{DD}^2 + 21.89V_{DD} - 11.35) (-46.37V_{TE}^2 + 64.26V_{TE} - 20.35)(-0.89C_E^2 + 5.85C_E - 8.35) \qquad (8)$$

$$t_{pLH} \approx t_{pLH0}[1 - 0.16(V_{DD} - 1.5) + 0.18(K_E - 2.5) - 0.3(n_E - 1.5) + 0.56(\beta - 2) + 0.31(FO - 1)]\left(\frac{1.6}{K_D}\right)\left(\frac{2.8}{C_E}\right)\left(\frac{1.2}{V_{TE}^2} - \frac{3}{V_{TE}} + 2.5\right) \qquad (9)$$

2.4.1. Fan-out

Inicialmente se ha considerado el fan-out como el número de puertas conectado al nudo de salida de una puerta lógica. El estudio de la respuesta temporal con respecto al fan-out ha determinado que el tiempo de propagación muestra una mayor dependencia respecto a la relación entre los valores de β de las distintas puertas que del número de puertas conectadas. Este hecho lleva a proponer un fan-out que viene dado por la relación entre los valores de β de la carga y el *driver*:

$$FO = \sum_{i=1}^{n} \frac{\beta_{Li}}{\beta_d} \quad (10)$$

donde β_{Li} corresponde al valor de la relación de aspecto de cada puerta que actúa como carga, y β_d es la relación de aspecto de la puerta lógica que hace de *driver*.

Con esta definición de fan-out se obtienen valores reales positivos, que en determinados casos resultan inferiores a la unidad.

Las curvas características que presentan los tiempos de propagación de las transiciones del nivel bajo a alto, t_{pLH}, y del nivel alto a bajo, t_{pHL}, para valores de fan-out mayores que la unidad no pueden ser representadas por las ecuaciones de modelado (8) y (9). Se han desarrollado ecuaciones para estimar el tiempo de retardo cuando el valor del fan-out, FO, supere el valor 1:

$$\begin{aligned} t_{pHL} \approx t_{pHL0}[1 &+ 0.18(R_{SE} - 6) - 0.1(R_{SD} - 5) - 0.2(V_{SBD} - 1.5) - 0.2(\delta_D - 1.9) \\ &- 4.81(V_{TD} + 0.1) - 1.08(K_E - 2.5) + 2.33(n_E - 1.5) - 1.42(\beta - 2)] \\ &(-0.91K_D^2 + 4.29K_D - 3.54)(2.56V_{DD}^3 - 12.94V_{DD}^2 + 21.89V_{DD} - 11.35) \\ &(-46.37V_{TE}^2 + 64.26V_{TE} - 20.35)(-0.89C_E^2 + 5.85C_E - 8.35) + \left(\frac{2.3207FO + 8.2016}{t_{pHL0}}\right) \end{aligned}$$

(11)

$$t_{pLH} \approx t_{pLH0}[1-0.16(V_{DD}-1.5)+0.18(K_E-2.5)-0.3(n_E-1.5)+0.56(\beta-2)]$$
$$\left(\frac{1.6}{K_D}\right)\left(\frac{2.8}{C_E}\right)\left(\frac{1.2}{V_{TE}^2}-\frac{3}{V_{TE}}+2.5\right)+\left(\frac{14.88FO+0.46}{t_{pLH0}}\right) \qquad (12)$$

Para la transición del nivel bajo a alto se ha obtenido una única ecuación válida para todos los valores de FO, pero por motivos de simetría el programa emplea las ecuaciones (11) y (12) para los casos de fan-out mayor a la unidad.

2.5. Puerta lógica NOR de dos entradas

Los circuitos lógicos utilizan inversores y puertas NOR para realizar sus funciones. Se ha modelado la puerta NOR de dos entradas implementada mediante lógica DCFL, que puede verse en la Figura 2.3. En esta sección se presenta el estudio realizado para obtener el modelo de la puerta NOR de dos entradas.

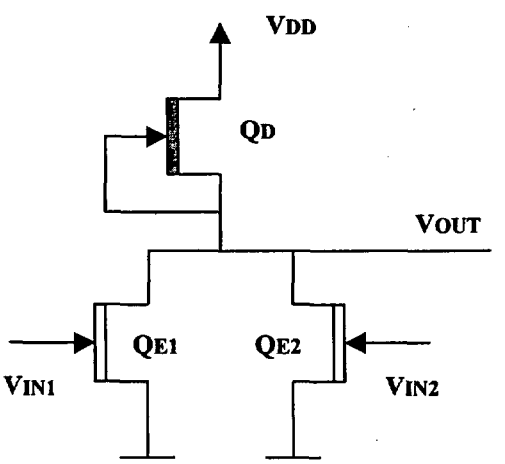


Figura 2.3. NOR2 DCFL.

El funcionamiento de esta puerta lógica es análogo al comportamiento del inversor DCFL. En este caso existen dos transistores de enriquecimiento Q_{E1} y Q_{E2}, siendo Q_D el transistor de deplexión. Los transistores de enriquecimiento se mantienen en corte mientras la tensión de entrada no supere el valor de la tensión umbral del transistor, correspondiendo la tensión de salida al nivel lógico alto. Si se aumenta la tensión hasta superar la tensión umbral, los Q_E empiezan a conducir haciendo que se alcance el nivel lógico bajo cuando los transistores de enriquecimiento entren en saturación. Una vez alcanzada esta situación, mientras una de las entradas esté en el nivel lógico alto la salida de la puerta se mantiene al nivel lógico bajo. De la misma manera si las dos entradas se encuentran en un nivel lógico bajo la salida es el uno lógico y éste pasa al cero lógico cuando una de las entradas pase de 0 a 1. Por lo que se puede reducir el estudio lógico de la puerta NOR como si se tratara de un inversor.

El caso que queda por estudiar es cuando ambas señales de entrada cambian a la misma vez o en un intervalo de tiempo cercano. El modelo considera que al producirse esta circunstancia se origina una 'colisión' de las señales que atacan la entrada de la puerta lógica.

2.5.1. Colisión

La existencia de casos de colisión hace preciso estudiar en qué medida sigue siendo válida la aproximación al tratamiento de la puerta NOR2 como inversor.

En la colisión de 0 a 1 la salida cambia con la primera entrada en conmutar. En la colisión de 1 a 0 para el cambio lógico en la salida han de pasar ambas entradas a nivel bajo. La señal que más tarda en cambiar es la que define el paso de la transición de bajo a alto en la salida.

Para ambos tipos de colisión mediante simulaciones SPICE se ha estudiado el comportamiento temporal de la puerta NOR2, y se han medido el t_{pHL} y t_{pLH}, con respecto a la influencia de la separación de las señales de entrada. Se ha asignado a ambas señales de entrada niveles de tensión y pendiente idénticos. Inicialmente se simula el caso ideal en el

que ambas entradas coinciden, para a continuación realizar simulaciones en las que se retarda una señal respecto de la otra.

Tras el análisis se ha determinado que la puerta lógica NOR2 puede ser sustituida por un inversor equivalente, cuya relación de aspecto se encuentra en el intervalo $W_E/W_D<\beta<2\cdot W_E/W_D$. De esta forma se encuentran relaciones de aspecto que producen el mismo tiempo de propagación que la separación de las señales a la entrada, tanto en transiciones de bajada como de subida. Así se establece que la relación de aspecto equivalente, β_{eq}, es función del intervalo de tiempo, ϕ, que están separadas las señales de entrada.

Se obtiene una expresión que relaciona la separación de las señales a la entrada de la NOR con la relación de aspecto del inversor equivalente. El modelo establece que a partir de una separación de las señales dada, 30 ps, se considera que el inversor equivalente tiene una relación de aspecto constante, $\beta_{eqLH} = 3.5$ y $\beta_{eqHL} = 2.3$. La ecuación (13) da la expresión de la relación de aspecto para estos inversores.

$$\beta_{eq} = \frac{F(\phi)}{\beta_0}\beta_m \qquad (13)$$

donde β_0 es la relación de aspecto de la NOR nominal y β_m es la relación de aspecto de la NOR medida, y la función $F(\phi)$ viene dada por:

$$F_{pHL}(\phi) \approx 0.0008\phi^2 - 0.052\phi + 2.94 \qquad (14)$$

$$F_{pLH}(\phi) \approx 0.0002\phi^2 - 0.009\phi + 3.6 \qquad (15)$$

donde ϕ es la distancia de separación de las señales de entrada medida en picosegundos.

En ambos casos el tiempo de propagación se calcula mediante las expresiones:

$$t_{pHL} = -1.96\beta_{eq}^2 - 3.96\beta_{eq} + 21.87 \qquad (16)$$

$$t_{pLH} = 12.1\beta_{eq}^2 - 39.78\beta_{eq} + 55.54 \qquad (17)$$

3. Descripción general del programa

3.1 Introducción

Se ha desarrollado una aplicación que implementa el modelo temporal basado en el análisis de la sensibilidad presentado en el Capítulo 2. Se consigue con ello reducir el tiempo de computación con respecto a las simulaciones SPICE, manteniendo el nivel de exactitud en la estimación del tiempo de propagación.

El programa utiliza tres ficheros que se deben definir previamente a la ejecución de la simulación. Estos ficheros se describen con más detalle en el punto 3.2.2. El fichero donde se representa el esquemático del circuito a simular tendrá la extensión 'net' de *netlist*. En este fichero se enumeran las puertas lógicas especificando los nudos a los que están conectadas y la β, relación de aspecto, que tienen. Los valores de los parámetros de modelado que se utilizan en las expresiones se definen en el fichero de extensión 'dat'; estos datos son los parámetros tecnológicos propios de los transistores HFETs y la tensión de alimentación, V_{DD}. El tercer fichero necesario para realizar una simulación será el fichero de las entradas, con la extensión 'ent', donde se precisan los nudos donde se producen las excitaciones y el tipo de éstas.

El programa contiene una herramienta que realiza la traducción a ficheros SPICE del *netlist* simulado que permite, si el usuario lo desea, volver a simular en SPICE el mismo circuito. Se dispone, de igual forma, de una utilidad que realiza el proceso inverso, es decir, a partir del contenido de un fichero SPICE se obtienen ficheros en el formato establecido para la aplicación, lo que posibilita la simulación del fichero SPICE empleando el modelo temporal desarrollado. Estas herramientas se describen en los apartados 3 y 4 del presente capítulo.

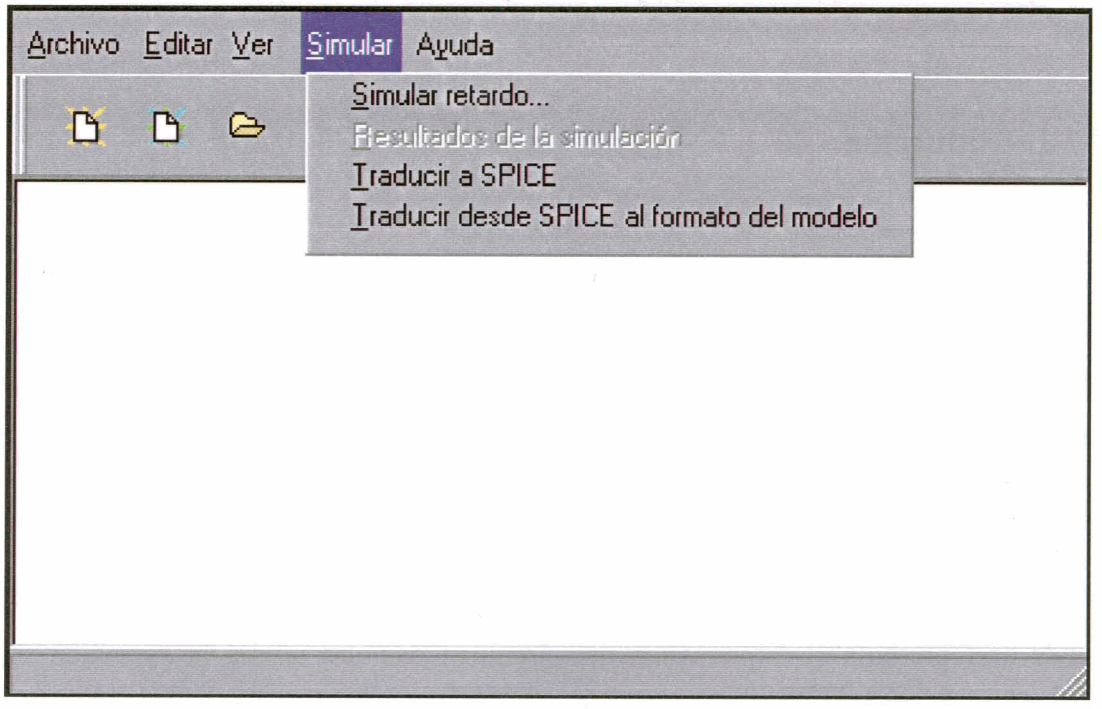


Imagen 3. 1. Ventana principal de la aplicación.

3.2. Cálculo de tiempos de propagación

La simulación se realiza sobre el fichero *netlist* (*.net) abierto en la ventana principal de la aplicación. A este archivo *netlist*, se le aplican las excitaciones definidas en el fichero de entradas (*.ent) que se especifique. Asimismo, se hace uso de un fichero, seleccionable por el usuario, que recoge los valores de los parámetros de modelado a utilizar en el cómputo de las ecuaciones. Dicho archivo se llama fichero de la tecnología (*.dat). Estos ficheros son generados por el usuario desde el editor de textos de la aplicación. Inicialmente, el programa actúa aplicando las ecuaciones del modelo temporal, correspondientes a los tiempos de retardo t_{pHL} y t_{pLH}, descritas en el capítulo 2, pero se permite al usuario introducir desde teclado nuevas ecuaciones. Esta capacidad de la aplicación la convierte en una herramienta para el desarrollo de modelos temporales basados en ecuaciones, permitiendo introducir nuevos parámetros no modelados actualmente, como puede ser la temperatura. La imagen 3.2 es el cuadro de diálogo que el

programa ofrece al usuario para cambiar el modelo y seleccionar los ficheros de entradas y de la tecnología.

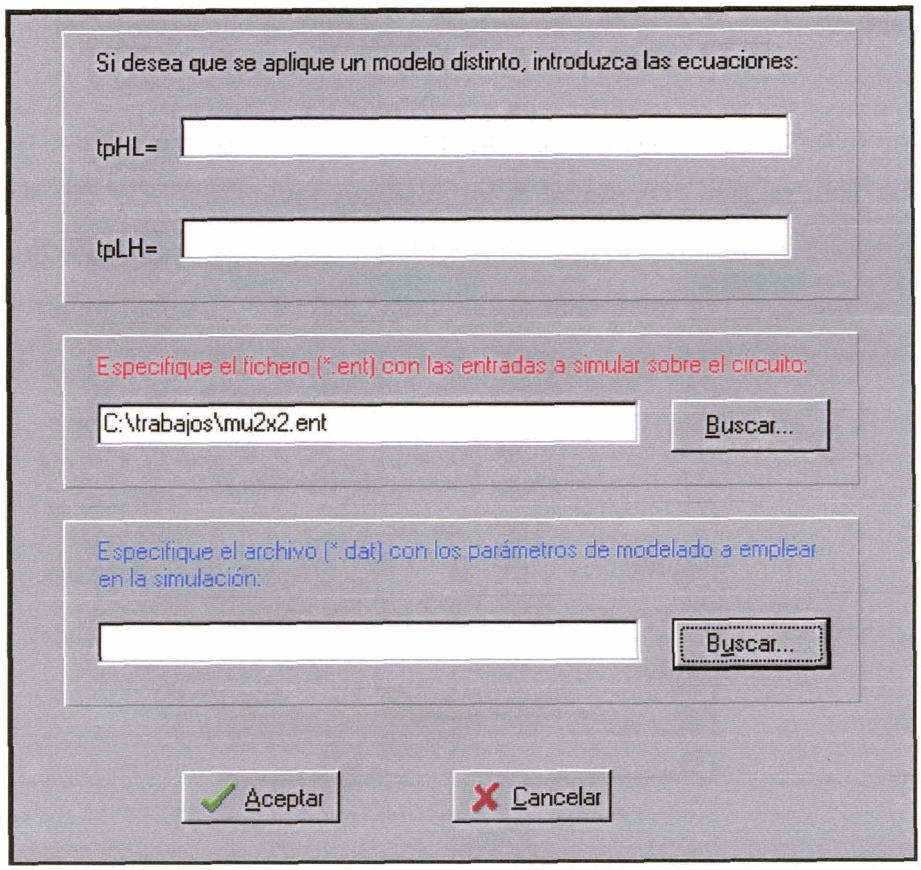


Imagen 3. 2. Cuadro de diálogo para la introducción de ecuaciones de modelado y seleccionar los ficheros de entradas y de la tecnología.

3.2.1. Ecuaciones temporales

Las ecuaciones se emplean para modelar los tiempos de las transiciones baja-alta, t_{pLH}, y alta-baja, t_{pHL}. Estas ecuaciones son, en principio, las del modelo presentado en el capítulo 2, aunque como se ha dicho se da al usuario la posibilidad de cambiar el modelo introduciendo nuevas ecuaciones.

Es posible, de igual forma, introducir una única ecuación, bien para t_{pHL}, bien para t_{pLH}. En este caso para el cálculo de los tiempos de propagación el programa usará dicha ecuación conjuntamente con la del modelo correspondiente a la transición no introducida desde teclado.

Los valores de los parámetros de modelado que forman parte de estas ecuaciones externas deben definirse también en el fichero de la tecnología. Además, las ecuaciones han de ser únicas para cada tipo de transición, es decir, válidas para todos los casos a diferencia de lo que ocurre con el fan-out en las ecuaciones de modelado.

Se ha establecido para las expresiones aritméticas introducidas desde teclado qué caracteres son válidos para representar a los distintos operadores. De esta forma para agrupar sumandos o expresiones se utilizan paréntesis, no aceptándose el uso de corchetes ('[',']'). La multiplicación se representa mediante el carácter '·' y como punto decimal se emplea el carácter '.'. En la siguiente tabla se muestran los operadores permitidos.

Tabla 3. I. Operadores matemáticos permitidos.

paréntesis	'(' ')'
suma	'+'
resta	'-'
multiplicación	'.'
división	'/'

Además se ha fijado el orden de precedencia de evaluación entre los distintos tipos de operadores, el cual se glosa a continuación:

1. Primero se evalúan todas las subexpresiones entre paréntesis. Las subexpresiones con paréntesis anidados se evalúan de dentro-afuera, es decir, el paréntesis más interno se computa primero.

2. Dentro de una misma expresión o subexpresión, los operadores se evalúan en el siguiente orden:

   ·, / primero

   +, - último

3. Los operadores en una misma expresión o subexpresión con igual nivel de prioridad, son calculados de izquierda a derecha.

El cálculo de las ecuaciones introducidas desde teclado se realiza mediante el análisis textual de las mismas y consiste en la búsqueda de los diversos operadores que contiene la ecuación y la evaluación de las subexpresiones ligadas a dichos operadores. La repetición de estas operaciones para cada puerta del circuito, al aplicar la correspondiente ecuación temporal, t_{pHL} ó t_{pLH}, daría lugar a un incremento en el tiempo de ejecución del programa. Se reduce el tiempo de ejecución efectuando inicialmente una simplificación de la expresión.

En las expresiones del modelo utilizado, tan sólo dos parámetros, la relación de aspecto, β, y el fan-out, FO, son informaciones contenidas en el *netlist*, y por lo tanto, propias de cada puerta presente en el mismo. El resto de los parámetros empleados en la ecuación son aportados por el fichero de la tecnología, información constante en las repetidas evaluaciones de la ecuación. Esto permite expresar dichas ecuaciones como:

$$t_{pHL,LH} = K \cdot f(\beta, FO),$$

donde K es una constante que se evalúa en la simplificación previa.

Debido al sistema usado para el análisis de las expresiones, se facilita la simplificación situando los términos en β y FO al final de la expresión. Así :

T=(exp1)·(exp2)·((exp3)±a·(b±β)±c·(d±FO)), da lugar a
T=K1·(K2±a·(b±β)±c·(d±FO))

3.2.2. Tipos de ficheros

Como ya se ha mencionado, el programa hace uso de tres ficheros que contienen la información necesaria para simular un circuito. Se trata de un fichero *netlist*, que contiene la descripción del circuito a simular, un fichero de entradas y el fichero de la tecnología con los parámetros de modelado a emplear en la simulación.

Una característica común a los ficheros *netlist* y de entradas es la posibilidad de incluir comentarios anteponiendo a la línea de texto el carácter '*'.

A continuación se describe cada uno de ellos.

3.2.2.1. Netlist

El fichero *netlist* (*.net) consiste en un fichero de texto, que contiene la información relativa al circuito que se desea simular. Esta información corresponde a las puertas que conforman el circuito, los nodos, identificadores numéricos, a que se conecta cada una, y el valor de la relación de aspecto, β, correspondiente a cada puerta.

La aplicación soporta dos tipos de puerta, inversores y puertas NOR de dos entradas, nor2. No se ha incluido el tipo nor3, NOR de tres entradas, al no estar desarrollado el modelado de la misma.

Cada línea de *netlist* contiene la información propia de una puerta, así se tiene:

INV Nodo_de_entrada Nodo_de_salida β
NOR2 Nodo_de_entrada_1 Nodo_de_entrada_2 Nodo_de_salida β

donde INV y NOR2 representan, respectivamente, a la puerta lógica inversora y la puerta NOR de dos entradas.

Se ha impuesto la restricción de no poder usar el 0 como identificador de un nodo. Esto se debe a su uso en el proceso de evaluación de los tiempos de propagación, para identificar el tipo de las puertas del circuito.

En la Figura 3.1 se presenta como ejemplo un circuito multiplexor 2-1 implementado con puertas básicas inversoras y NOR2; el circuito tiene dos entradas de datos y una de selección. El *netlist* correspondiente a dicho circuito se muestra en el cuadro 3.1.

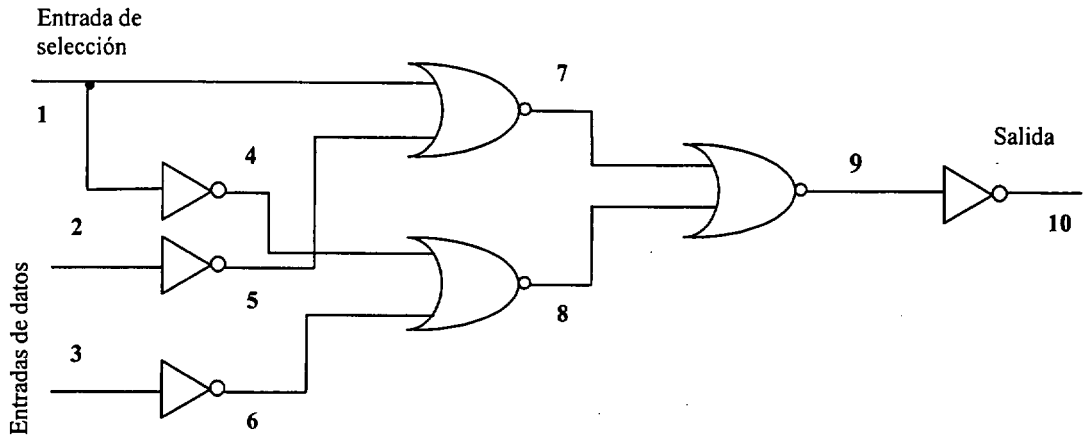


Figura 3. 1. Multiplexor 2-1.

```
*comentarios
* puerta nodos β
inv 1 4 2
inv 2 5 2
inv 3 6 2
nor2 1 5 7 2
nor2 4 6 8 2
nor2 7 8 9 2
inv 9 10 2
```

Cuadro 3. 1. *Netlist* del multiplexor 2-1.

Para almacenar los datos del *netlist* se utilizan estructuras dinámicas de datos, concretamente una lista enlazada. Así, no se establecen límites al tamaño de los circuitos a simular.

3.2.2.2. Entradas

El fichero de entradas (*.ent) recoge las entradas que se aplican al circuito cuya respuesta temporal se desea simular.

El modelo evalúa la respuesta temporal a las pendientes de las señales de entrada, cosiderándose el sentido de las mismas, subida o bajada, y no los tiempos de subida y bajada que no son considerados como parámetros sensibles por el modelo. Una pendiente ascendente se designa por el valor entero 1, y a una descendente se le asigna el valor –1.

Se especifica el nodo que recibe la señal de entrada, así como el posible retardo relativo entre las distintas entradas que puede poseer el circuito. De esta forma se permite al usuario comprobar el efecto de aplicar entradas con diferencias temporales entre sí.

Una entrada que permanezca en el nivel lógico alto se especifica mediante la letra 'h'. Análogamente se señala con la letra 'l' una entrada de nivel lógico bajo. Se pueden emplear letras mayúsculas indistintamente para ambos tipos de entrada.

El programa reconoce el final del fichero cuando encuentra la palabra clave 'end'. El Cuadro 3.2 muestra un ejemplo de fichero de entradas de aplicación sobre el *netlist* del apartado anterior:

```
* la entrada 1 (selección)
* se encuentra a nivel bajo, 2
* recibe una transición bajo-alto y
* 3 presenta un nivel alto
* nodo pendiente retardo
1 L
2 1 0
3 h
end
```

Cuadro 3. 2. Fichero de entradas.

En las entradas con las que trabaja la aplicación se puede definir el sentido de la transición y el instante en que se produce la misma, pero sólo se evaluará una única transición sobre cada entrada.

3.2.2.3. Tecnología

El fichero de la tecnología (*.dat) es, al igual que los dos anteriores, un fichero de tipo texto. Contiene los valores de los parámetros de modelado a utilizar en el cómputo de las ecuaciones ya sean las del modelo o las de usuario. Corresponden a la tensión de alimentación, V_{DD}, y a los parámetros tecnológicos de los transistores HFET empleados. El formato del fichero es muy sencillo, cada línea incluye el identificador del parámetro, seguido de su valor. En el Cuadro 3.3 se presenta un ejemplo del mismo.

```
vdd 1.5
tphl0 6.12
tplh0 25.41
ne 1.5
rse 6
ke 2.5
vte 0.5495
ce 2.95
rsd 5
dd 1.9
vtd -0.1
kd 1.7
vsbd 0.78
```

Cuadro 3. 3. Fichero de la tecnología.

3.2.2.4 Editar ficheros

Si el usuario desea editar los ficheros utilizados en la simulación dispone de comandos de menú y botones en la barra de herramientas que realizan las funciones estándar de este tipo de utilidades como cortar, copiar, borrar y pegar.

3.2.3. Resultados

Una vez realizada la simulación de un *netlist* determinado, donde se hace uso de los ficheros *netlist*, de entradas y de la tecnología, los resultados se pueden consultar de dos formas: por nodo, que corresponden a los identificadores de las salidas de cada puerta del circuito, o por nodo crítico. En la consulta por nodo el usuario selecciona el nodo correspondiente a aquella puerta del circuito cuya respuesta temporal desea conocer. El nodo escogido puede encontrarse a nivel lógico alto o bajo, o bien presentar una transición, en cuyo caso se informa del tiempo de propagación registrado en el nodo. Nodo crítico es aquel nodo del circuito correspondiente a la puerta que presenta el mayor retardo. Se ha establecido una opción que permite al usuario acceder de forma directa a esta información. A la secuencia de nodos del circuito que conectan la entrada del circuito y el nodo cuya respuesta temporal solicita el usuario se le denomina camino. Esta información se muestra al ususario conjuntamente con el tiempo de propagación. En el caso del nodo crítico se le llama camino crítico.

Se realiza una representación gráfica de los tiempos de propagación de los nodos presentes en el camino relativo al nodo seleccionado por el usuario. El usuario puede conocer el retardo en un nodo de la serie colocando el cursor del ratón sobre el punto que lo representa. Si la serie contiene un número elevado de puntos la visualización de éstos puede mejorarse aplicando un zoom sobre la serie. Para ello se debe situar el cursor del ratón en el borde superior del área de la representación y arrastrarlo hacia la parte inferior derecha englobando los puntos que desea ver. Para devolver la vista a su estado original se sitúa el ratón en el borde inferior derecho de la representación y se lleva hacia la parte superior izquierda.

El usuario puede seleccionar la referencia de tiempo respecto a la que se presentan los resultados de la simulación. El origen de tiempos se sitúa en 0, o bien en el nodo inicial del camino en función de la elección que realice. Con esta opción si la entrada no cambia hasta un tiempo inicial determinado, éste no aparece en el tiempo de propagación de la señal. La imagen 3.3 de SiDSen muestra la ventana de la aplicación designada para que el usuario consulte los resultados.

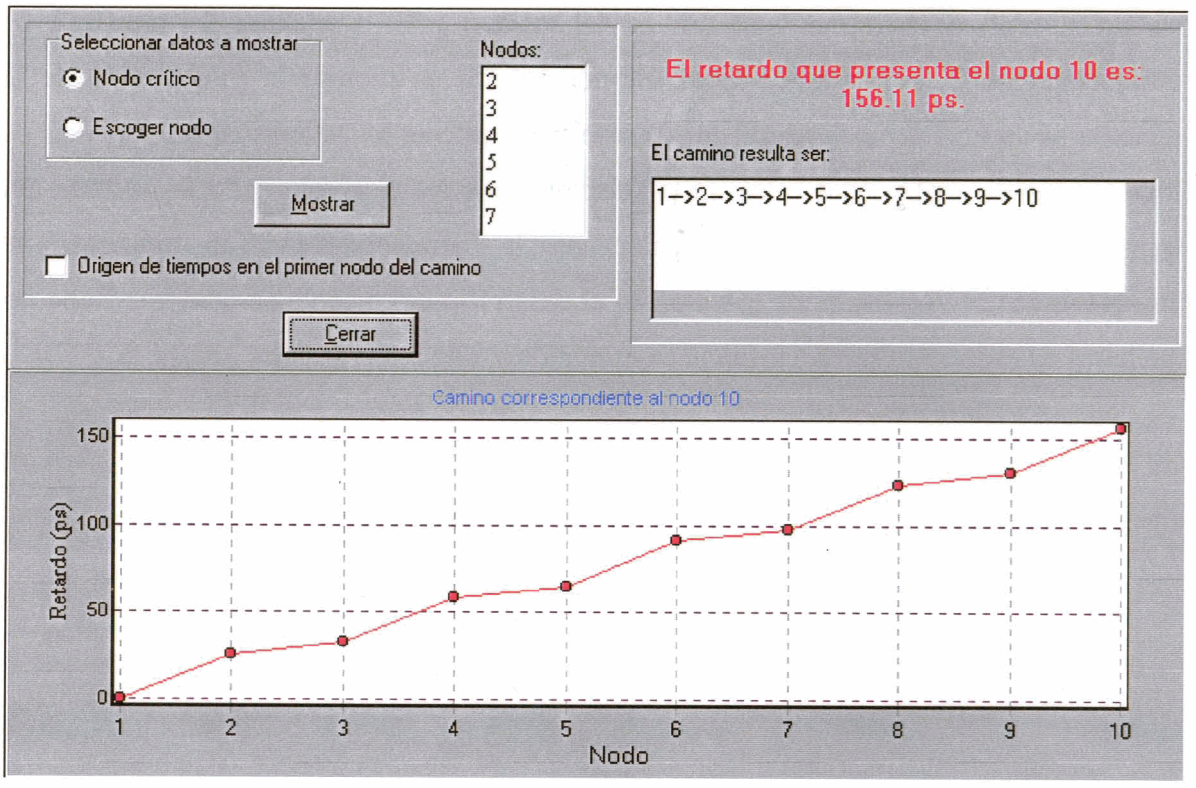


Imagen 3. 3. Ventana para la consulta de resultados.

3.3. Traducción a SPICE

A partir del *netlist* activo se genera un fichero SPICE, al que se asigna la extensión SPI. La versión de SPICE utilizada, IAFSPICE, acepta la extensión SPI de forma válida. Esta utilidad permite comparar los resultados obtenidos en la simulación realizada a través de la aplicación de las ecuaciones del modelo con los obtenidos mediante SPICE. Antes de realizar la simulación debe convertirse el fichero creado empleando el comando **dos2unix**.

La especificación de las puertas del circuito en el fichero SPICE se realiza mediante la definición previa de cada tipo de puerta en la declaración de subcircuitos, SUBCKT, para a continuación proceder al listado de las mismas.

Se dispone de dos tipos básicos, inversor y NOR de dos entradas, que derivan hacia diversos subtipos, cuya diferencia reside en el valor de la relación de aspecto, β. En la declaración de cada subcircuito se especifican las conexiones de cada uno de los transistores *HFET*, *Heteroestructure Field Effect Transistor*, así como sus respectivas

anchuras, W_D (anchura del transistor de deplexión) y W_E (anchura del transistor de enriquecimiento). A través de estos últimos valores se incluye la información de la relación de aspecto de cada puerta del circuito. Para obtener los valores de las anchuras de los transistores a partir del valor de la relación de aspecto se ha fijado la anchura del transistor de deplexión, W_D, obteniéndose la correspondiente al transistor de enriquecimiento, W_E, mediante cálculo. En el desarrollo del modelo se comprobó que las anchuras de cada transistor no influían en el retardo y sí la relación entre ambas, β.

$$\beta = \frac{W_E}{W_D}$$

donde $W_D = 5$

y se obtiene $W_E = 5 \cdot \beta$

El fichero SPI incluye la información de los parámetros que modelan a cada uno de los transistores. El valor de los parámetros de modelado se obtiene del fichero de la tecnología (*.dat) que el usuario de la aplicación determine. Los valores correspondientes a los restantes parámetros que modelan a los transistores se han fijado.a su valor nominal debido a que el modelo temporal basado en el análisis de la sensibilidad los clasificó como insensibles.

La equivalencia entre los parámetros usados por el modelo y los asignados a los modelos de los transistores de SPICE permite realizar estudios comparativos entre ambas herramientas de simulación.

La tensión de alimentación, V_{DD}, se obtiene también del fichero de la tecnología, pues se trata de un parámetro de gran importancia, aunque no sea un parámetro propio de la tecnología HFET utilizada. El valor de V_{DD} se especifica en la declaración de subcircuito.

A continuación se presenta en la tabla 3.II el listado de los nombres de los parámetros SPICE y su traducción al modelo utilizado.

Tabla 3. II. Parámetros de modelado.

	SPICE	Modelo	
Transistor de enriquecimiento	VC	vte	Tensión umbral
	RS	rse	Resistencia de fuente
	CDVC	ke	Transconductancia
	NP	ne	Factor de idealidad del diodo
	BETA	ce	Factor complementario a la tensión umbral
Transistor de deplexión	VC	vtd	Tensión umbral
	RS	rsd	Resistencia de fuente
	CDVC	kd	Transconductancia
	VSB	vsbd	Tensión fuente sustrato
	DELTA	dd	Factor de la tensión VSB

El contenido del fichero SPICE se completa con las entradas a simular sobre el circuito, y la información del análisis que se realizará.

Las entradas se construyen a partir del fichero de entradas (*.ent) que se haya especificado. Se genera un escalón, creando una entrada del tipo Pwl, a partir de cada entrada que presente transición, dado que como alternativa sólo se tiene la generación de una cadena de pulsos. Como elemento de comparación con el modelo de ecuaciones interesa la respuesta temporal al escalón de la señal de entrada SPICE. Las entradas especificadas como de nivel fijo se declaran en el fichero SPICE empleando la palabra clave 'DC'.

La información del tipo de análisis, que incluye la especificación del nodo cuya respuesta se desea conocer, y los tiempos de análisis transitorio y de paso, la edita el usuario a través de un cuadro de diálogo. El tiempo que establezca para la duración del análisis transitorio se utiliza como última referencia de tiempo en la definición de cada entrada de tipo Pwl. Si el nodo que escoge es una salida del circuito se debe conectar una carga al mismo para poder realizar la simulación SPICE. Automáticamente se inserta una cadena de tres inversores, que presentan igual relación de aspecto que la puerta de salida del circuito.

La imagen 3.4 muestra el cuadro de diálogo que el programa presenta al usuario para seleccionar los ficheros de entradas y de la tecnología, y definir la información de análisis.

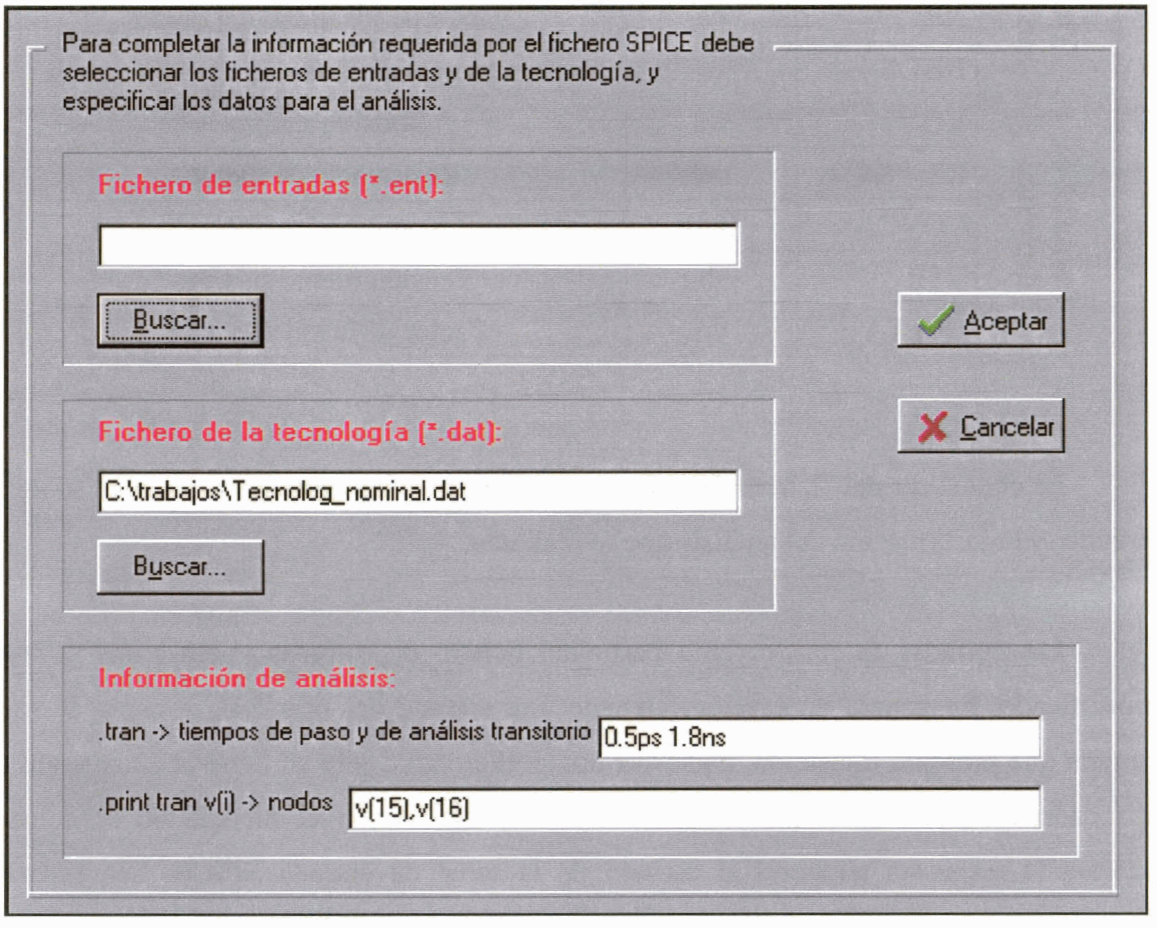


Imagen 3. 4. Cuadro de diálogo para seleccionar los ficheros de entradas y de la tecnología, y especificar los datos necesarios para el análisis.

3.4. Traducción desde SPICE

Esta herramienta permite al usuario simular un fichero SPICE empleando el modelo desarrollado. Para ello la aplicación genera archivos *netlist*, de entradas y de la tecnología a partir del fichero SPICE abierto en el editor de la ficha principal. Para poder convertir el archivo generado mediante IAFSPICE es necesario aplicar previamente el comando **unix2dos**.

La información del fichero SPICE se estructura en diversas secciones. Se declaran los modelos de los transistores empleados en la simulación. El valor de los parámetros de modelado considerados como sensibles se incorpora al fichero de la tecnología que se crea. La Tabla 3.II contiene los parámetros de modelado SPICE y sus equivalentes en el fichero de la tecnología.

La declaración de subcircuitos establece la definición de los distintos tipos de puerta presentes en el circuito a simular. Se especifica un subcircuito para cada puerta básica, inversora o NOR2, con valor diferente de la relación de aspecto, β. El valor de β se calcula a través de los valores de las anchuras de los transistores que conforman la puerta que se define mediante el subcircuito. El valor de la relación de aspecto correspondiente al tipo de subcircuito declarado se obtiene dividiendo el valor especificado para la anchura del transistor de enriquecimiento, W_E, por el valor establecido para la anchura del transistor de deplexión, W_D.

El modelo utilizado por la aplicación emplea un único valor de tensión de alimentación en la evaluación de los tiempos de retardo, con lo que ha de utilizarse el mismo valor en todos los subcircuitos declarados en el fichero SPICE, o especificar un único valor en el fichero SPICE. Dicho valor se añade al fichero de la tecnología como el parámetro vdd.

Con el circuito declarado en la correspondiente sección y los subcircuitos definidos en el fichero SPICE se genera el fichero *netlist* equivalente asignando a cada puerta declarada el valor de β que le corresponde según el tipo de subcircuito al que pertenece.

El fichero de entradas se crea con el contenido de la sección de entradas del archivo SPICE. En las entradas de tipo Pwl el sentido de la transición que presenta la señal se utiliza para determinar la transición, 1 o –1, de la entrada equivalente en el formato del modelo. La segunda referencia de tiempo especificada en la entrada SPICE se asigna a la entrada que se añade al fichero de entradas como retardo parcial.

4. Descripción del código

4.1. Introducción

En el presente capítulo se describe detalladamente el código utilizado para realizar el programa. Al haber sido diseñado de forma abierta para que se puedan incorporar nuevos parámetros de modelado, o incluso nuevos modelos, este capítulo pretende ser una guía del código que permita su posterior modificación.

La aplicación se ha desarrollado bajo el entorno de programación Delphi3 [7], [8], [9]. El conocimiento previo del lenguaje Pascal [10] ha sido uno de los fundamentos para hacer uso de dicho entorno. Se trata de un entorno de fácil utilización, muy visual, que permite la construcción de aplicaciones mediante el uso de sus componentes predefinidos. Delphi3 provee al programador de controles y componentes típicos de la interfaz de usuario de Windows95. Esto permite la compatibilidad de la aplicación con ésta y posteriores versiones de Windows. Los componentes se obtienen de la Paleta de componentes de Delphi. El programador caracteriza los controles modificando sus propiedades en tiempo de diseño. El código se asocia a los eventos generados por el usuario de la aplicación sobre los elementos de la interfaz de usuario. Delphi3 facilita la labor del programador generando y manteniendo parte del código de forma automática.

4.2. Explicación del código

El programa, como corresponde a una aplicación desarrollada bajo Delphi3, mantiene el código estructurado en varias unidades. Delphi genera de forma automática una unidad para cada ficha incluida en la aplicación. Estas fichas, desde el punto de vista del usuario, corresponden a ventanas de la aplicación. La unidad sim4 se ha creado para contener el código necesario para realizar la simulación, y no se asocia a ninguna ficha. Éste es el código fundamental de la aplicación y se agrupa en una unidad independiente de la asociada a la ficha principal de la aplicación.

La aplicación cuenta con seis unidades y cinco fichas o *forms*, que se presentan en la Tabla 4.I.

Tabla 4. I. Unidades y fichas de la aplicación.

Unidad	Ficha
cuadro_t	TAS
cuadro_sim	DatSim
princ5	FormPFC
res4	Resultado
sec4	SelSalv
sim4	---------

La ficha **FormPFC** corresponde a la ventana principal de la aplicación. Las fichas **TAS, DatSim** y **SelSalv** se han usado para la implementación de diversos cuadros de diálogo. Se trata de fichas que se caracterizan como cuadros de diálogo especificando el tipo de borde que presentan y la interfaz de usuario. La interfaz de la ficha carece de icono de sistema, cuadros de maximizar o minimizar, y tiene el típico borde fino de un cuadro de diálogo estándar de Windows, que no permite modificar su tamaño. Dicha caracterización se realiza fijando el valor de la propiedad *BorderStyle* de la ficha. Las fichas contienen componentes de edición para recoger datos de usuario, así como botones a los que se ha asignado valor de retorno, *ModalResult*. De esta forma al pulsar el usuario un botón se devuelve el valor asignado y se cierra la ficha de forma automática. La ficha se muestra en forma modal. Esto quiere decir que toma el foco permaneciendo activa, y el usuario debe cerrarla para poder volver a la ventana principal de la aplicación que hasta ese momento se encuentra inhabilitada.

La ficha **Resultado** tiene carácter secundario, y se utiliza como interfaz para el acceso por parte del usuario a los resultados de la simulación. Su llamada se efectúa, también, en forma modal.

La unidad **sim4** contiene la declaración del tipo de datos usado en la aplicación para contener los datos del circuito, así como el código que implementa la simulación.

Todas las fichas se crean de forma automática por el archivo fuente del proyecto **SiDSen.dpr**, que se describe en el apartado 4.2.2.

4.2.1. Unidades del programa

Se presentan las unidades que constituyen el programa, describiendo los controles incorporados a las fichas y el código implementado.

4.2.1.1. Unidad cuadro_t

Esta unidad contiene una ficha de cuadro de diálogo que se muestra en forma modal. Contiene elementos de edición, de tipo *Edit*, para obtener del usuario la información de análisis necesaria para el fichero SPICE que se está creando. También integran el cuadro de diálogo dos botones, Button1 y Button2, de tipo *TButton*, cada uno de ellos asociado a un componente de edición, Edit1 y Edit2, para seleccionar los ficheros de entradas y de la tecnología precisos para completar los datos del fichero SPICE. Cuando el usuario pulsa uno de los botones SiDSen muestra un cuadro de diálogo estándar de selección de archivos. A través de estos cuadros se puede navegar por el directorio y seleccionar el fichero adecuado.

Dos botones, de tipo *BitBtn*, a los que se fijan valores de retorno modal, Aceptar y Cancelar, se sitúan en la parte inferior del cuadro de diálogo. Si se pulsa el botón Aceptar se da validez a los datos introducidos que son incluidos en el fichero SPICE. Si se pulsa Cancelar se aborta el proceso de traducción a SPICE.

Seguidamente se describe el código asociado a los controles de la unidad:

4.2.1.1.1 Método Button1Click.

El método Button1Click gestiona el evento OnClick sobre el botón Button1. Se emplea para seleccionar el fichero con las entradas a insertar en el archivo SPICE en proceso de creación. El método muestra el cuadro de diálogo estándar de selección de ficheros dialogante, de tipo *OpenDialog*. Se ejecuta el método *Execute* del mismo y su propiedad *FileName* recoge el nombre del archivo que el usuario selecciona, el cual se asigna al componente de edición Edit1.

4.2.1.1.2. Método Button2Click.

El método Button2Click maneja el evento OnClick sobre el botón Button2. Este botón se utiliza para escoger el fichero de la tecnología con los parámetros a incluir en el fichero SPICE. El código generado realiza una llamada al cuadro de diálogo predefinido dialogante, componente de tipo *TOpenDialog*. El método *Execute* de la clase *TOpenDialog* devuelve el valor lógico falso si el usuario cancela la operación o verdadero si pulsa el botón Abrir que presenta el cuadro de diálogo. La propiedad *Filename* del cuadro de diálogo registra, en este último caso, el nombre del fichero seleccionado que se asigna al campo *Text* del componente de edición Edit1.

4.2.1.2. Unidad cuadro_sim

La unidad se asocia a una ficha con formato de cuadro de diálogo. Se ejecuta en modo modal, y su llamada se produce en el proceso de simulación.

Se emplea como interfaz de usuario para seleccionar los ficheros de entradas y de la tecnología a aplicar en la simulación, así como para introducir ecuaciones de modelado desde teclado. Para seleccionar los ficheros dispone de dos componentes de edición Edit1 y Edit2, tipo *TEdit*, y dos botones, Button1 y Button2 de tipo *Button*. Otros dos componentes de edición thledit y tlhedit se han dispuesto para especificar las expresiones externas de modelado si el usuario lo desea.

Se le han añadido dos botones modales, Aceptar y Cancelar, componentes de tipo *BitBtn,* para confirmar la realización de la simulación. Si el usuario pulsa el botón Aceptar se lanza la simulación empleando los ficheros seleccionados y las posibles ecuaciones introducidas. Pulsando el botón Cancelar impide que se ejecute la simulación.

Se describe seguidamente el código implementado para los elementos de la ficha:

4.2.1.2.1. Método Button1Click.

El método gestiona el evento OnClick del botón Button1. Se emplea para seleccionar el fichero con las entradas a simular sobre el fichero *netlist.* Para ello se muestra el cuadro de diálogo predefinido de apertura de ficheros *Dialogo* a través de su método *Execute.* La propiedad *FileName* del cuadro contiene el nombre del fichero, el cual se asigna al campo *Text* del componente Edit1 de la ficha.

4.2.1.2.2. Método Button2Click.

Método que maneja el evento OnClick del botón Button2. Este botón se utiliza para escoger el fichero de la tecnología que contiene los parámetros de modelado a emplear en la simulación. Se realiza una llamada al cuadro de diálogo predefinido *Dialogo,* componente de tipo *TOpenDialog* mediante el método *Execute* de la clase *TOpenDialog.* El nombre del fichero que seleccione el usuario se recoge en la propiedad *FileName* del cuadro, y se copia al campo *Text* del elemento de edición Edit2.

4.2.1.3. Unidad sec4.

La unidad sec4 contiene una ficha caracterizada como cuadro de diálogo. Se la llama en modo modal. Su uso se asocia a diversas funciones del menú de tratamiento de archivos como Nuevo, Abrir, etc. Si el archivo actual ha cambiado se pregunta al usuario si desea guardar los cambios, botón Si, descartarlos, No, o saltar la operación en curso, Cancelar. A dichos botones, elementos de tipo *BitBtn,* se les asignan valores de retorno, lo

que produce que al pulsarlos se devuelva dicho valor y se cierre el cuadro de diálogo automáticamente.

4.2.1.4. Unidad sim4.

El código de esta unidad no está asociado a ninguna ficha. En la unidad se encuentra el código implementado para la obtención de los datos contenidos en los ficheros *netlist*, de entradas y de la tecnología, así como el código desarrollado para evaluar los tiempos de propagación.

Esta sección comienza con la presentación del tipo de datos definido para soportar la información del circuito a simular, para a continuación describir el código contenido en la unidad.

Al tratarse de una aplicación cuya finalidad es la simulación de tiempos de propagación y con la intención de no limitar el número de puertas que puede soportar, se ha optado por el uso de tipos de datos dinámicos, concretamente punteros.

Se ha creado una estructura en forma de lista doblemente enlazada. Cada elemento de la estructura es de tipo registro, *record*, constando de varios campos de información y de dos punteros a este tipo, que enlazan a cada elemento con el anterior y el posterior en la estructura de la lista. La elección de una lista doblemente enlazada se debe a la mayor flexibilidad que ofrece a la hora de recorrerla respecto a la estructura de enlace simple, permitiendo el movimiento en ambos sentidos.

Cada registro o elemento de la lista contiene toda la información relativa a una puerta del circuito, esto es:

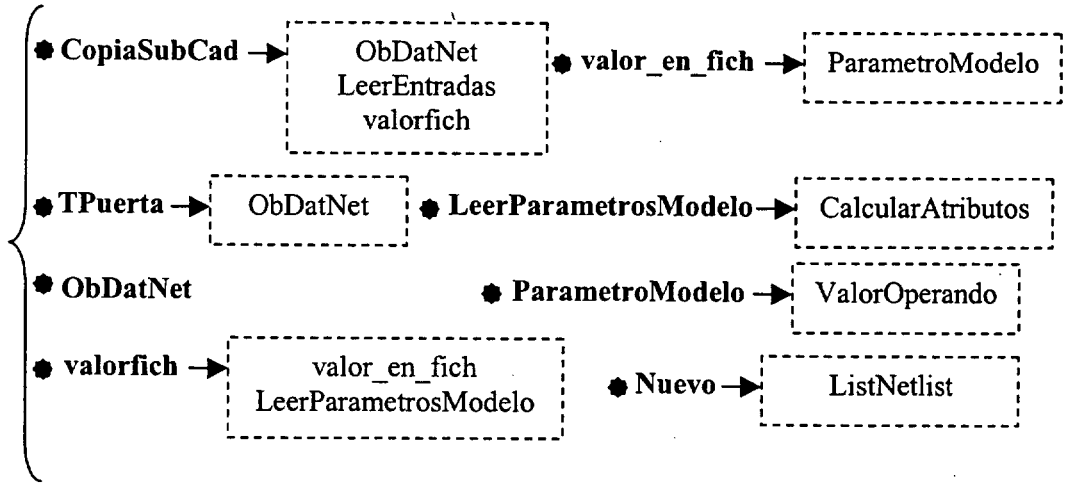
_ nodo (o nodos) de entrada de la puerta: campos nodoe1 y nodoe2, de tipo entero, *integer* ;
_ nodo de salida de la puerta: campo nodo_s, de tipo *integer*;
_ fan-out: campo fanout, tipo real, debido al concepto de fan-out aplicado;
_ nodo de entrada a cuyo cambio responde la puerta: campo respcambio, tipo *integer*;
_ valor de la relación de aspecto, β: campo beta, tipo real;

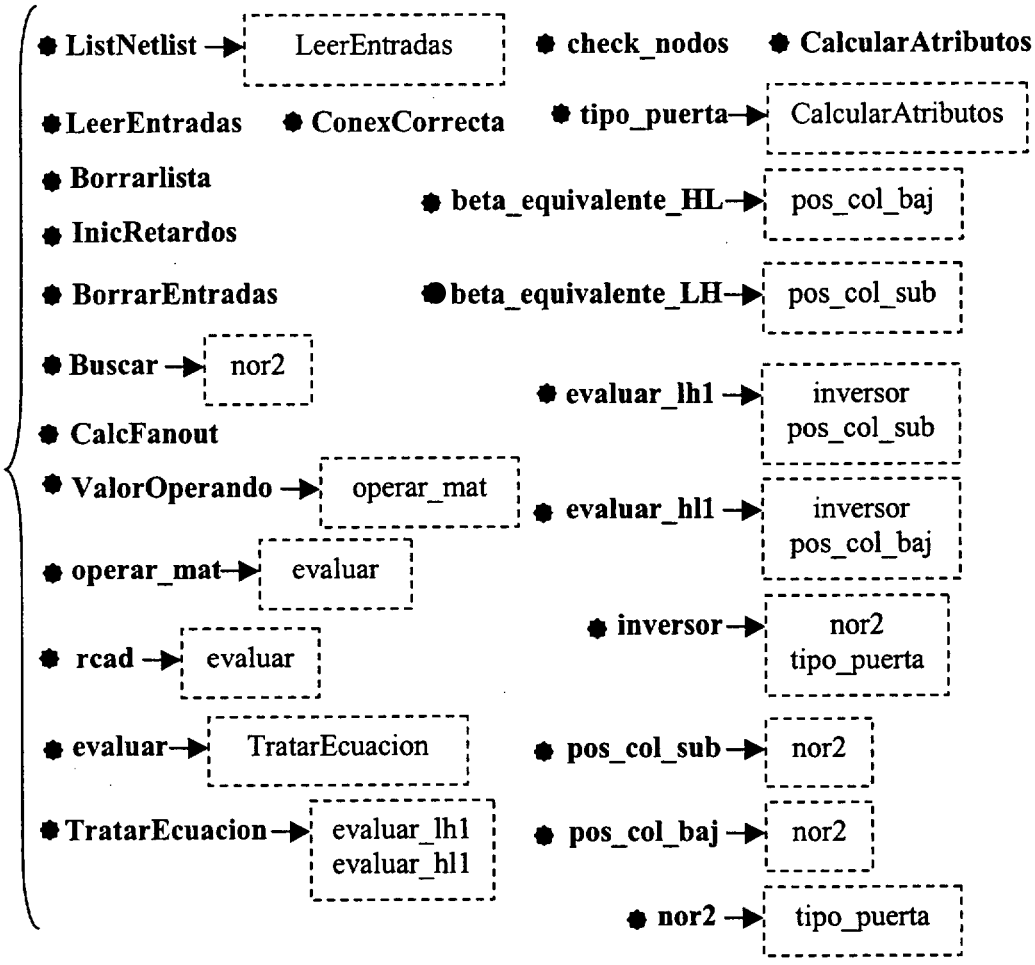
_ pendiente de la salida de la puerta: campo pendiente, tipo *integer*, pues no se modela el tiempo de subida o bajada, sino el sentido de la transición;
_ tiempo de propagación alto-bajo ó bajo-alto: campo tp, tipo real.

Seguidamente se presenta la declaración de este tipo de datos:

```
PunteroPuerta = ^Puerta;
Puerta = record
        nodoe1,nodoe2,nodo_s,fanout,respcambio: integer;
        beta,pendiente,tp: real;
        ant,prox : PunteroPuerta;
end;
```

El orden en que se explican las funciones y procedimientos obedece a que los primeros que se describen se utilizan por parte de los subprogramas presentados posteriormente. Con objeto de facilitar el seguimiento de la descripción de los procedimientos y funciones se presenta en forma de esquema una relación de los subprogramas de la unidad. Se muestran en negrita, y se incluyen en unos cuadros adyacentes los subprogramas de sim4 que los utilizan.





A continuación se presentan los procedimientos y funciones incluidos en la unidad, describiendo su funcionalidad y presentando sus organigramas.

4.2.1.4.1. Función CopiaSubCad.

La función se usa para extraer cada uno de los datos que contiene una línea de texto correspondiente a los ficheros con que trabaja la aplicación (*.net, *.dat y *.ent).

Extrae una subcadena formada por los caracteres existentes anteriores al primer carácter espacio encontrado, y devuelve la misma. Borra dichos caracteres, así como los posibles caracteres espacio que pasasen a ocupar el inicio de la nueva cadena resultante. Por ejemplo, si se tiene la siguiente cadena:

N	O	R	2			1	

La función CopiaSubCad extrae la cadena 'NOR2' que devuelve como resultado de la función y borra los caracteres obtenidos y los dos espacios que ocuparían el comienzo, con lo que la cadena fuente queda:

1	...

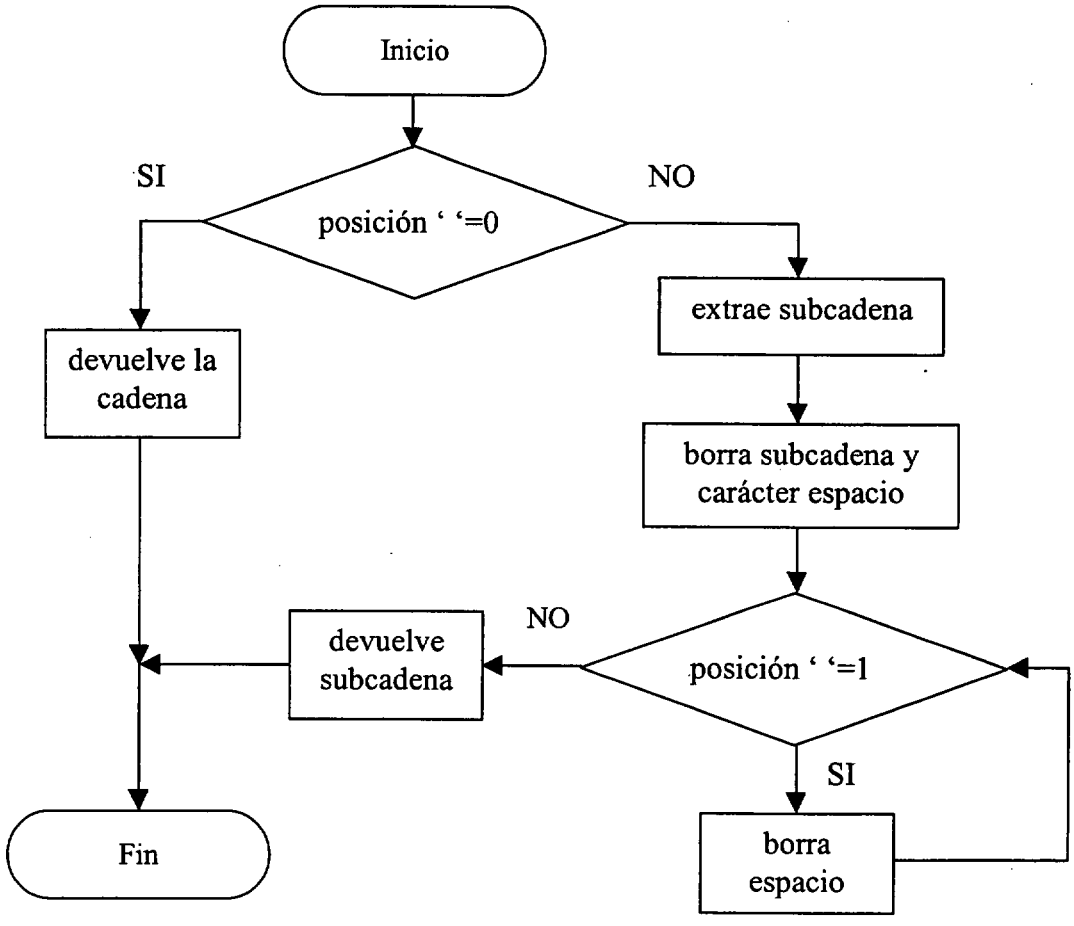


Figura 4. 1. Diagrama de flujos de la función CopiaSubCad.

4.2.1.4.2. Función TPuerta.

Comprueba que la puerta especificada en el fichero *netlist* es soportada por la aplicación: inversor o NOR de dos entradas. Devuelve un valor entero que identifica el tipo de puerta.

Tabla 4. II. Tipos de puerta.

Valor devuelto	Tipo de puerta
0	Puerta no soportada
1	Inversor
2	NOR 2

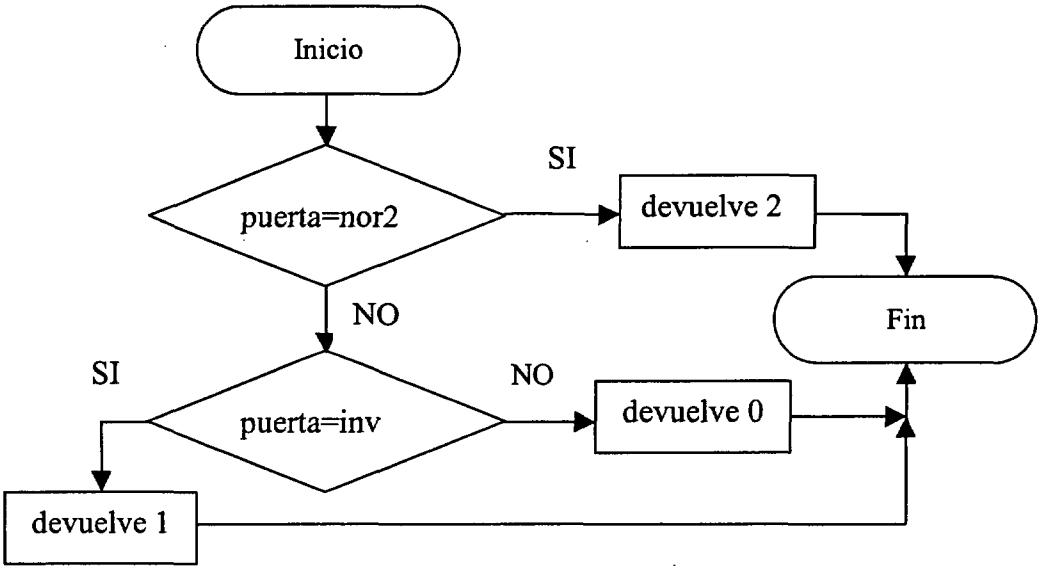


Figura 4. 2. Diagrama de flujos de la función TPuerta.

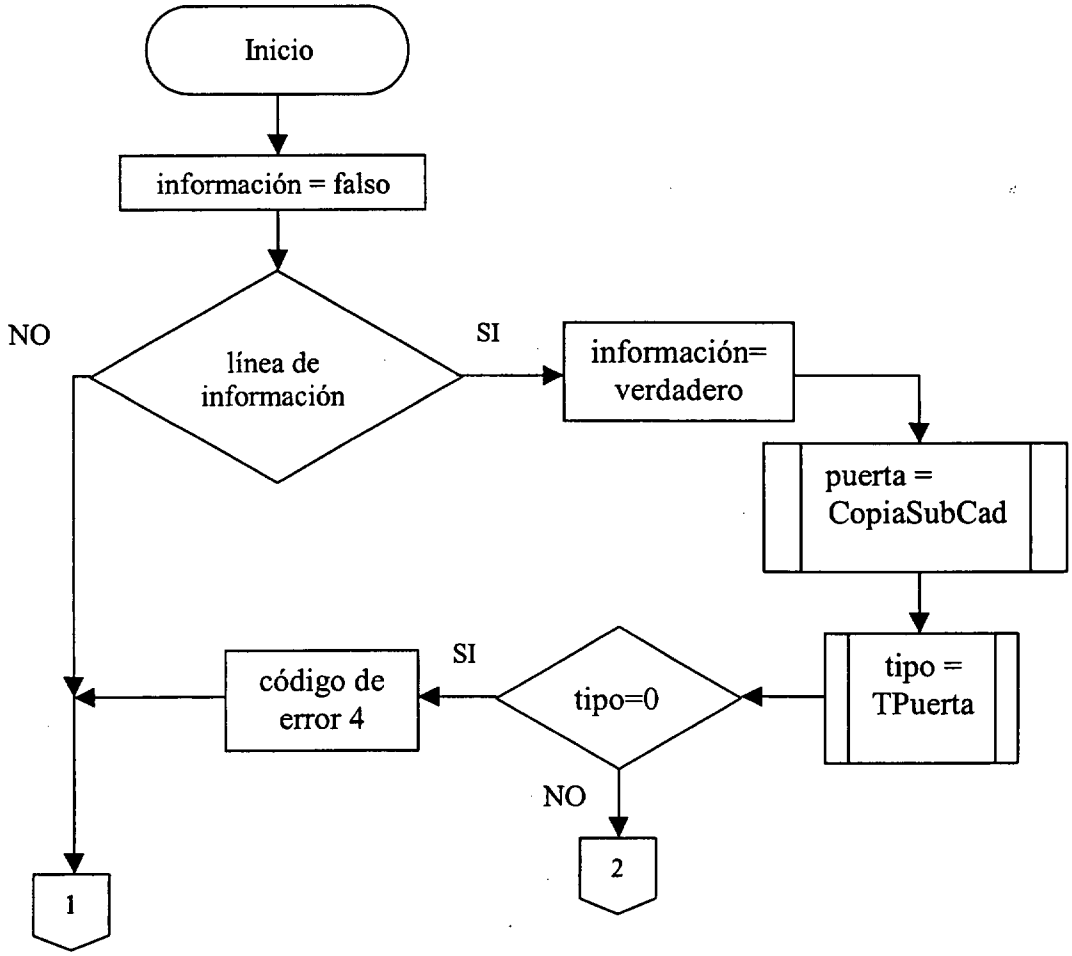
4.2.1.4.3. Procedimiento ObDatNet.

El procedimiento extrae de una línea del fichero *netlist* la información que ésta contiene, omitiendo las líneas de comentarios y realizando un proceso de análisis de dichos datos para comprobar su validez en cuanto a formato. Devuelve un parámetro a modo de indicador sobre el contenido de la línea de fichero: información o comentarios.

Comprueba que la puerta es aceptada por la aplicación, inversor o NOR2; verifica que ninguno de los nodos se conecta al nodo 0; y confirma el carácter numérico de los identificadores de los nodos y del valor de la relación de aspecto. Genera a partir de este proceso un código de error, que se presenta en la siguiente tabla:

Tabla 4. III. Códigos de error.

Código	Significado
0	No hay error(inicializado por el código llamador)
1	Identificador de nodo no numérico
2	Valor de β no numérico
3	Nodo conectado a 0
4	Puerta no soportada por la aplicación



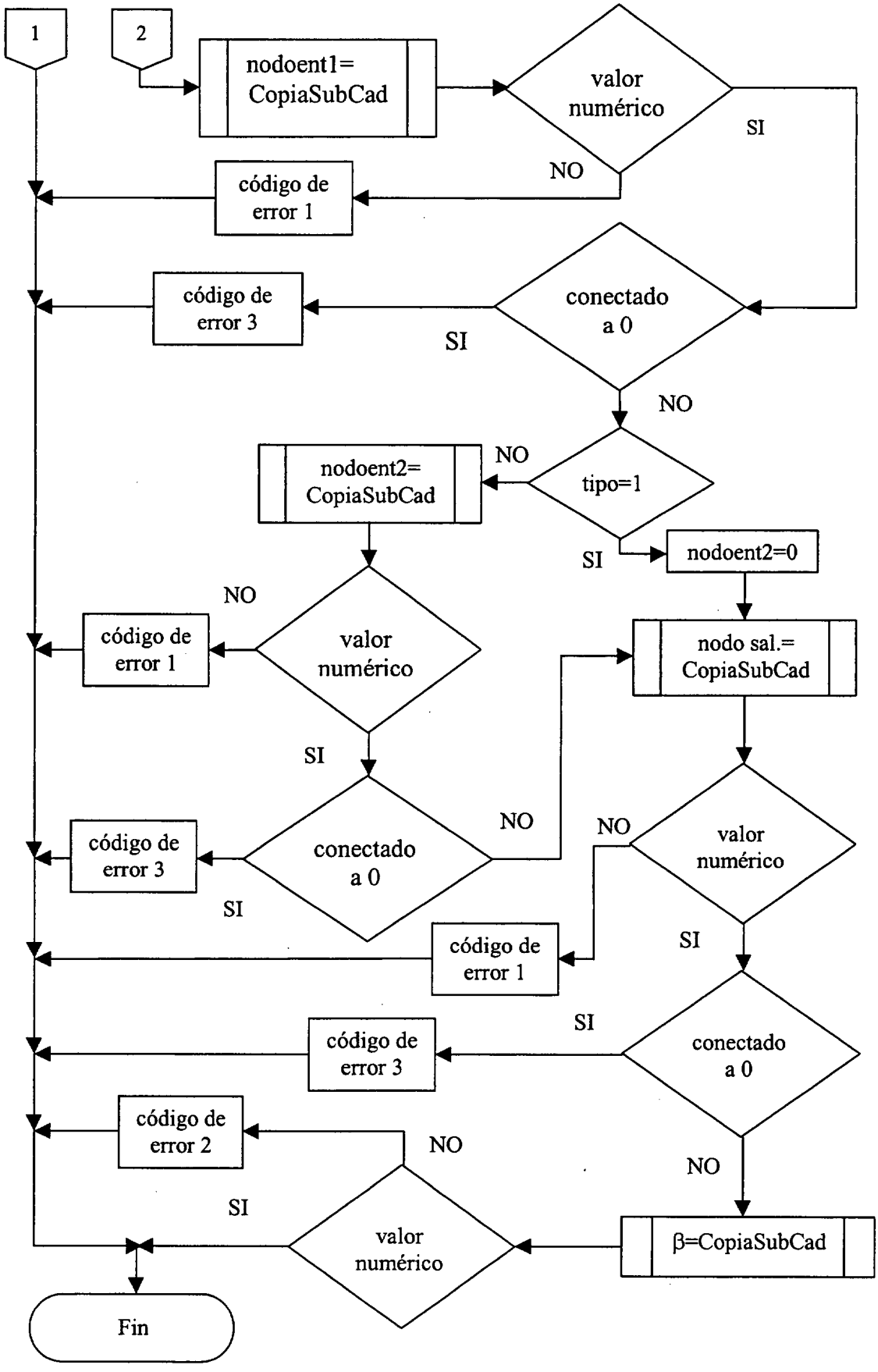


Figura 4. 3. Diagrama de flujos del procedimiento ObDatNet.

4.2.1.4.4. Función valorfich.

Esta función devuelve un valor de tipo cadena de caracteres correspondiente al parámetro de modelado que se especifique, que se le pasa como parámetro a la función. Dicho valor se extrae del fichero de la tecnología que también se le pasa como parámetro. Abre el fichero de texto y lo recorre hasta encontrar el parámetro buscado, devolviendo su valor. Si no encuentra el parámetro devuelve una cadena vacía.

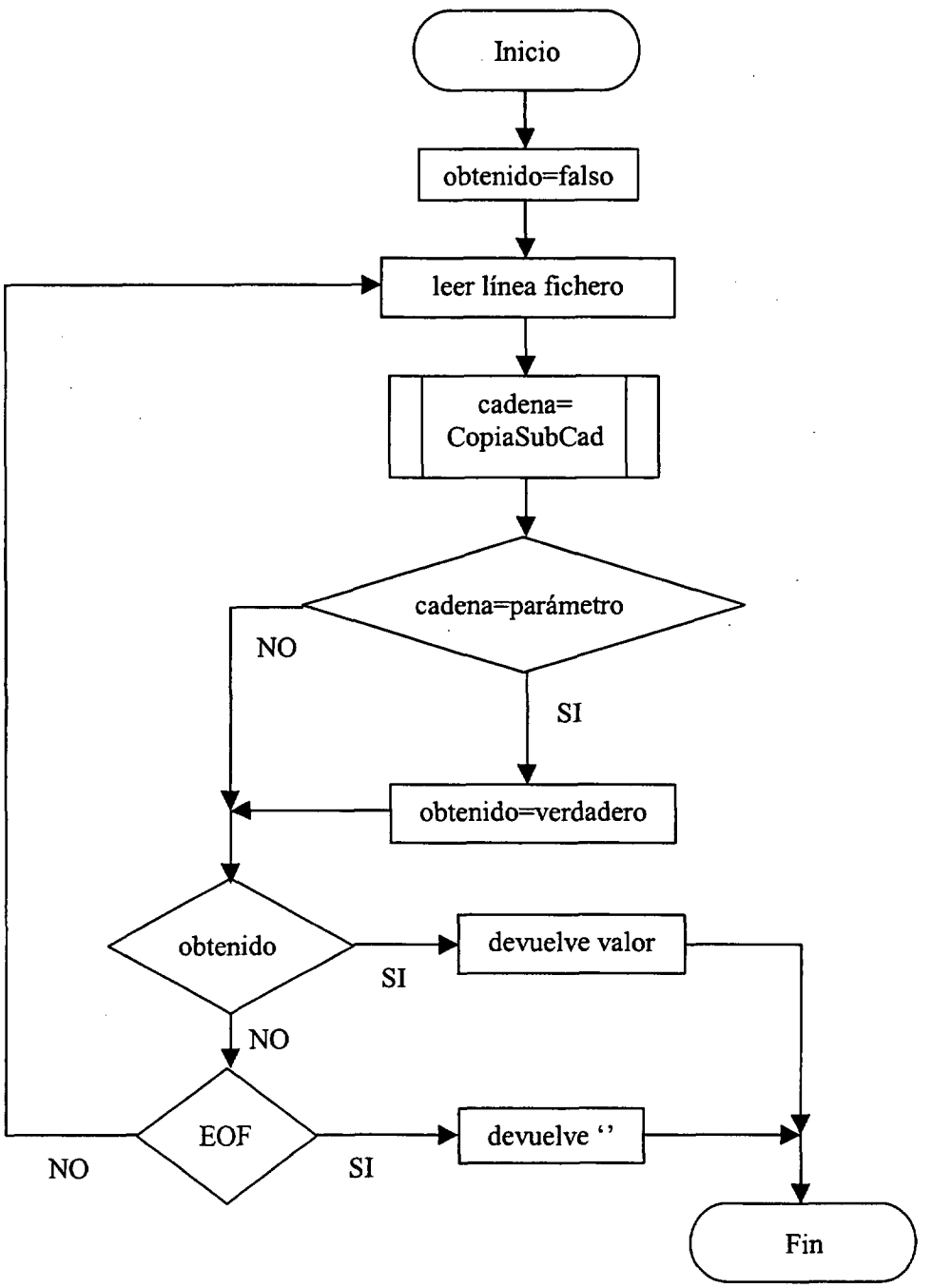


Figura 4. 4. Diagrama de flujos de la función valorfich.

4.2.1.4.5. Función valor_en_fich.

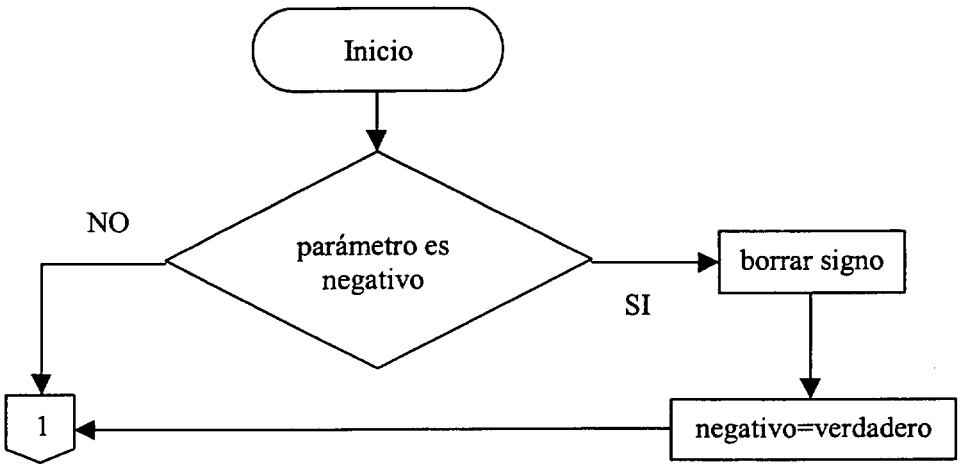
Devuelve un valor de tipo real correspondiente al parámetro del modelo que se especifica, contenido en el fichero de la tecnología. Mediante llamada a la función valorfich se obtiene el parámetro en formato de cadena de caracteres que se convierte a real empleando el procedimiento predefinido *val*. Si no se encuentra el parámetro devuelve 0.

4.2.1.4.6. Procedimiento LeerParametrosModelo.

Se llama a este procedimiento cuando se emplean las ecuaciones del modelo incluidas en la aplicación. Mediante sucesivas llamadas a la función valor_en_fich se obtienen del fichero de la tecnología los parámetros de modelado presentes en las ecuaciones, salvo β y FO, cuyos valores contiene el fichero *netlist*.

4.2.1.4.7. Procedimiento ParametroModelo.

Este procedimiento se llama cuando el programa utiliza para la evaluación de los retardos las ecuaciones introducidas desde teclado. Accede a los valores de los parámetros incluidos en la ecuación. Los valores se obtienen del fichero de la tecnología que esté usando la aplicación, salvo cuando se trata de β y el fan-out, cuyos valores se le pasan como parámetros al procedimiento.



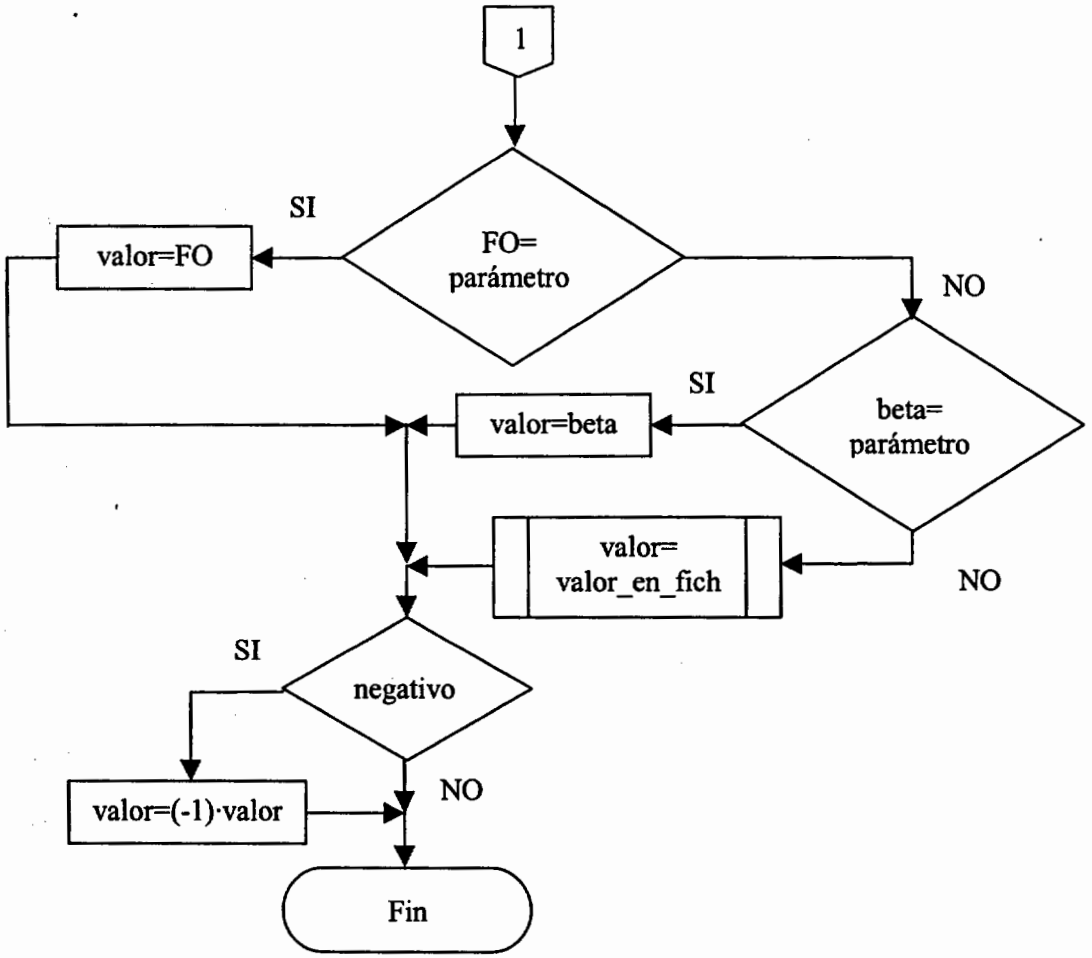


Figura 4. 5. Diagrama de flujos del procedimiento ParametroModelo.

4.2.1.4.8. Procedimiento Nuevo.

Crea un nuevo elemento de tipo Puerta para insertar en la lista. Mediante el procedimiento estándar *new* se asigna espacio en memoria suficiente para contener una variable de tipo Puerta, y se hace apuntar a dicha dirección a la variable de tipo PunteroPuerta que se le pasa como parámetro.

Recibe como parámetros datos con que inicializar los campos nodo_s, nodoe1, nodoe2, beta, pendiente y tp. Los restantes campos de información son puestos a 0, y los punteros a la constante predefinida nil que se utiliza para dar un valor a una variable puntero que no apunta a ninguna posición.

4.2.1.4.9. Procedimiento ListNetlist.

El procedimiento ListNetlist construye una lista doblemente enlazada de elementos de tipo Puerta. Inserta los elementos de forma ordenada, tomando como criterio para ordenarla el valor ascendente del nodo de salida de la puerta. Recibe como parámetros los datos que va a contener el elemento a incluir en la lista, así como los punteros al primer y último elementos de la lista. Conserva un puntero al último elemento insertado, que constituye un parámetro del procedimiento. Este es un puntero auxiliar usado para encontrar el punto de la lista donde incluir el nuevo elemento.

La inserción se efectúa creando enlaces entre los elementos de la lista a través de los correspondientes punteros de que consta cada elemento. A continuación se presenta el método seguido para ello.

Inicialmente se comprueba si la lista está vacía. En ese caso el nuevo elemento pasa a constituir el único de la lista y se hace apuntar a dicho elemento a los tres punteros: comienzo y final de lista, y el auxiliar que apunta al último elemento insertado.

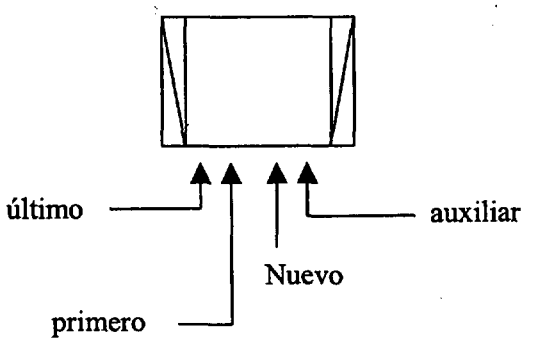


Figura 4. 6. Inserción en lista vacía.

Si no es así, se determina la posibilidad de insertar al comienzo de la lista. Para ello el identificador del nodo de salida del elemento a insertar ha de ser menor del que corresponde al primer elemento de la lista. En este caso el campo prox del nuevo elemento pasa a apuntar al primer elemento de la lista, cuyo campo ant y el puntero primero de lista apuntarán al nuevo elemento, que pasa a ser el elemento inicial de la lista.

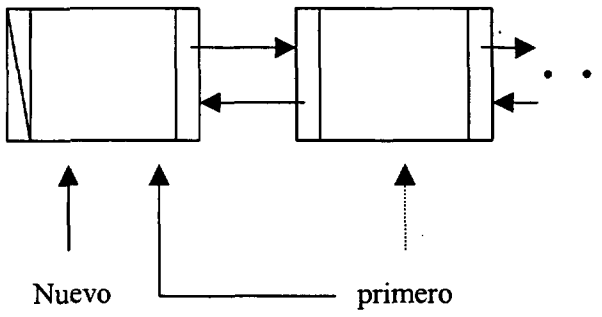


Figura 4. 7. Inserción por la cabeza de la lista.

La inserción se realiza al final de la lista si el nodo de salida del nuevo elemento es mayor que el perteneciente al último elemento de la lista. El campo ant del nuevo elemento se hace apuntar al último elemento de la lista, cuyo campo prox así como el puntero último elemento de lista pasan a apuntar al nuevo elemento que ocupa de esta forma la posición final de la lista.

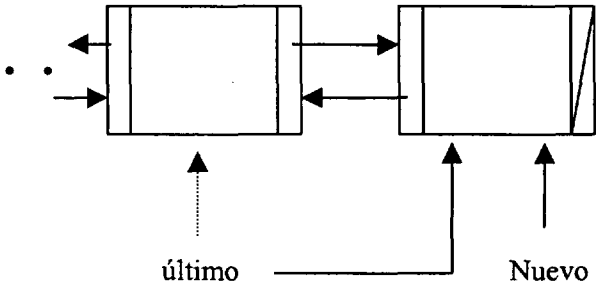


Figura 4. 8. Inserción por el final de la lista.

Por último, si no se produce ninguno de los casos anteriores, la inclusión del nuevo elemento se efectúa en el interior de la lista. Se hace uso del puntero auxiliar que apunta al último elemento insertado para dividir la lista en cuatro intervalos, con objeto de disminuir el número de iteraciones necesarias para encontrar el lugar donde situar el nuevo elemento. Se obtienen dos valores enteros intermedios entre el identificador del elemento apuntado por el puntero auxiliar y los elementos inicial y final de la lista.

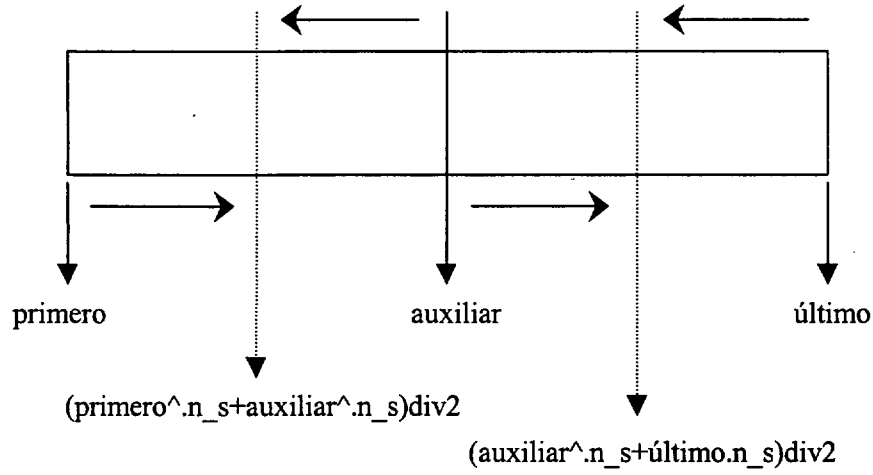


Figura 4. 9. Intervalos de búsqueda.

Se compara el identificador del nodo de salida del nuevo elemento con los valores intermedios y con el nodo de salida correspondiente al puntero auxiliar, con el fin de fijar el intervalo en que va a ser insertado el nuevo elemento. La búsqueda de la posición de inserción depende del intervalo en que se sitúe, como se muestra en la Tabla 4.IV. Se inicia en el puntero auxiliar o en el correspondiente a comienzo o final de lista. La búsqueda se realiza mediante desplazamiento del puntero auxiliar a través de la lista, desde uno de esos puntos iniciales, efectuándose en sentido ascendente, hacia valores crecientes del nodo de salida, o descendente, hacia valores decrecientes del nodo de salida, según la relación que guarde el nodo de salida del nuevo elemento con el correspondiente al puntero auxiliar o con los valores intermedios que se muestran en la Figura 4.10.

Tabla 4. IV. Sentido del desplazamiento en la lista en el proceso de inserción.

Relación (valores de nodo de salida)	**Sentido**	**Punto de inicio**
(auxiliar+último div2) >Nuevo > auxiliar	Ascendente	auxiliar
auxiliar > Nuevo > (primero+auxiliar div2)	Descendente	auxiliar
Nuevo > (auxiliar+último div2)	Descendente	último
(primero+auxiliar div2) > Nuevo	Ascendente	primero

Una vez hallado el punto de inserción, ésta se realiza estableciendo los pertinentes enlaces entre elementos de la lista. Se hace apuntar a los campos de tipo PunteroPuerta del nuevo elemento a los que pasan a ser sus elementos anterior y posterior, para a continuación romper el enlace entre éstos haciendo que apunten al nuevo elemento.

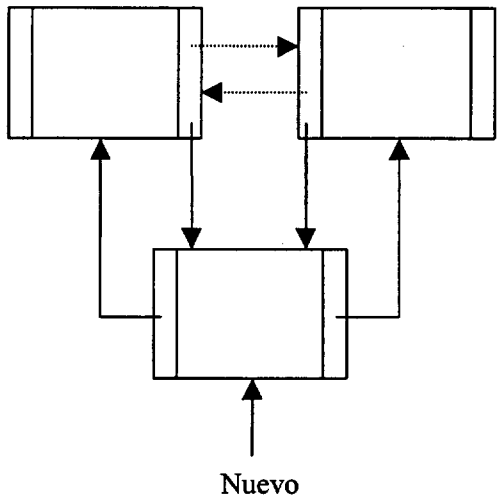
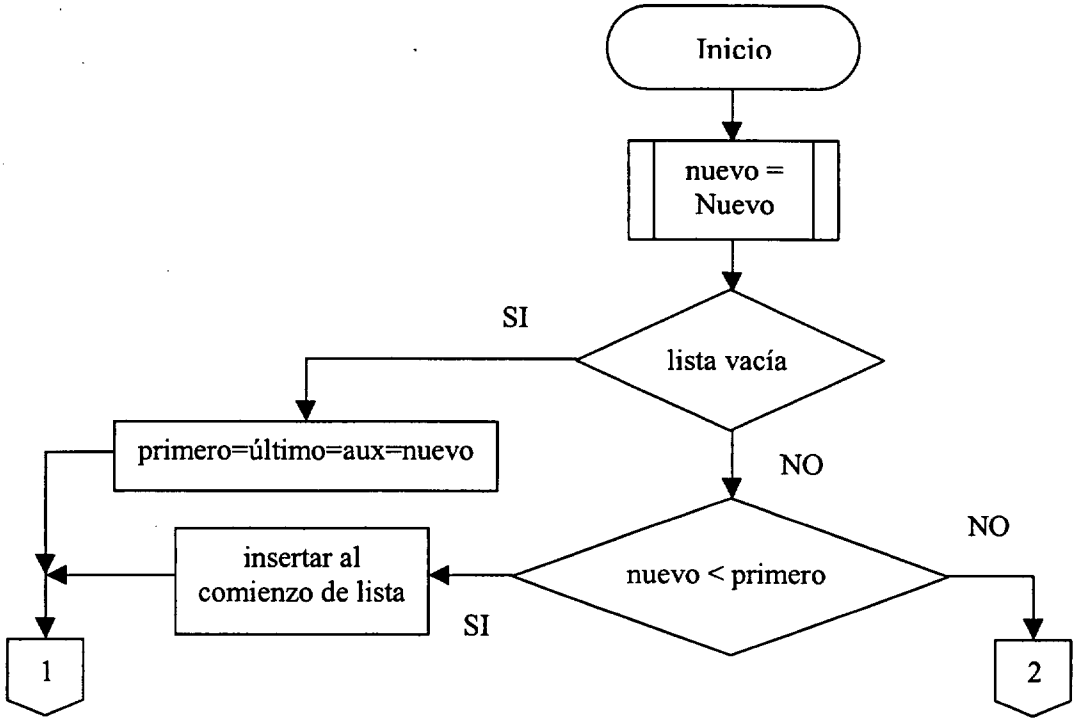


Figura 4. 10. Inserción en el cuerpo de la lista.



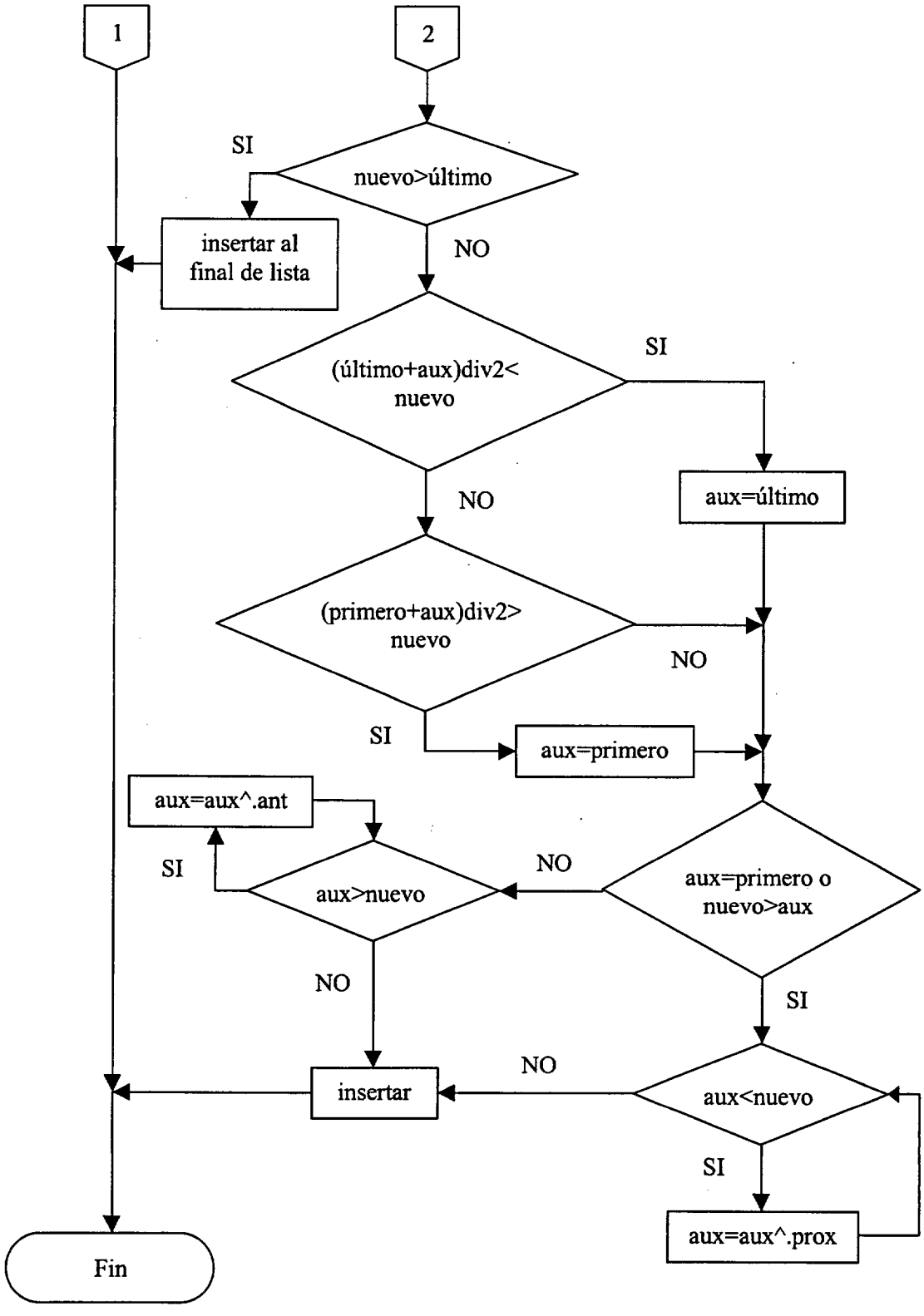


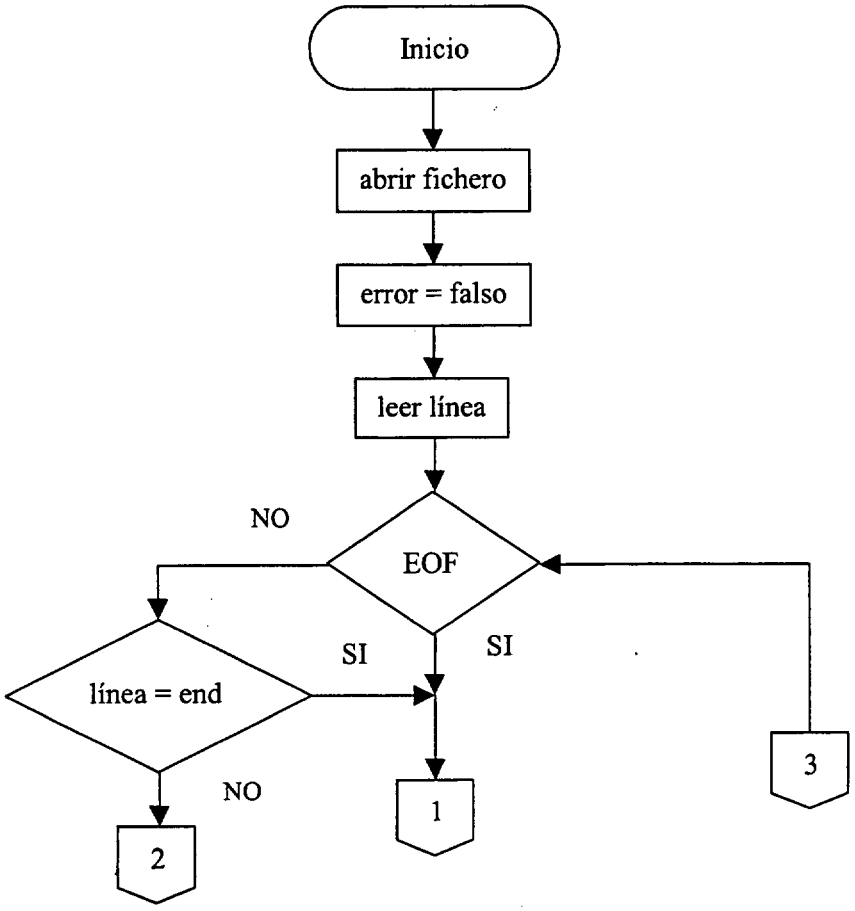
Figura 4. 11. Diagrama de flujos del procedimiento ListNetlist.

4.2.1.4.10. Procedimiento LeerEntradas

El procedimiento recoge los datos del fichero de entradas. La información de entrada se añade a la lista como si se tratase de puertas cuyos nodos de entrada se conectan al nodo 0. La entrada se considera como una puerta más del circuito cuyo estado, definido por la transición que presenta, ha sido resuelto. El procedimiento recorre secuencialmente el fichero de tipo texto, transformando su contenido de cadenas a valores enteros y reales, y los añade a la lista como elementos de la misma. Se aceptan como entrada la definición de una transición o un nivel fijo, bajo o alto.

Si se encuentra un valor incorrecto en los datos del fichero de entradas se devuelve información de error y se detiene el proceso de adquisición de los datos.

Para insertar dichos elementos en la lista se llama al procedimiento ListNetlist, pasándole como información de nodo de salida el nodo que recibe la excitación de entrada, así como la información temporal correspondiente a la pendiente y el retardo.



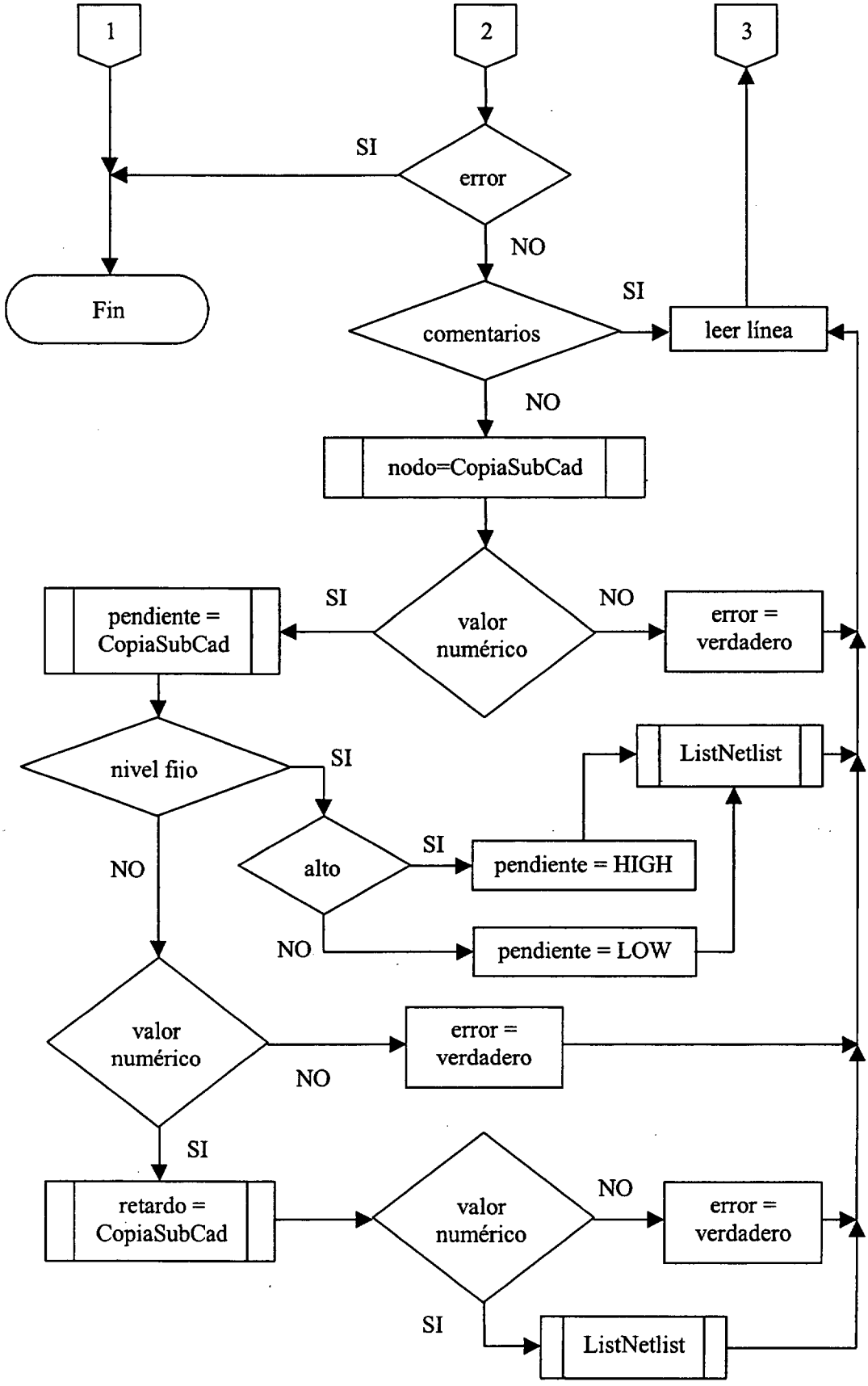


Figura 4. 12. Diagrama de flujos del procedimiento LeerEntradas.

4.2.1.4.11. Procedimiento Borrarlista.

Borra la lista enlazada, liberando el espacio de memoria que ésta ocupaba. El proceso se realiza por la cabeza de la lista, borrando secuencialmente la totalidad de los elementos de la misma. El borrado de los elementos concluye cuando el puntero al primer elemento alcance el valor nil que contiene el campo prox del último elemento de la lista.

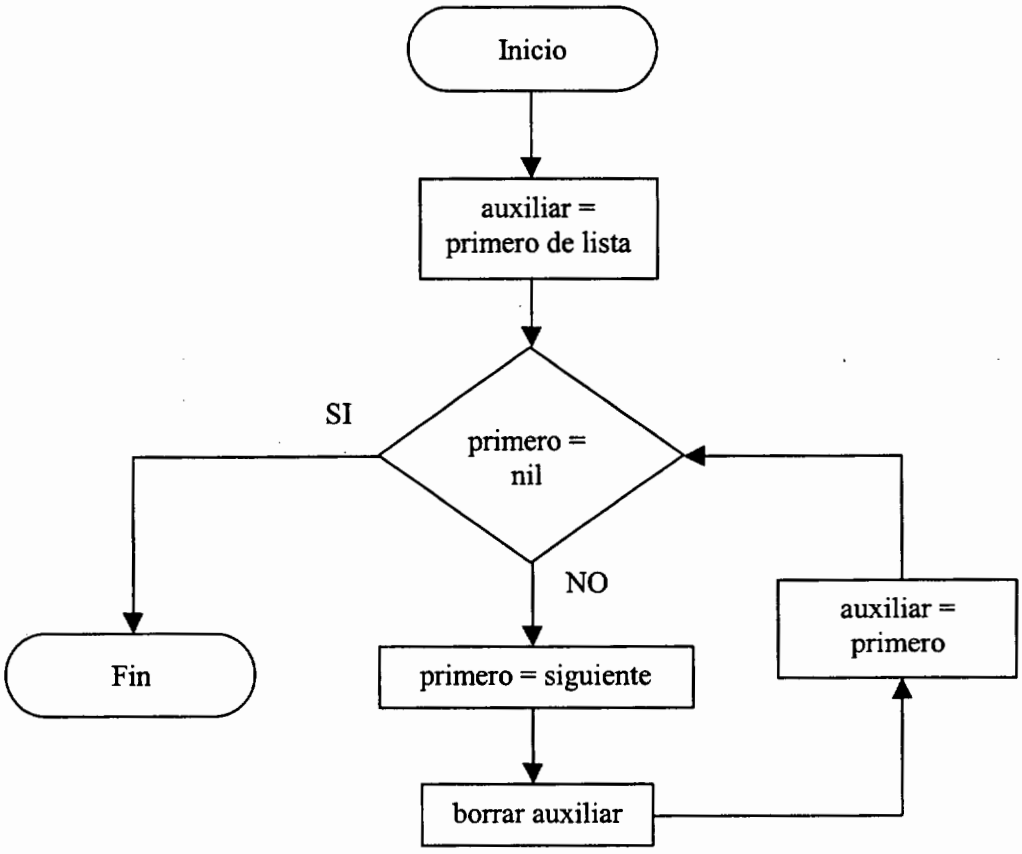


Figura 4. 13. Diagrama de flujos del procedimiento Borrarlista.

4.2.1.4.12. Procedimiento InicRetardos.

El programa soporta la posibilidad de aplicar sucesivos ficheros de entradas (*.ent) sobre el *netlist* actual. Para que pueda llevarse a cabo sin generar errores en la evaluación de los retardos, los campos de información tratados en el proceso de simulación, respcambio, pendiente y tp, son inicializados a 0 para cada elemento de la lista mediante la realización de un recorrido completo de la misma. Se ha hecho uso de un puntero auxiliar

para recorrer la lista, saltando de un elemento de la misma a otro a través del campo prox, de tipo PunteroPuerta, de cada elemento de la lista.

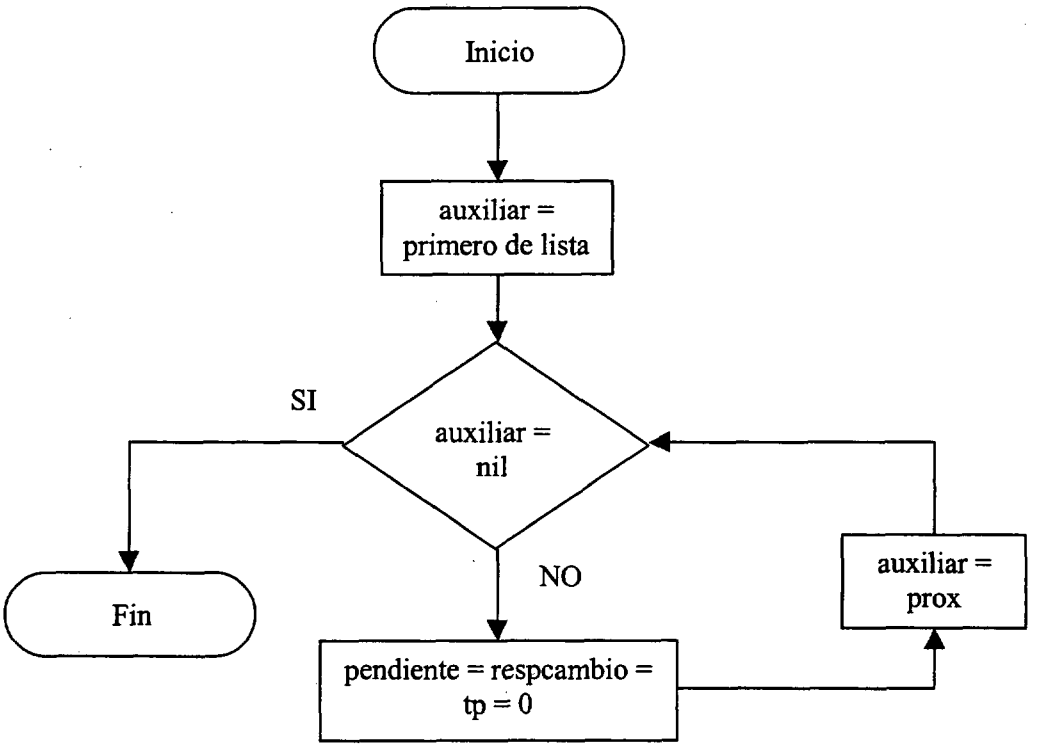


Figura 4. 14. Diagrama de flujos del procedimiento InicRetardos.

4.2.1.4.13. Procedimiento BorrarEntradas.

La información contenida en el fichero de entradas se añade a la lista como si se tratase de puertas. Los elementos de la lista que corresponden a entradas tienen sus nodos de entrada, campos nodoe1 y nodoe2, a 0, característica que se utiliza para su reconocimiento. Este procedimiento recorre la lista completa eliminando los elementos identificados como entradas.

Se estructura en dos bucles, el primero elimina aquellas entradas que sucesivamente figuren como cabeza de lista, y el segundo borra las entradas incluidas en el cuerpo de la lista. En el primer caso, el campo ant del nodo que sigue al que se pretende borrar es fijado a la constante nil, que lo identifica como el nuevo comienzo de lista, además de hacer apuntar al puntero primero de lista a dicho elemento. En el segundo caso, se establecen enlaces entre los elementos anterior y posterior al que se va a eliminar con objeto de

mantener la lista. Si el elemento a borrar fuese el último de la lista se fija como elemento final el anterior al mismo.

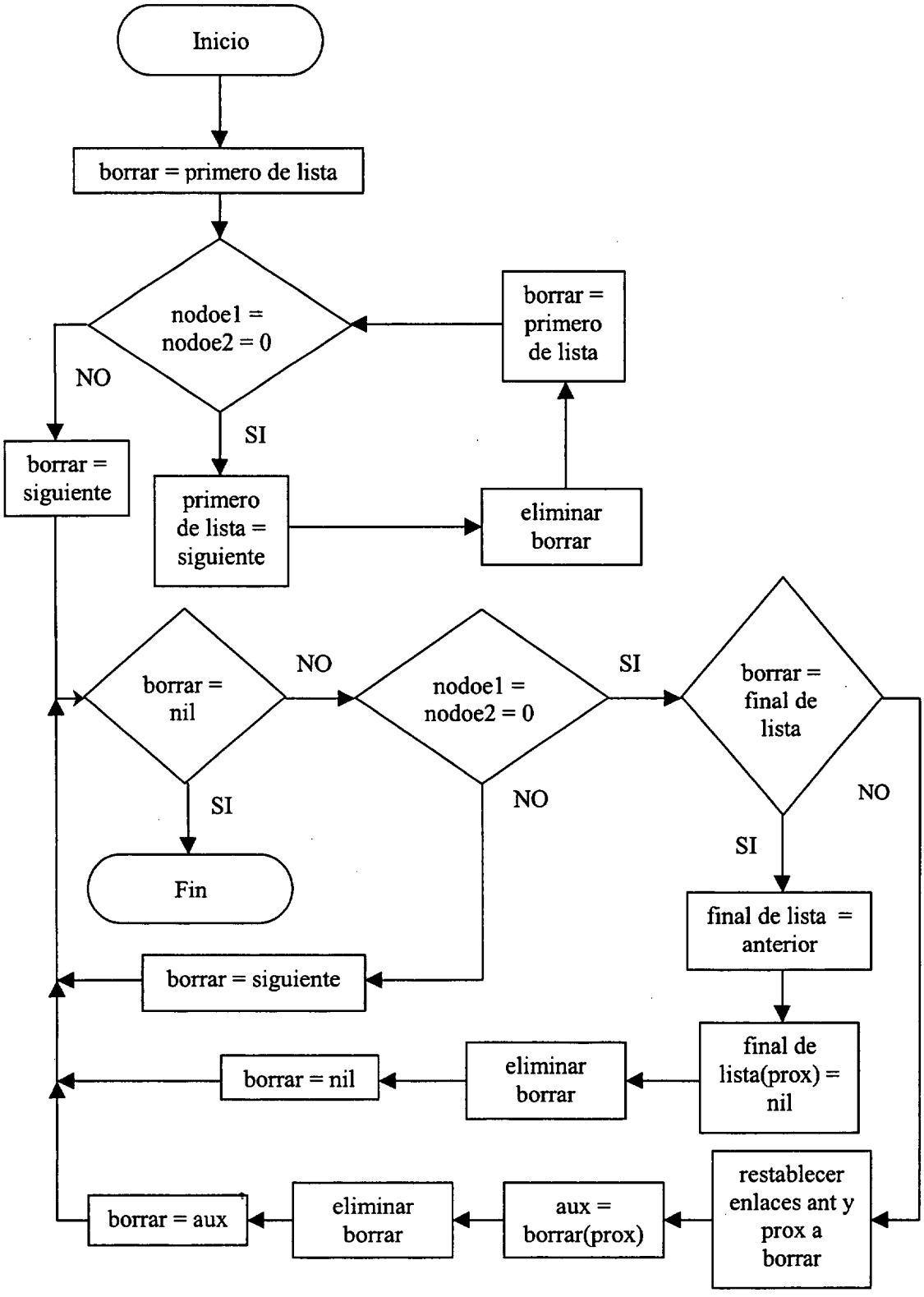


Figura 4. 15. Diagrama de flujos del procedimiento BorrarEntradas.

4.2.1.4.14. Procedimiento Buscar.

El procedimiento busca el elemento de la lista cuyo identificador de nodo de salida recibe como parámetro. Se utiliza en la evaluación de una puerta de tipo NOR2 para hallar el elemento de la lista enlazada que corresponde a una de las entradas de la puerta. El procedimiento devuelve un puntero a dicho elemento si lo encuentra. En caso contrario el puntero apuntará a la constante nil. Recibe como parámetros punteros al comienzo y final de la lista, y al elemento que representa a la puerta NOR2 cuyo nodo de entrada se busca.

Se determina sobre qué parte de la lista se efectúa la búsqueda mediante la comparación del valor buscado con el nodo de salida correspondiente a la puerta NOR2.

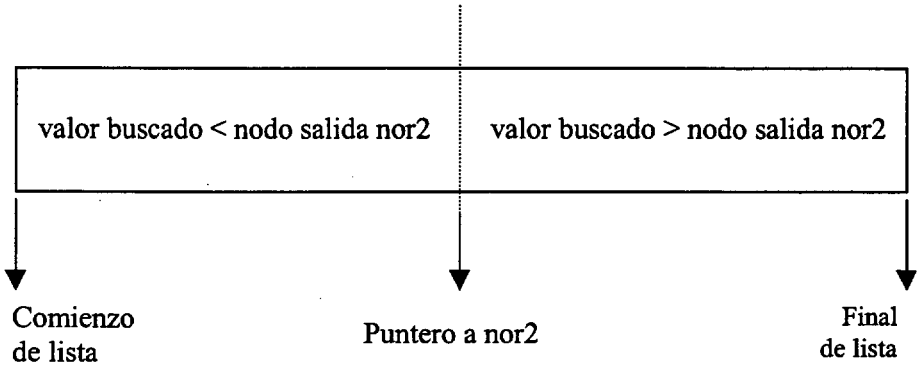


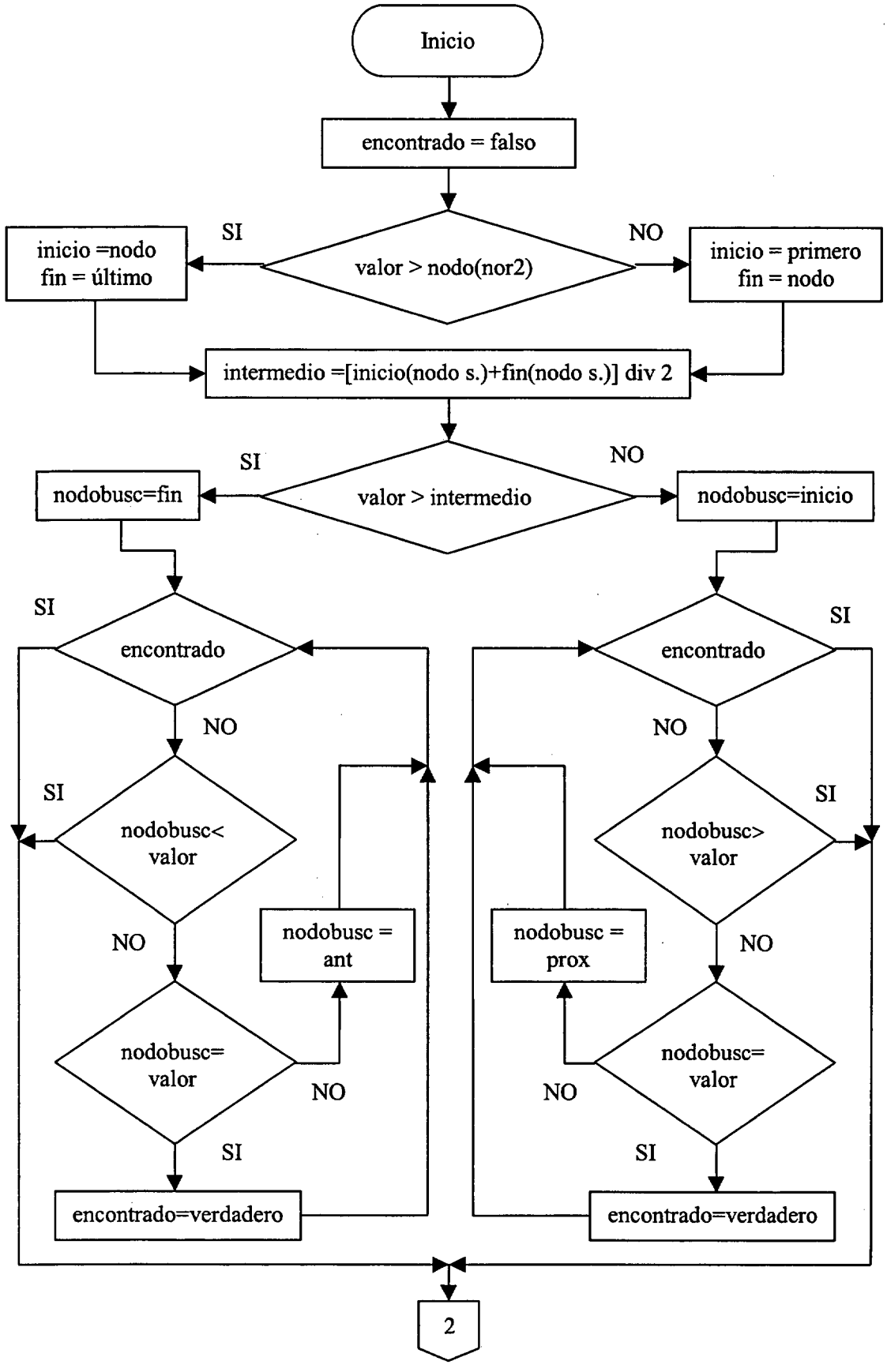
Figura 4. 16. Intervalos de búsqueda.

Una vez definido el intervalo de búsqueda, se obtiene el valor intermedio del mismo dividiendo entre dos la suma de los nodos de salida correspondientes a los elementos inicial y final del intervalo. Se realiza una división entera:

Valor intermedio = [innicio^.nodo_s + fin^.nodo_s] **div** 2

El valor buscado se compara con el valor obtenido para determinar el subintervalo en que se encuentra, lo cual establece el sentido en que se realiza la búsqueda: valores ascendentes de nodo de salida desde el puntero que marca el inicio del intervalo ó descendentes desde el puntero final con objeto de reducir el número de iteraciones preciso para dar con el elemento buscado.

Este tipo de búsqueda es posible gracias a que los elementos de la lista se encuentran ordenados.



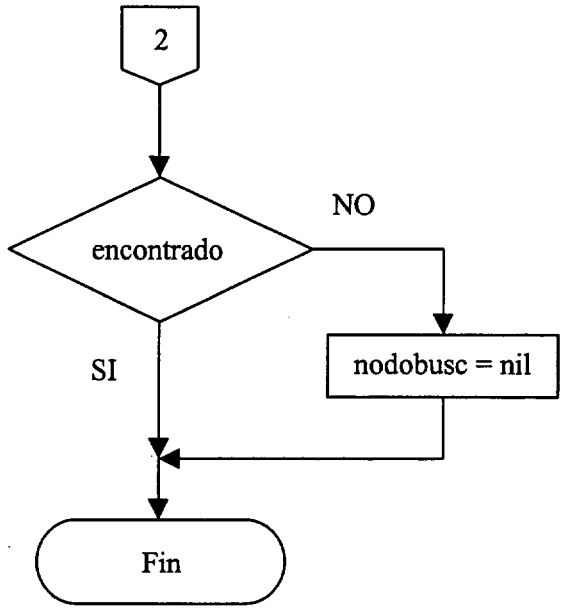


Figura 4. 17. Diagrama de flujos del procedimiento Buscar.

4.2.1.2.15. Función ConexCorrecta.

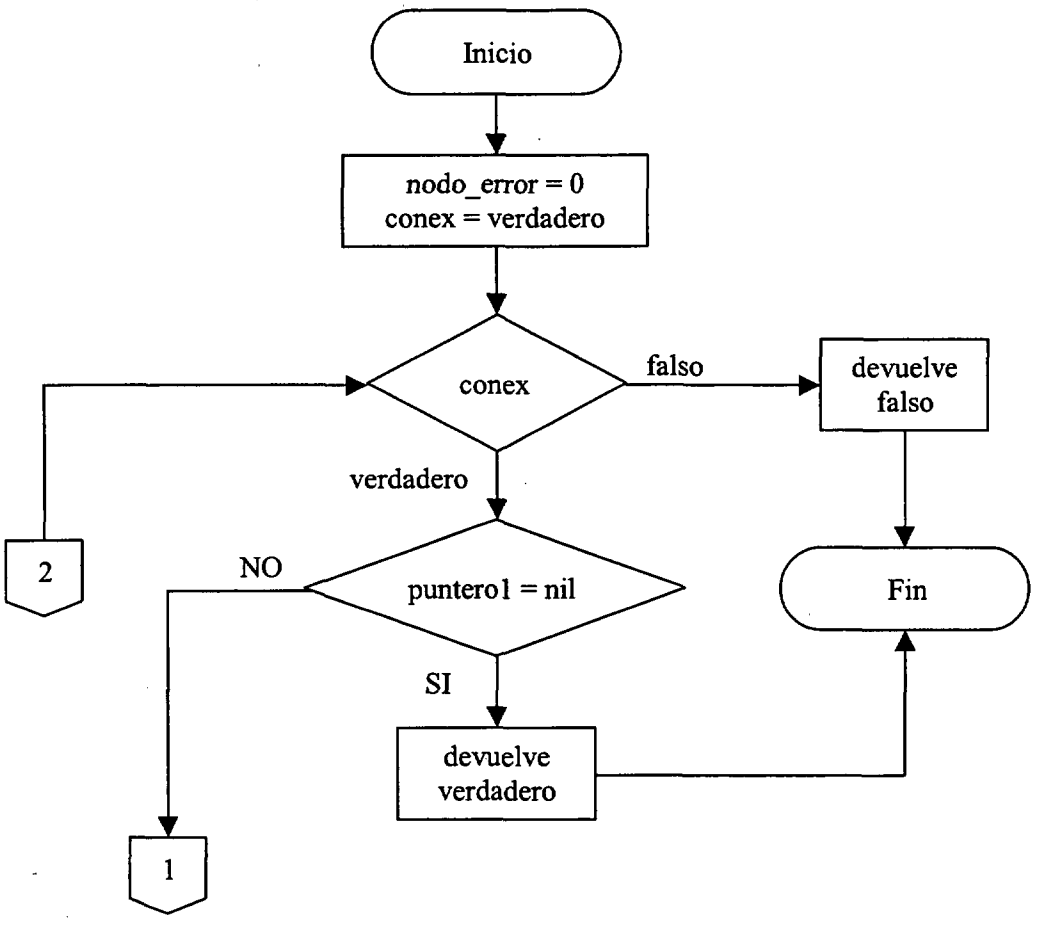
La función ConexCorrecta, que devuelve un valor de tipo lógico, se define para verificar que todo nodo de entrada al circuito recibe una señal y la correcta conexión interna de los nodos del circuito. Esta última labor consiste en comprobar que cada nodo de entrada a una puerta está conectado a otro nodo. Mediante un puntero definido de forma local a la función se recorre la lista enlazada que representa al *netlist* para comprobar ambos aspectos. Esta lista contiene también las entradas a simular, como se explica en el punto 4.2.1.4.10 referido al procedimiento LeerEntradas. La función devuelve un valor lógico falso si encuentra un error en las conexiones del *netlist* o en las entradas definidas.

La función recibe como parámetros los punteros que apuntan a los elementos inicial y final de la lista enlazada. Se define un parámetro de tipo entero para devolver el identificador del nodo de salida de la puerta que presenta el error con objeto de informar al usuario.

Cuando el puntero apunta a un elemento de la lista se comprueba si éste representa a una entrada al circuito o a una puerta del mismo. En el primer caso el campo nodoel

contiene el valor 0 y al puntero se le asigna la dirección el siguiente elemento de la lista. Si corresponde a una puerta se busca al miembro de la lista cuyo nodo de salida sea igual al contenido de nodoe1. Independientemente de que la puerta sea una entrada al circuito o no, debe existir el elemento para una correcta simulación. Se llama al procedimiento Buscar para ello. Si se encuentra se comprueba si el elemento apuntado por el puntero representa a una puerta NOR. En este caso el campo nodoe2 será distinto de 0 y se realiza el mismo proceso que para nodoe1. En el caso de que se trate de un inversor, o si tiene éxito esta segunda búsqueda se hace apuntar al puntero al siguiente miembro de la lista enlazada. Si en uno de los dos casos no se encuentra el elemento buscado se asigna el valor falso a un indicador lógico definido localmente a la función.

El proceso de comprobación finaliza cuando se haya recorrido completamente la lista si no se detecta error, devolviendo la función el valor verdadero. Si existe error se detiene el recorrido y se devuelve el valor falso.



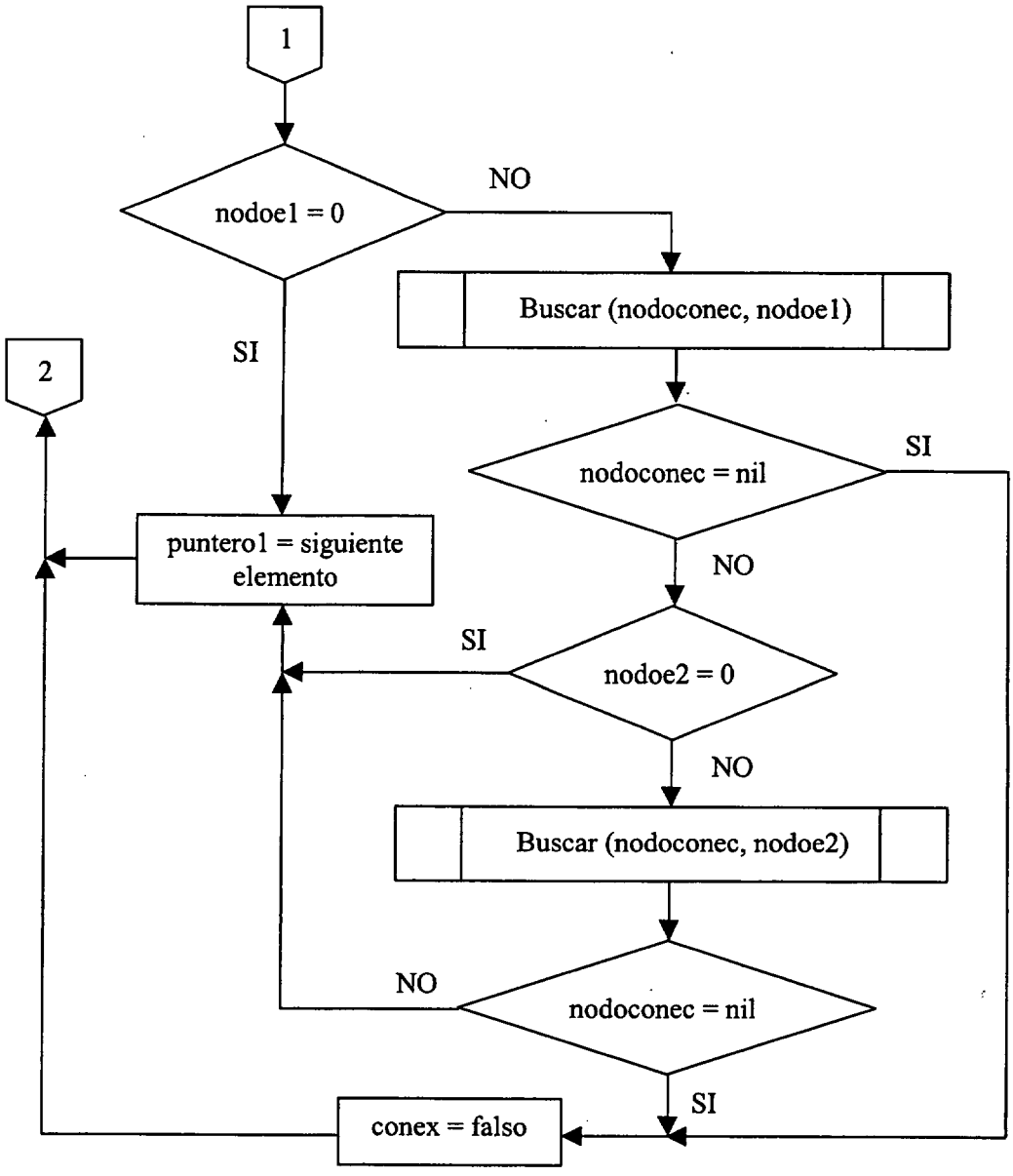


Figura 4. 18. Diagrama de flujos de la función ConexCorrecta.

4.2.1.4.16. Procedimiento CalcFanout.

El procedimiento CalcFanout calcula el fan-out para cada puerta del circuito aplicando la ecuación 2.10 del apartado 2.4.1. El valor obtenido de esta forma se asigna al campo fanout del elemento de la lista que representa a la puerta. Se utilizan dos punteros, uno apunta al elemento cuyo fan-out se pretende calcular, y otro realiza el recorrido completo de la lista para comprobar las conexiones del nodo de salida de la puerta y realizar el cómputo.

Si se detecta una puerta conectada al nodo de salida se suma el valor de β de la misma al contenido del campo fanout que se calcula. Se tiene en cuenta la posibilidad de que la salida se conecte a las dos entradas de una NOR2 para crear un inversor. Por ello el valor del campo nodo_s del elemento cuyo fan-out se calcula se compara con los campos nodoe1 y nodoe2 del elemento al que apunta el puntero auxiliar de forma independiente. Al finalizar el recorrido de la lista se divide el valor del campo fanout por el valor de la relación de aspecto de la puerta para obtener el fanout de acuerdo al concepto empleado.

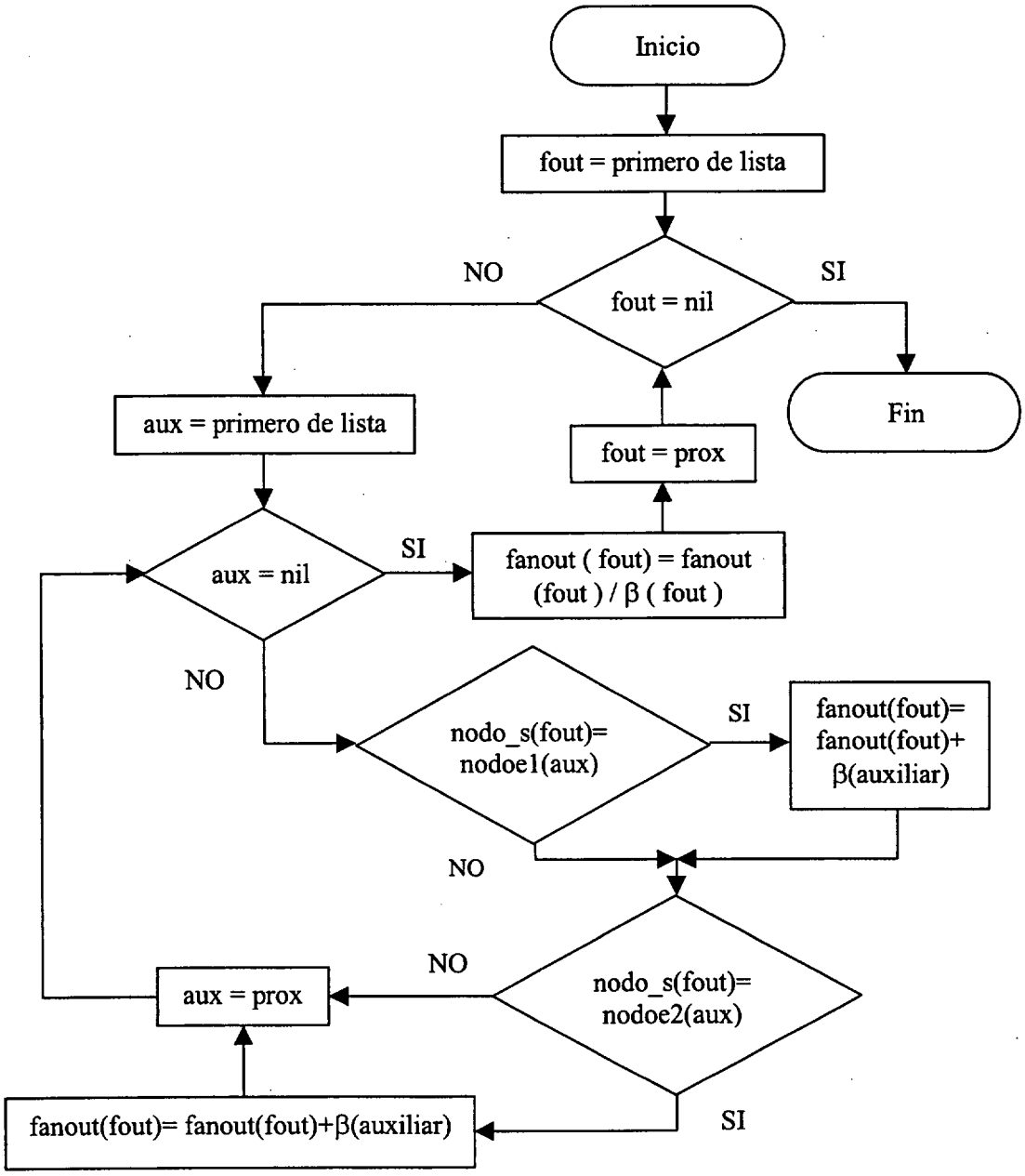


Figura 4. 19. Diagrama de flujos del procedimiento CalcFanout.

4.2.1.4.17. Función ValorOperando.

Esta función se utiliza cuando la simulación emplea ecuaciones de teclado para la evaluación de los tiempos de propagación. Devuelve un valor real correspondiente a un operando de una expresión aritmética, que se pasa a la función como parámetro de tipo cadena, *string*.

El operando puede ser un parámetro de modelado, en cuyo caso se obtiene su valor, de tipo real, mediante llamada al procedimiento ParametroModelo. Si se trata de un valor literal se utiliza el procedimiento estándar *val*, que convierte una cadena de caracteres a variable numérica.

4.2.1.4.18. Función operar_mat.

Realiza la operación aritmética entre los dos operandos que recibe como parámetros de tipo cadenas de caracteres. La operación matemática a realizar se le especifica a través de un parámetro de tipo carácter que contiene el operador de la expresión. Una sentencia de elección entre diferentes alternativas de tipo *case* sobre el parámetro operador se emplea para seleccionar la operación a realizar entre los operandos, que se convierten a datos numéricos por llamadas a la función ValorOperando. La función devuelve el resultado de la operación en formato real.

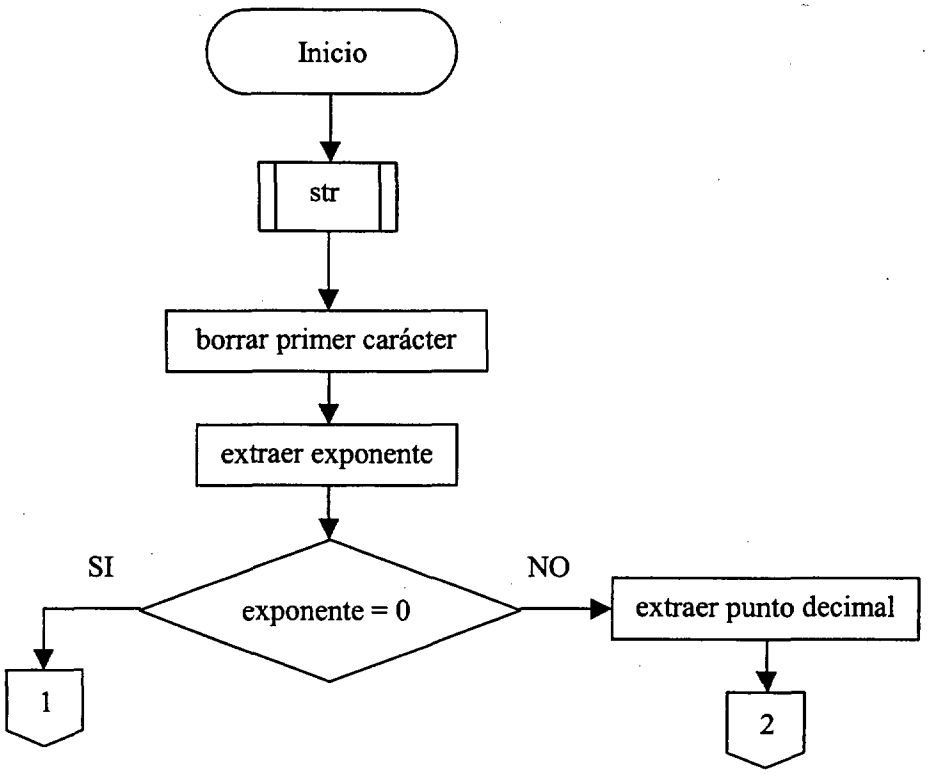
4.2.1.4.19. Función rcad.

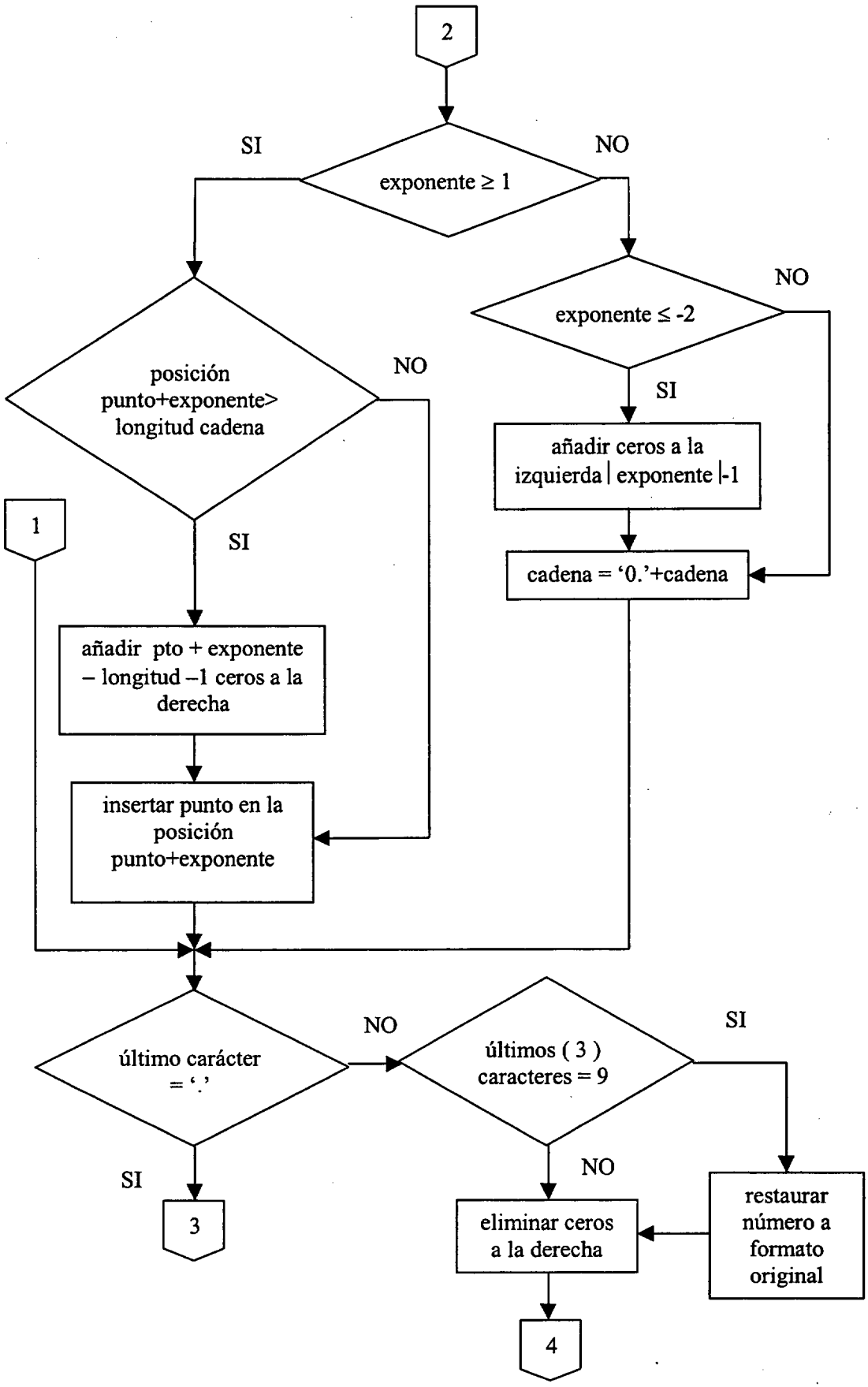
La función rcad devuelve una cadena de caracteres equivalente al dato de tipo real que recibe como parámetro. La cadena representa al número en notación de coma fija. Se ha escogido esta notación en lugar de la de coma flotante, mantisa y exponente, que resulta de aplicar el procedimiento estándar *str* porque éste incluye signo en el exponente, aún siendo éste positivo. Este hecho puede originar errores al interpretarse el signo como operador matemático por los subprogramas creados para evaluar el *tiempo de propagación* cuando se emplean ecuaciones temporales introducidas desde teclado.

La función rcad convierte el dato numérico a cadena mediante el procedimiento *str* y trabaja sobre la cadena resultante para obtener una representación en notación de coma fija. Para ello, se extraen de la cadena los caracteres correspondientes al exponente y se traslada el punto decimal a derecha o izquierda según el signo y valor del exponente.

La cadena generada por *str* tiene como parte entera de la mantisa un número distinto de cero, a menos que el dato real lo sea. Si el exponente es mayor o igual que 1, se traslada el punto decimal a la derecha tantas posiciones como indique añadiendo ceros a la derecha si fuera preciso en función de la magnitud del exponente. Si es menor o igual que -1 se mueve hacia la izquierda añadiendo por la cabeza de la mantisa ceros en un número inferior en una unidad al valor absoluto del exponente.

La mantisa generada por el procedimiento str tiene una longitud fija. Si el número real a representar presenta una cantidad de dígitos decimales pequeña el procedimiento str completa la longitud de la mantisa añadiendo ceros, o restando 1 al último dígito decimal y completando la cadena con repeticiones del carácter 9. La función rcad considera ambas posibilidades y restituye el número a su formato original eliminando dichos caracteres.





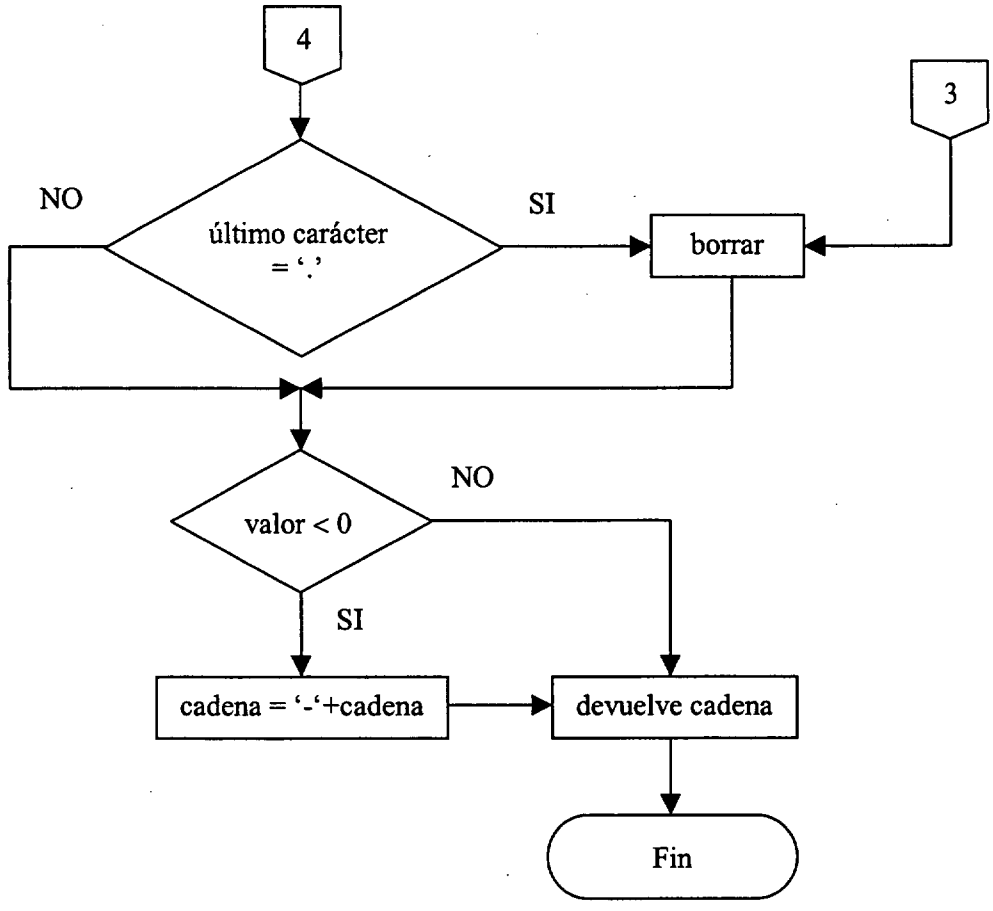


Figura 4. 20. Diagrama de flujos de la función rcad.

4.2.1.4.20. Función evaluar.

La función se emplea en la evaluación de expresiones introducidas desde teclado y devuelve una cadena de caracteres como resultado. Se aplica sobre expresiones aritméticas constituidas por un operador y dos operandos de tipo cadena de caracteres que se le pasan como parámetros.

La función distingue entre la etapa de simplificación, en la que se busca dejar la ecuación de modelado como función exclusivamente de β y el fan-out, y la posterior evaluación de los tiempos de propagación de cada puerta. Para diferenciar ambas fases se utiliza una variable de tipo lógico que se le pasa como parámetro.

En la etapa de evaluación se hace uso de las funciones operar_mat y rcad para obtener la cadena equivalente al resultado de la expresión aritmética.

En la fase de simplificación las expresiones que contienen como operando la relación de aspecto, 'beta', o el fan-out, 'FO', no se evalúan. Se comprueba el contenido de la expresión y si ambos operandos no son los citados la expresión se resuelve de igual forma que en la etapa de evaluación. Si alguno de los operandos es uno de los dos parámetros citados, la función devuelve una cadena que identifica a la expresión aritmética tratada como función del parámetro, y por lo tanto no resuelta. La expresión no evaluada es memorizada y devuelta por la función como parámetro por referencia.

El procedimiento llamador puede estar evaluando una expresión como:

$$0.31 \cdot (FO-1)$$

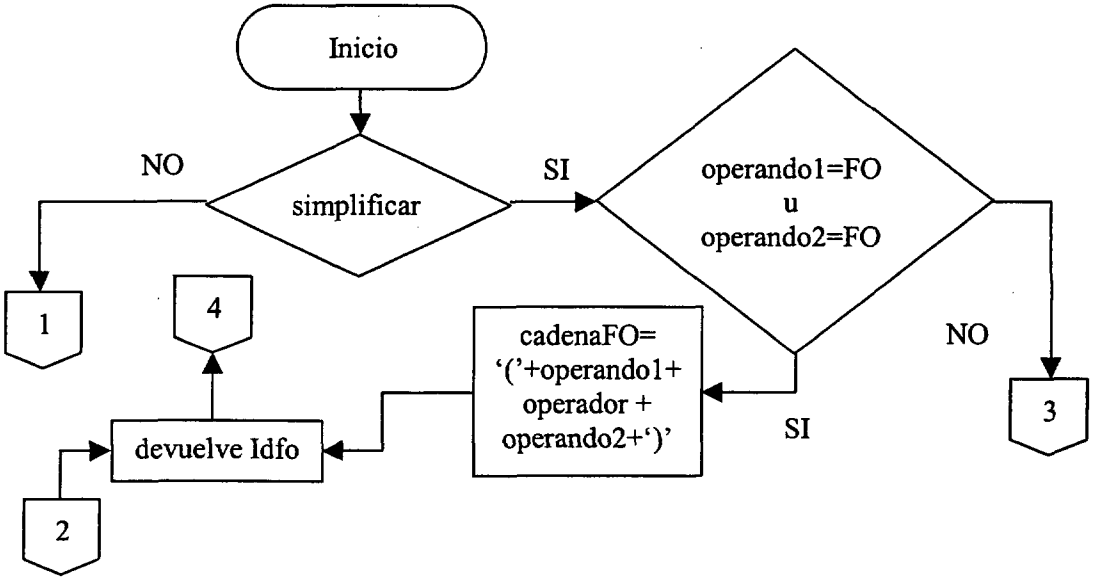
Inicialmente pasará a la función la expresión FO-1, por orden de precedencia de evaluación, que incluye como operando 'FO', con lo que no es evaluada. La expresión es memorizada y devuelta como parámetro por referencia, y como resultado la función devuelve 'Idfo', identificador de expresión en FO. De esta forma en el procedimiento llamador se tendrá:

$$0.31 \cdot Idfo$$

como expresión a evaluar, y (FO-1) como expresión no evaluada y memorizada.

En la siguiente llamada a evaluar, '0.31·Idfo' no será evaluada, pues contiene una expresión no resuelta 'Idfo'. La función devuelve como resultado el identificador de expresión, y la expresión memorizada pasa a ser:

$$(0.31 \cdot (FO-1))$$



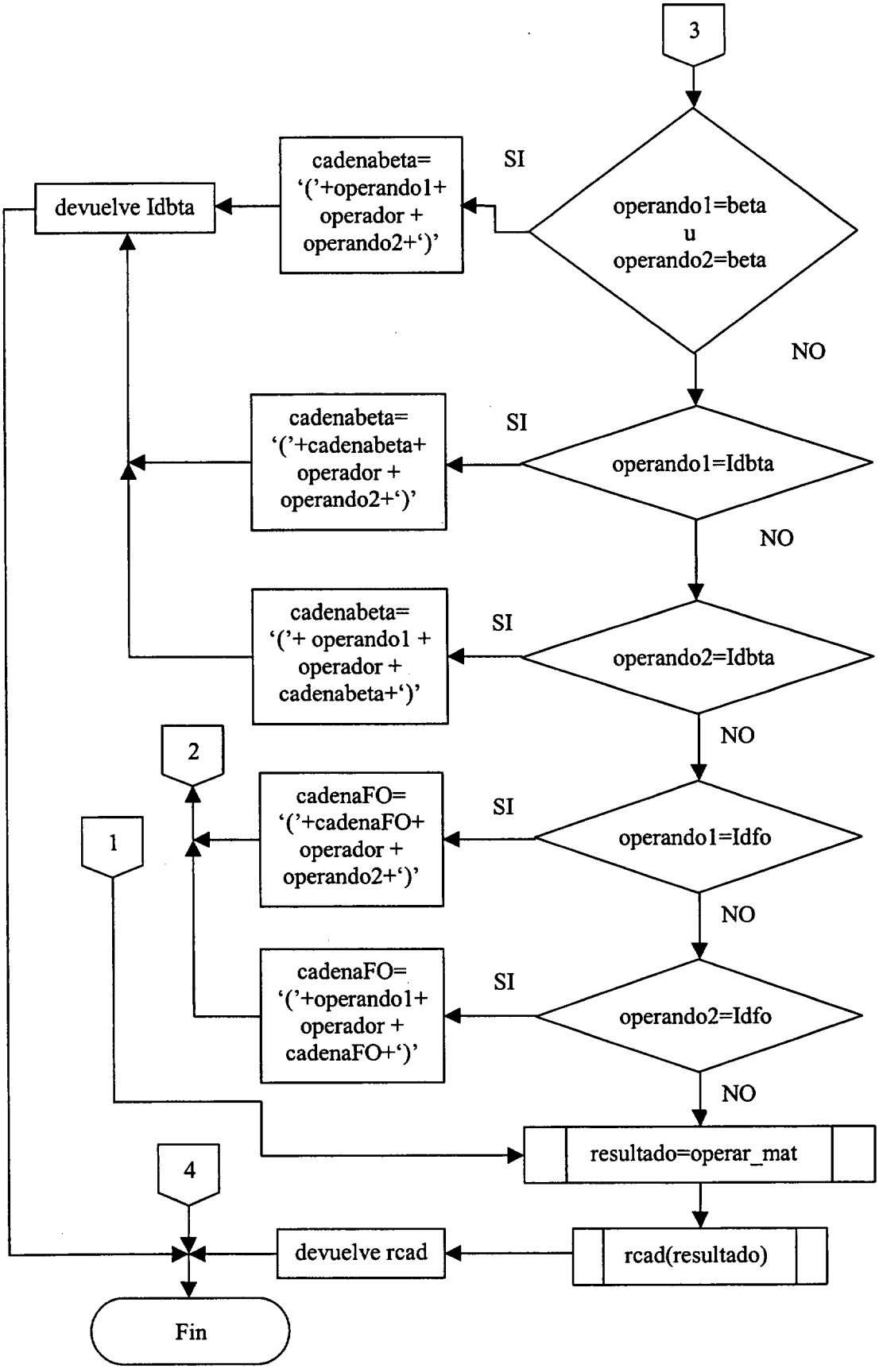


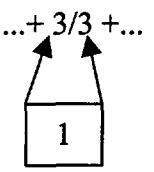
Figura 4. 21. Diagrama de flujos de la función evaluar.

4.2.1.4.21. Función TratarEcuacion.

Cuando el usuario emplea una ecuación externa se efectúa una simplificación de la misma previa a su aplicación sobre el circuito. La función TratarEcuacion realiza la simplificación de la ecuación, y se utiliza también para el cálculo de los retardos mediante el empleo de la ecuación reducida. Devuelve en forma de cadena de caracteres el resultado tras evaluar la ecuación.

El procesamiento de la ecuación consiste en la extracción de subexpresiones que son resueltas y sustituidas por sus resultados. El tipo *string* en Delphi32 corresponde por defecto al tipo *AnsiString* y no tiene fijada una longitud máxima, sino que ésta corresponde al número de caracteres que contenga en cada momento. La sustitución directa supone un método sencillo de almacenamiento de los resultados parciales sin necesidad de recurrir a otras estructuras de datos ni complejos algoritmos de cómputo.

La evaluación de la ecuación concluye cuando en su contenido no se detecte la presencia de operadores matemáticos. En el ejemplo que se muestra la subexpresión 3/3 se sustituye por su resultado, 1.



Las subexpresiones se encuentran contenidas entre paréntesis, evaluándose primero los paréntesis más internos en caso de estar anidados, y de izquierda a derecha. Esto es debido a que la búsqueda de las subexpresiones se realiza detectando la posición en la cadena de la primera aparición del carácter ')'. Una vez encontrado, por medio de un bucle se retrocede en la cadena hasta la apertura del paréntesis. Se extrae su contenido y se evalúa. Si la expresión no cuenta con paréntesis se procede a su evaluación completa.

Para el cálculo de la subexpresión se buscan los operadores matemáticos que ésta posea. Si no se encuentran es que ha sido resuelta, con lo que el resultado sustituye en el cuerpo de la ecuación a aquella.

Cuando se detecta el primer operador se busca en la subexpresión un segundo operador con objeto de delimitar los operandos. Si no existe segundo operador se resuelve la operación matemática, concluyendo la evaluación de la subexpresión. En el caso que el segundo operador exista, se compara con el primero a fin de establecer qué operación se ejecuta primero de acuerdo con el orden de precedencia de evaluación establecido. Si el primer operador resulta prioritario se realiza la operación y se sustituye por el resultado obtenido. Si resulta prioritario el segundo, se busca un tercer operador para delimitar el segundo operando de la expresión a evaluar. Este proceso de búsqueda de operadores se repite hasta que la subexpresión haya sido totalmente evaluada, lo cual se produce al comprobar la ausencia de más operadores.

Para distinguir entre el operador resta y el signo negativo se tiene en cuenta que éste es el primer carácter de la subexpresión, por ejemplo –a+b, o sigue a un operador como solución a una subexpresión evaluada con anterioridad.

En la fase de simplificación de la ecuación las expresiones dependientes de β y del fan-out no se resuelven. Son memorizadas en variables de tipo *string* y sustituidas por cadenas de caracteres que los identifican: Idbta para β e Idfo para el fan-out. Al concluir la evaluación de la ecuación ambos identificadores son sustituidos por las expresiones memorizadas.

$$tphl = (\ldots)\cdot(\ldots)\cdot(1+ \ldots +0.56\cdot(beta-2)+1.5\cdot (FO-1)),$$

que tras su procesamiento queda:

$$\begin{aligned} tphl &= A\cdot B\cdot(1+ C + \ldots +0.56\cdot Idbta+1.5\cdot(FO-1))= \\ &= A\cdot B\cdot(1+ C + \ldots +0.56\cdot Idbta+1.5\cdot Idfo)= \\ &= A\cdot B\cdot(D+Idbta+Idfo)= A\cdot B\cdot(Idbta + Idfo)= \\ &= A\cdot B\cdot Idbta = E\cdot Idbta = Idbta \end{aligned}$$

donde Idbta identifica a una expresión dependiente de β y que corresponde a

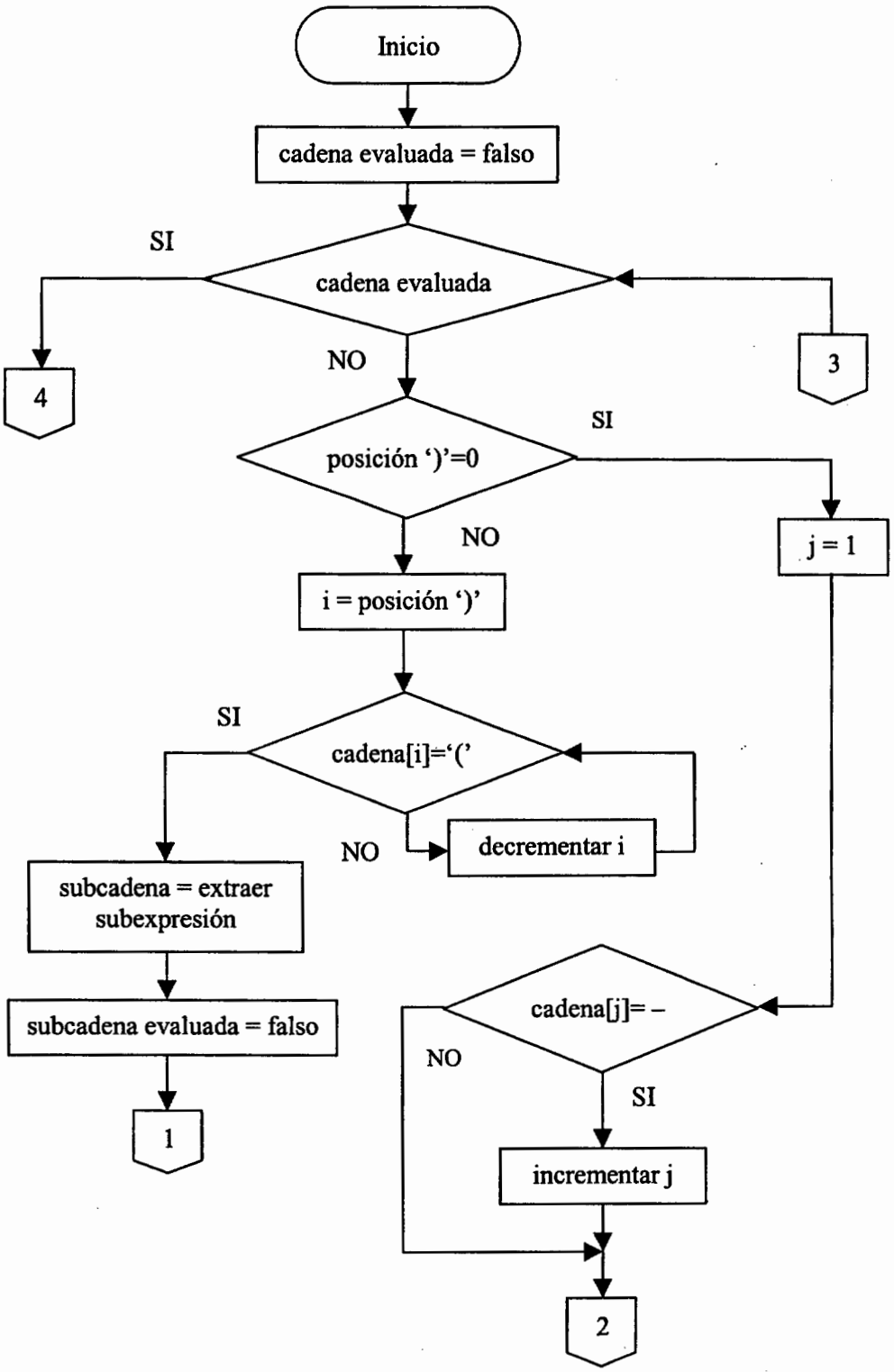
$$(E\cdot((D+(0.56\cdot(\beta-2)))+Idfo))$$

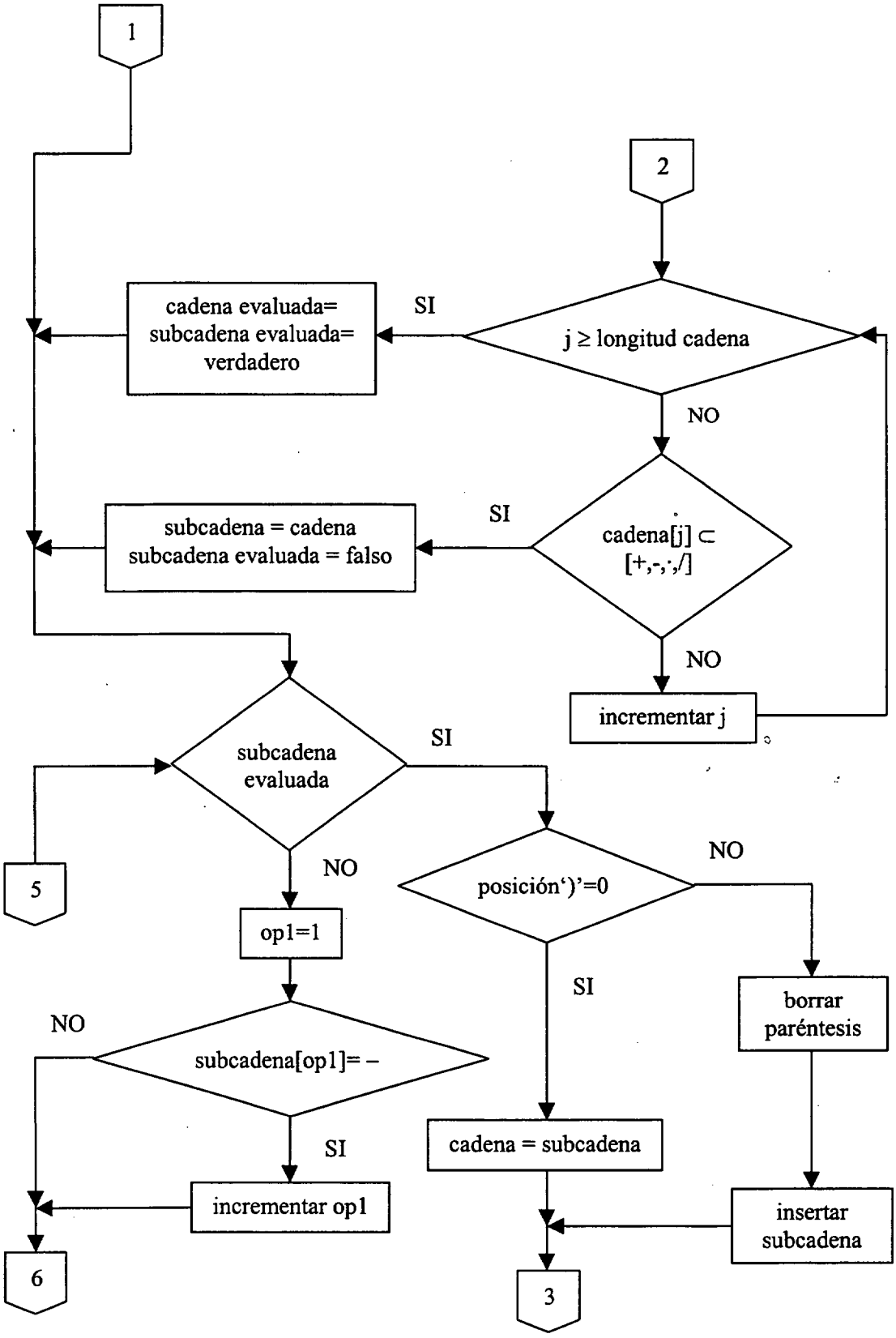
e Idfo a una expresión dependiente del fan-out

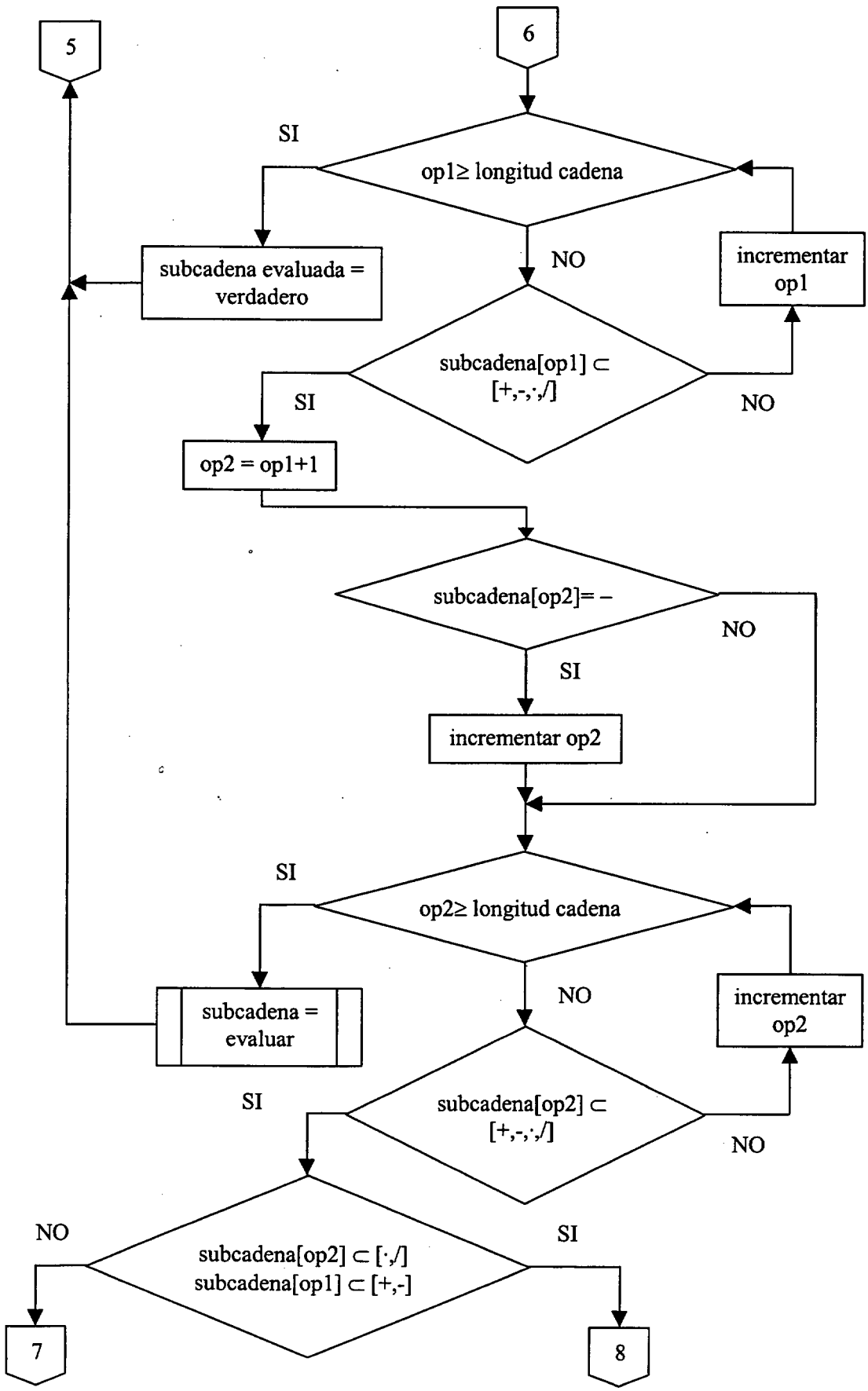
$$(1.5\cdot(FO-1))$$

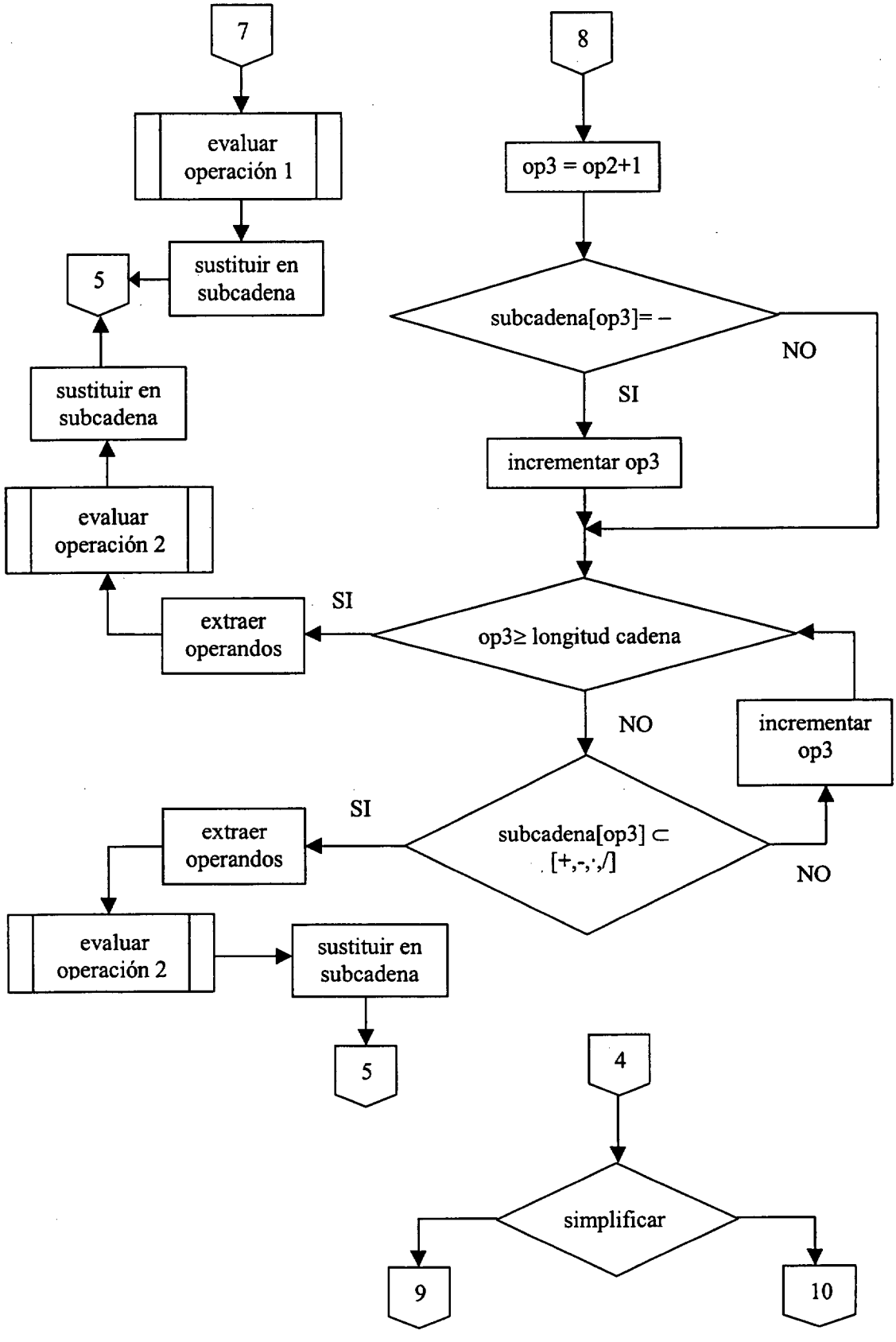
de tal forma que al sustituir ambos identificadores por sus correspondientes expresiones se obtiene una ecuación, producto de la reducción de la ecuación original, dependiente de los

parámetros β y fan-out. De esta forma la ecuación que se aplica para calcular los retardos requiere un menor tiempo de computación como se explica en el punto 3.2.1.









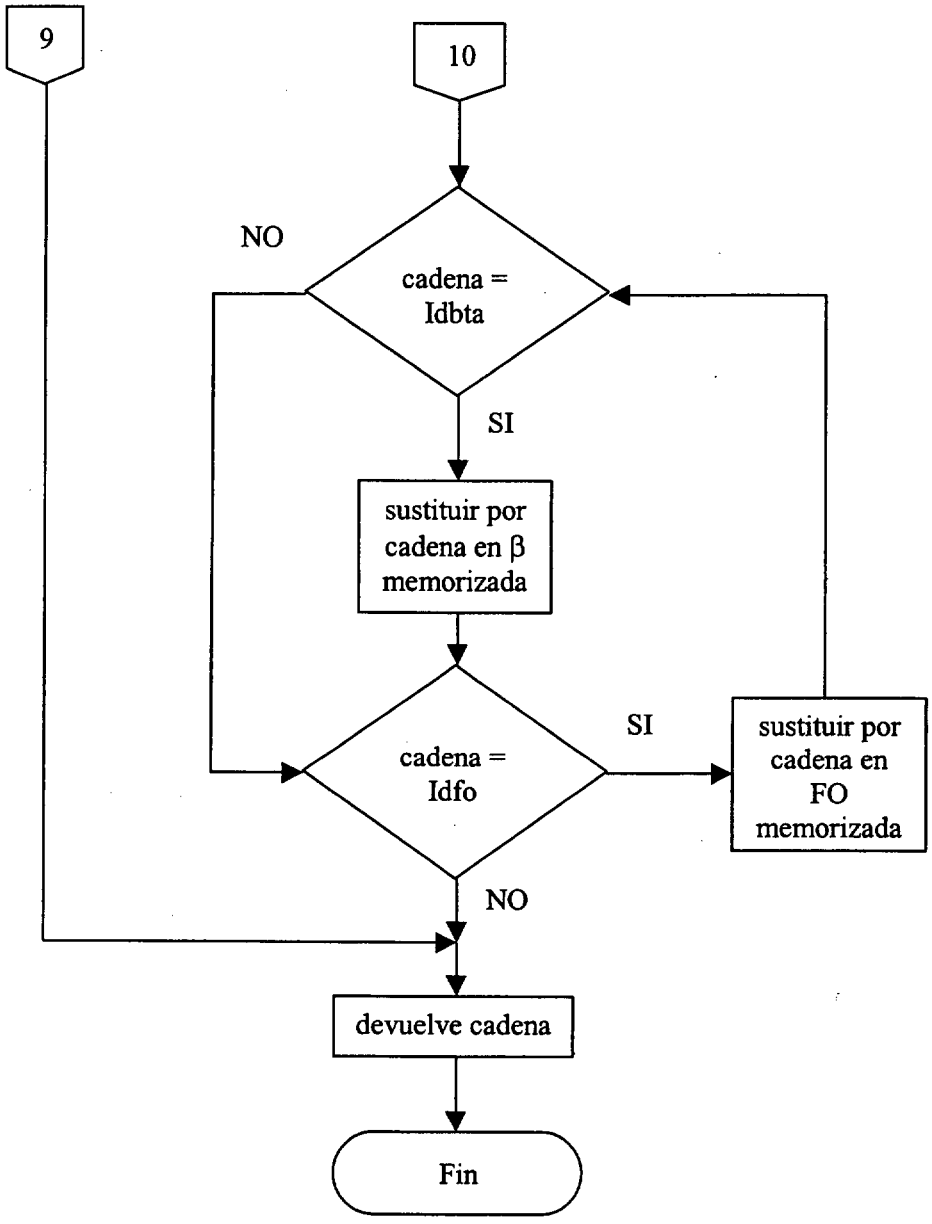


Figura 4. 22. Diagrama de flujos de la función TratarEcuacion.

4.2.1.4.22. Función check_nodos.

Esta función devuelve un valor lógico, de tipo *boolean*, verdadero si encuentra una puerta, elemento de la lista, que no ha sido evaluada. Para determinar si lo ha sido se comprueba el estado que presenta el campo pendiente del registro de tipo Puerta. Si el valor que presenta dicho campo es 0 se considera a la puerta como no evaluada.

Para verificar el estado de las distintas puertas del circuito se recorre la lista mediante un puntero para comprobar el valor del campo pendiente de cada uno de los elementos que la conforman. Si se encuentra un elemento que contenga el valor 0 se detiene el recorrido y la función devuelve el valor lógico verdadero.

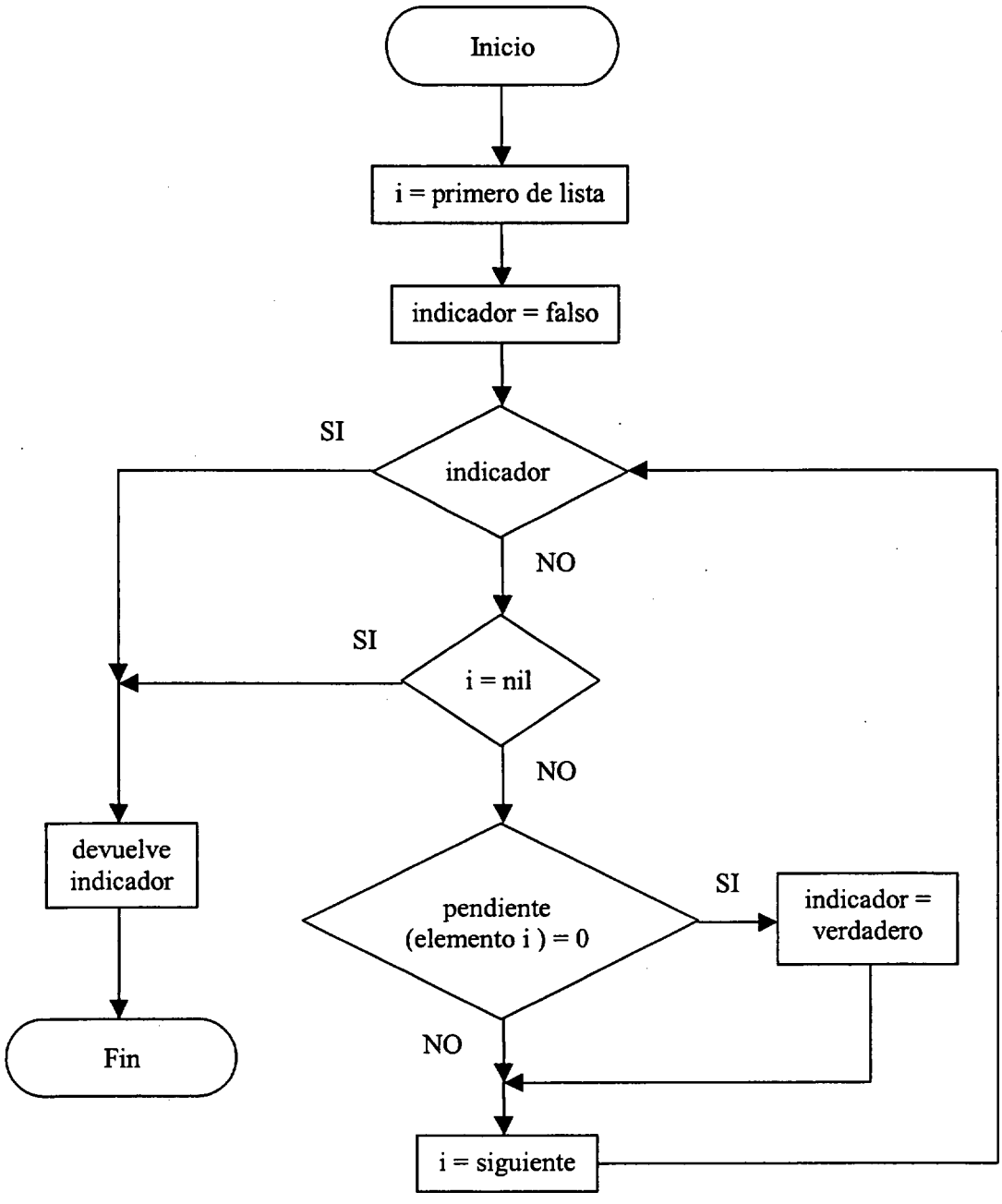


Figura 4. 23. Diagrama de flujos de la función check_nodos.

4.2.1.4.23. Procedimiento tipo_puerta.

Recibe un parámetro de tipo PunteroPuerta que apunta al elemento de la lista que representa a la puerta del circuito cuyo tiempo de propagación se procede a calcular. Determina el tipo de puerta de que se trata comprobando el contenido de los campos nodoe1 y nodoe2. La puerta es un inversor si el campo nodoe2 contiene el valor 0 y el campo nodoe1 un valor distinto de 0. Es una NOR2 si ambos valores son distintos de cero. Según el tipo de puerta realiza una llamada al procedimiento encargado de evaluar tal clase de puerta: procedimiento inversor o procedimiento nor2.

4.2.1.4.24. Función evaluar_hl1.

La función calcula y devuelve un valor de tipo real correspondiente al tiempo de propagación de la transición de nivel alto a bajo. Una variable de tipo lógica, recibida como parámetro por la función, se utiliza para determinar si se aplica la ecuación desarrollada por el modelo o una introducida por el usuario desde teclado. En el caso que se emplee la ecuación externa se comprueba que ésta no sea una cadena vacía, pues el usuario de la aplicación cuenta con la posibilidad de hacer uso de una única ecuación de introducción externa. De producirse tal circunstancia la función hará uso de la ecuación del modelo. En caso de aplicar la ecuación del modelo, el valor del fan-out de la puerta que se evalúa, si es mayor que uno o no, determina qué ecuación se utiliza de acuerdo con lo expresado en el punto 2.4.1.

La función recibe los valores de β y FO, campos beta y fanout del elemento de lista correspondiente, pertenecientes a la puerta cuyo tiempo de propagación se evalúa.

Una vez calculado el tiempo de propagación, se asigna el valor -1 al campo pendiente de la puerta evaluada. Dicho valor identifica la transición de nivel alto a bajo con que responde la puerta.

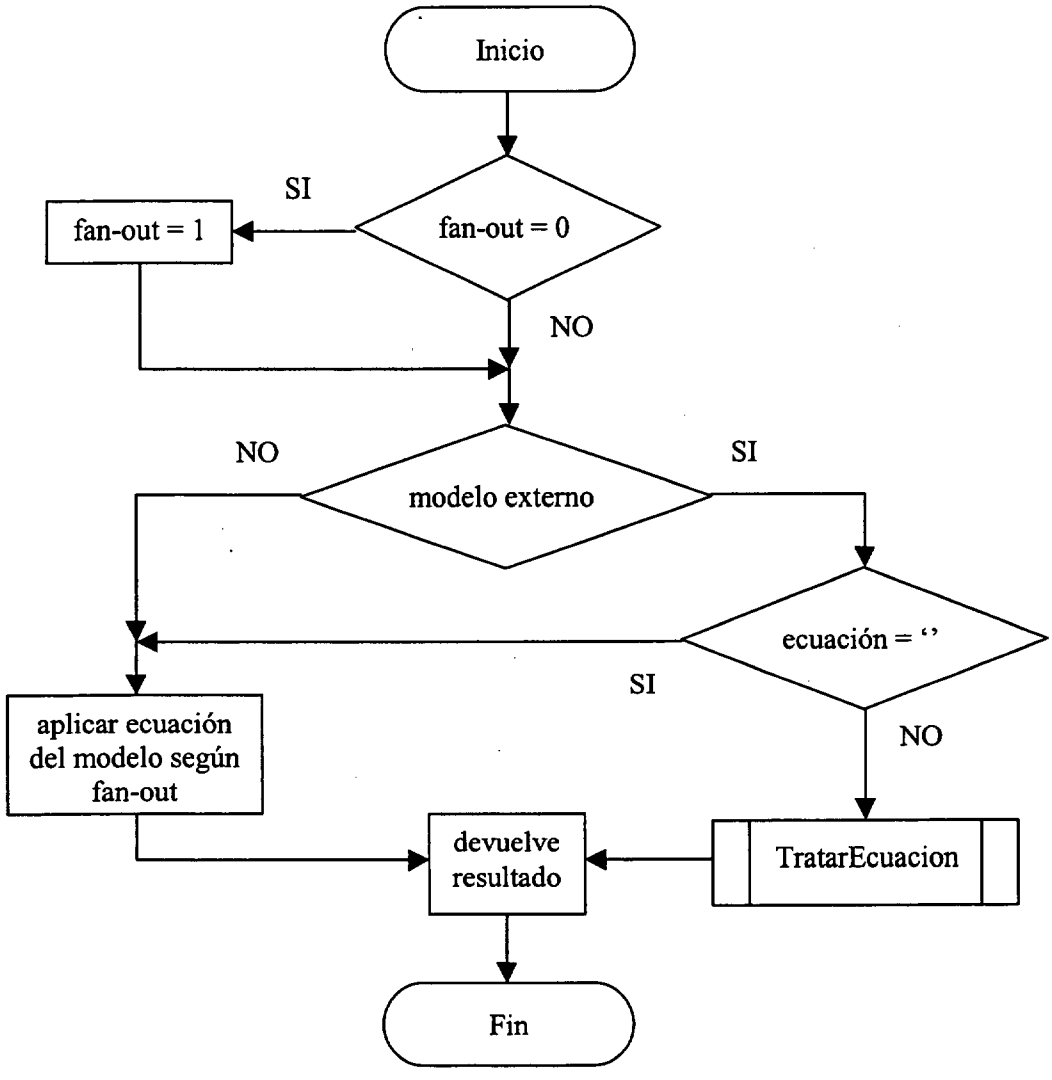


Figura 4. 24. Diagrama de flujos de la función evaluar_hl1.

4.2.1.4.25. Función evaluar_lh1.

Esta función, análoga a la anterior, calcula el tiempo de propagación de la transición de nivel bajo a alto devolviendo un valor de tipo real. La función recibe como parámetro una variable de tipo lógico mediante la que se determina si se aplica la expresión del modelo o la introducida por el usuario para evaluar el retardo. En el caso que se haga uso de la ecuación externa se comprueba que ésta no sea una cadena vacía, ya que el usuario de la aplicación cuenta con la posibilidad de introducir una única ecuación. Si se encuentra una cadena vacía la función hará uso de la ecuación del modelo. Si se emplea la

ecuación de modelado, se establece qué ecuación se aplica en función del fan-out tal como se expone en el apartado 2.4.1.

La función recibe como parámetros los valores que presentan los campos beta, relación de aspecto, y fanout, fan-out, del elemento de la lista que representa a la puerta que se evalúa.

Al campo pendiente del elemento evaluado se le asigna el valor 1 que representa la transición del 0 al 1 lógico a la salida de la puerta.

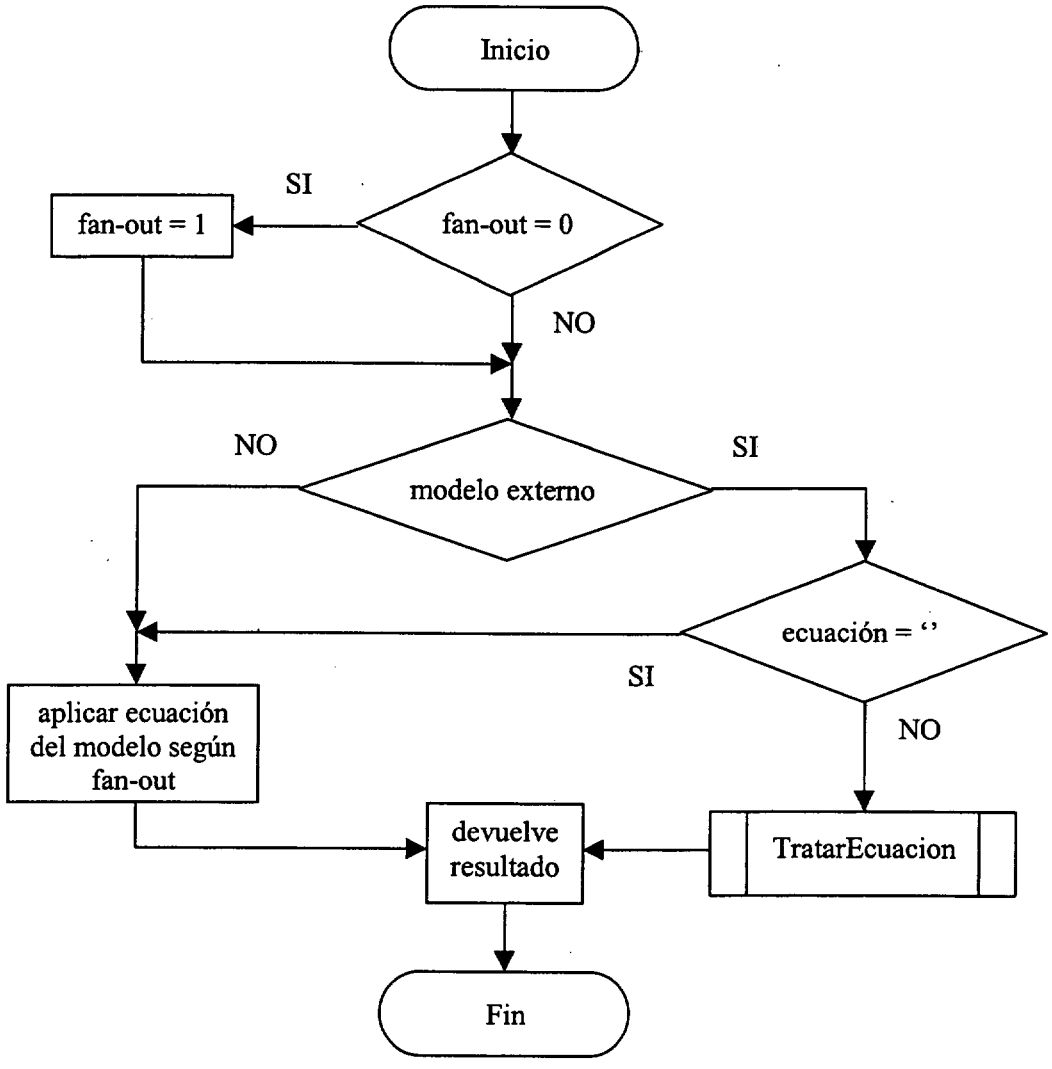


Figura 4. 25. Diagrama de flujos de la función evaluar_lh1.

4.2.1.4.26. Función beta_equivalente_LH.

La función devuelve un valor real correspondiente a la β equivalente para los casos de colisión de subida.

Recibe como parámetros la diferencia de los tiempos de propagación existente entre los nodos de entrada de la puerta NOR2 consignados en el campo tp de los elementos de la lista que corresponden a las entradas a la puerta NOR2, y el valor de la relación de aspecto propio de la puerta que se evalúa. Sobre los valores recibidos se aplica la ecuación de cálculo de la β equivalente desarrollada en el capítulo 2.

4.2.1.4.27. Función beta_equivalente_HL.

Esta función calcula el valor de la relación de aspecto equivalente para los casos de colisión de bajada. Devuelve dicho valor en formato real.

La diferencia de los retardos entre los nodos de entrada de la puerta NOR2 y el valor de la relación de aspecto de la puerta son las variables sobre las que se aplica la ecuación de cálculo de la β equivalente. Los tiempos de propagación en los nodos de entrada se encuentran consignados en los respectivos campos tp de los elementos de la lista que los representan. Estos valores se le pasan como parámetros a la función, así como el valor de la relación de aspecto de la puerta NOR2 en evaluación.

4.2.1.4.28. Procedimiento inversor.

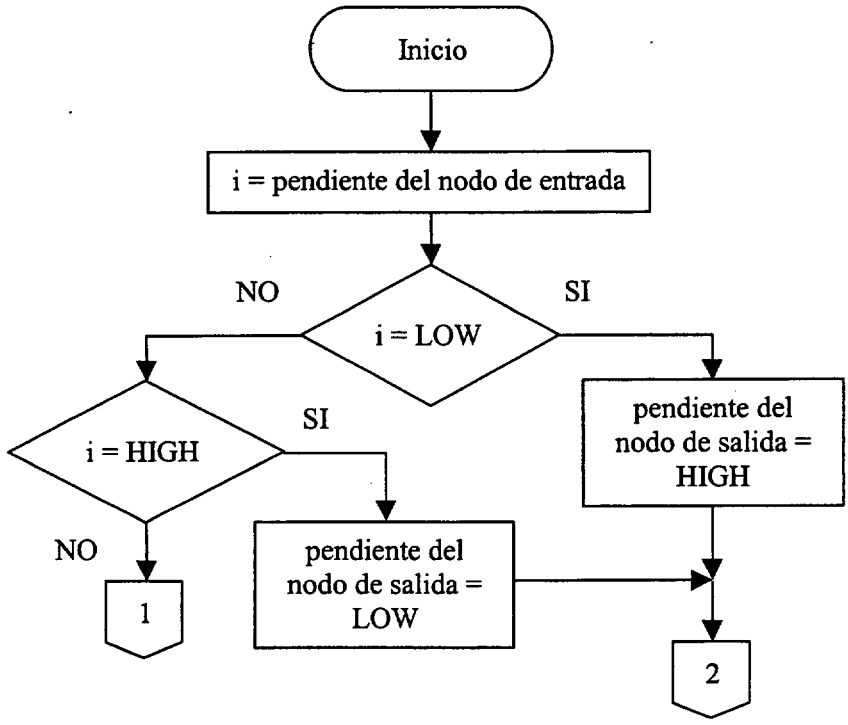
El procedimiento realiza la evaluación del tiempo de propagación correspondiente a una puerta de tipo inversora. Se le pasan punteros al elemento de la lista que se pretende evaluar y al elemento que corresponde al nodo de entrada del inversor, que es la salida de otra puerta o una entrada al circuito.

Se analiza el estado del campo pendiente del elemento equivalente al nodo de entrada al inversor. Si se mantiene a nivel bajo, es decir, campo pendiente = LOW, o alto,

HIGH, se asigna al campo pendiente del inversor el valor contrario, y debido a la inexistencia de transición no se efectúa cálculo alguno.

Si existe una transición, la pendiente correspondiente al nodo de entrada lo refleja conteniendo el valor 1, subida, o –1, bajada. Si el nodo de entrada transita del nivel alto al bajo el nodo de salida lo hace de nivel bajo a alto y se evalúa el tiempo de propagación mediante llamada a la función evaluar_lh1 para computar t_{pLH} del elemento de la lista que contiene al inversor, sumándose al valor así obtenido el retardo que presenta el nodo de entrada, recogido en el campo tp del elemento de lista que lo representa. Se llama a evaluar_hl1, cuando existe una transición bajo-alto en el nodo de entrada y alto-bajo en la salida, para obtener t_{pHL} del inversor, a lo que se suma tp del elemento representante del nodo de entrada. En ambos casos el resultado se asigna al campo tp del elemento de la lista que equivale a la puerta inversora.

Al campo respcambio del inversor se le asigna el identificador del nodo de salida perteneciente al elemento que es entrada del mismo, pues es el que origina la repuesta del inversor.



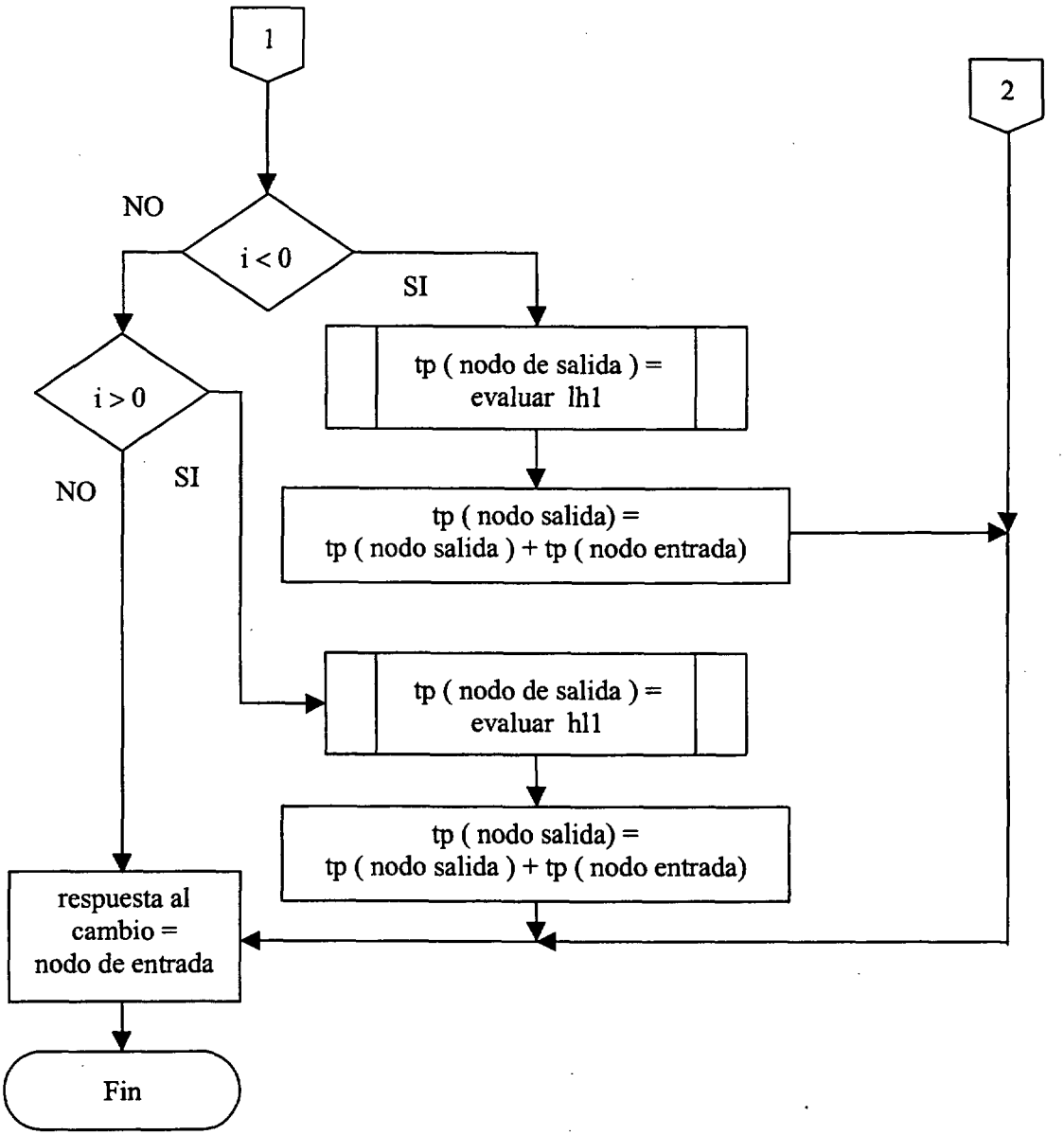


Figura 4. 26. Diagrama de flujos del procedimiento inversor.

4.2.1.4.29. Procedimiento pos_col_sub.

El procedimiento se utiliza en la evaluación de puertas de tipo NOR2. Se le llama para calcular el tiempo de propagación cuando ambos nodos de entrada presentan transiciones de nivel alto a bajo (la pendiente de los elementos de la lista que los representan tiene el valor −1). En la salida se produce una transición desde el nivel bajo al alto cuyo tiempo de propagación será asignado al campo tp del elemento que se evalúa.

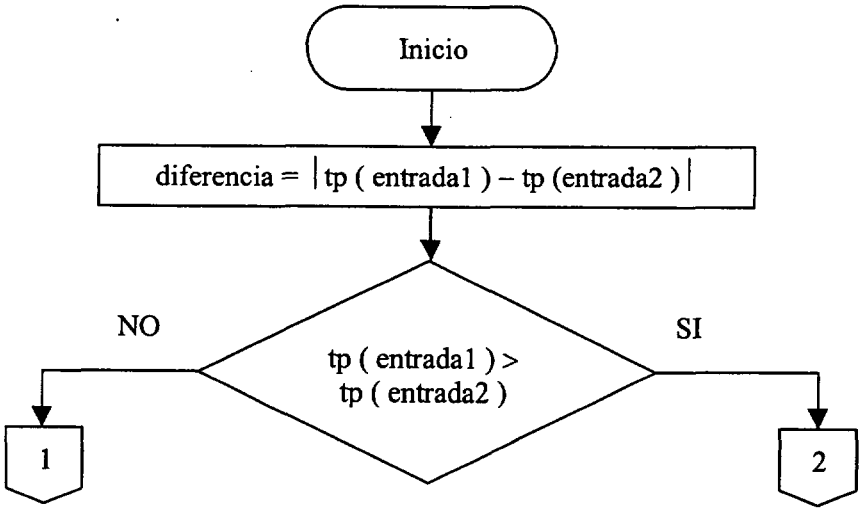
Recibe tres parámetros de tipo PunteroPuerta que apuntan al elemento correspondiente a la puerta NOR2 en proceso de evaluación y a los dos elementos de la lista que se conectan como las entradas de la puerta.

La puerta NOR2 en este caso responde a la última de las entradas en conmutar. Se asigna el identificador de dicho nodo de entrada como valor del campo respcambio del elemento de la lista que corresponde a la puerta en evaluación.

Se produce colisión de subida si la diferencia entre las transiciones de las entradas es menor o igual a 30 ps como ha sido descrito en el apartado 2.5.1.

El retardo se calcula mediante la correspondiente ecuación descrita en el punto 2.5.1 de la memoria. En los casos en que la separación entre las entradas sea mayor que 30 ps se utiliza el valor de la relación de aspecto equivalente $\beta = 3.5$ para todos los casos. Si se produce colisión el valor de β a usar se calcula mediante llamada a la función beta_equivalente_LH, como función de la relación de aspecto propia de la puerta y la diferencia entre los retardos de las entradas

En ambos casos se ha de sumar a dicho resultado el tiempo de propagación, campo tp, del nodo de entrada al que responde la puerta.



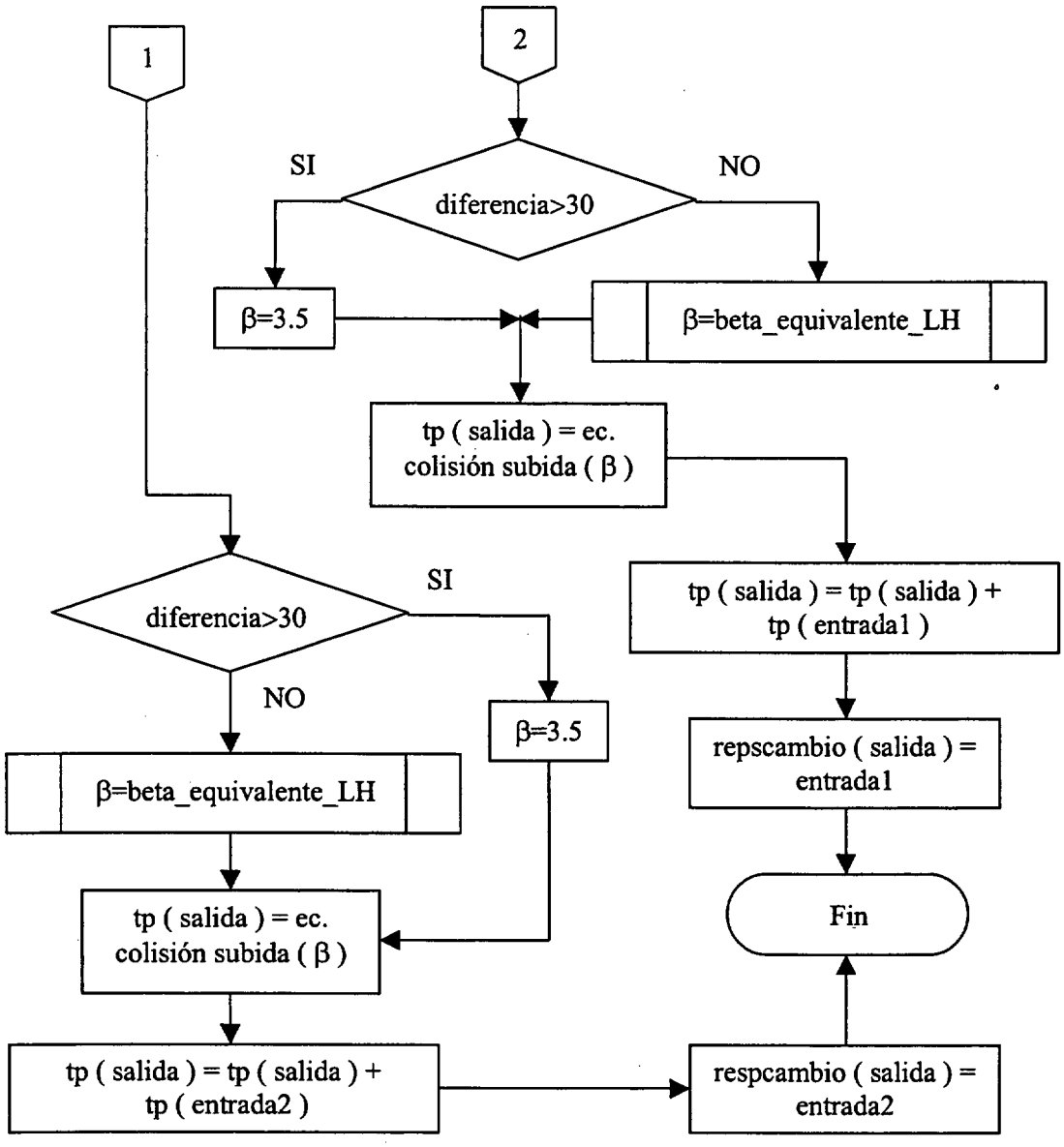


Figura 4. 27. Diagrama de flujos del procedimiento pos_col_sub.

4.2.1.4.30. Procedimiento pos_col_baj.

Este procedimiento se utiliza en la evaluación de puertas NOR2. Efectúa el cálculo del tiempo de propagación cuando ambas entradas a la puerta presentan transiciones del nivel bajo al alto, identificadas por el valor 1 contenido en el campo pendiente de los elementos de la lista enlazada que las representan. La salida de la puerta presenta una transición de nivel alto a bajo. El tiempo de propagación que se obtiene se consigna en el campo tp de la puerta en evaluación.

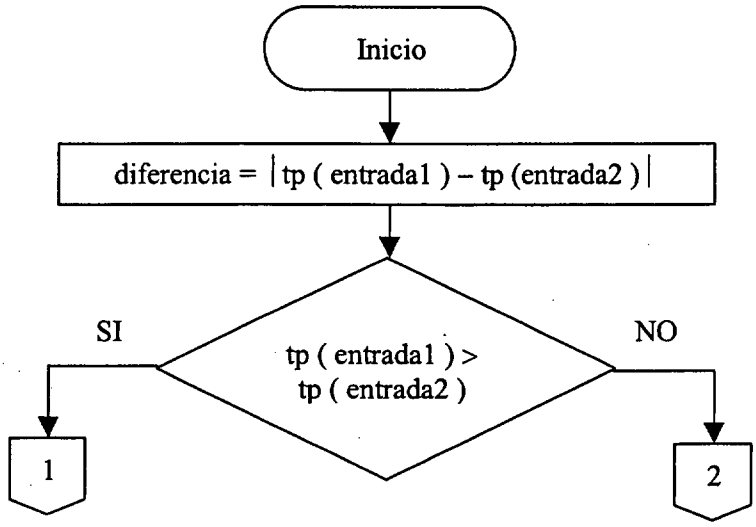
Al procedimiento se le pasan tres punteros de tipo PunteroPuerta que referencian al elemento equivalente a la puerta NOR2 que se evalúa y a los dos elementos de la lista que corresponden a las entradas de la puerta.

La puerta NOR2 responde a aquella de las entradas que cambie primero. El identificador del nodo de entrada de menor retardo se asigna al campo respcambio del elemento de la lista que corresponde a la puerta en evaluación.

Se produce colisión de bajada si la separación existente entre ambas entradas resulta igual o inferior a 30 ps como se explica en el punto 2.5.1.

Si la diferencia supera la citada cantidad el cómputo del tiempo de propagación se realiza empleando la ecuación para los casos de colisión de bajada, aplicándose el valor de la relación de aspecto equivalente $\beta = 2.3$ en todos los casos. Si hay colisión el valor de β a utilizar se obtiene mediante la función beta_equivalente_HL, como función de la relación de aspecto de la puerta y la diferencia entre los retardos de las entradas.

En ambos casos se suma al resultado el retardo, campo tp, del nodo de entrada al que responde la puerta.



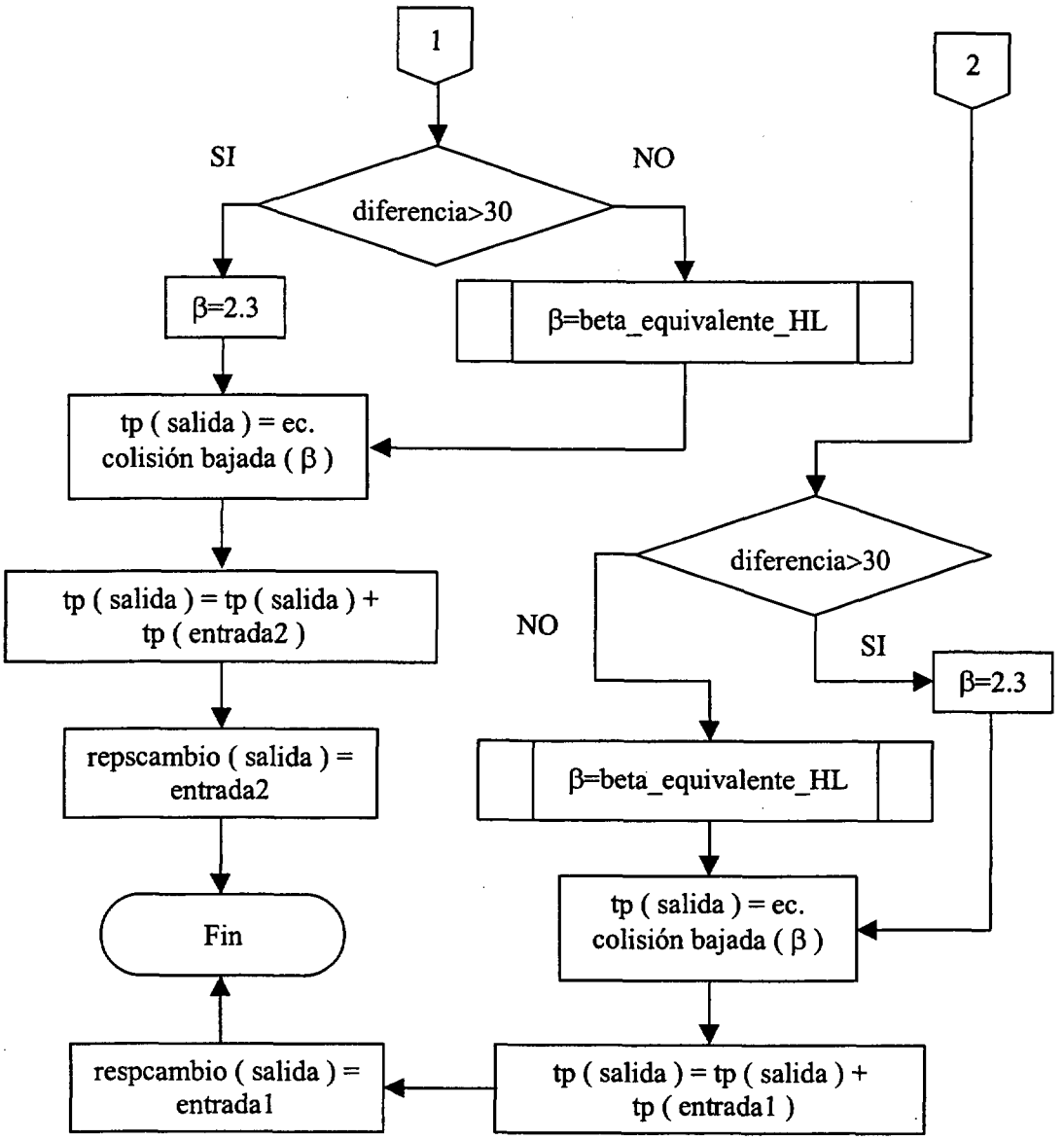


Figura 4. 28. Diagrama de flujos del procedimiento pos_col_baj.

4.2.1.4.31. Procedimiento nor2.

El procedimiento evalúa el tiempo de propagación perteneciente a una puerta de tipo NOR2.

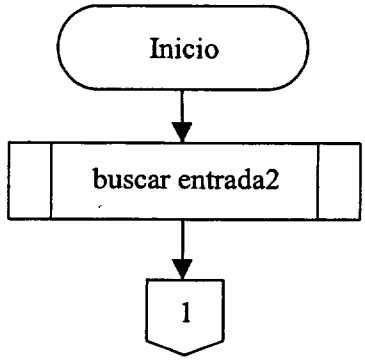
Recibe punteros a los elementos de la lista correspondientes a la puerta NOR2 que se evalúa y a una de sus entradas, cuyo estado definido por su pendiente y retardo es conocido.

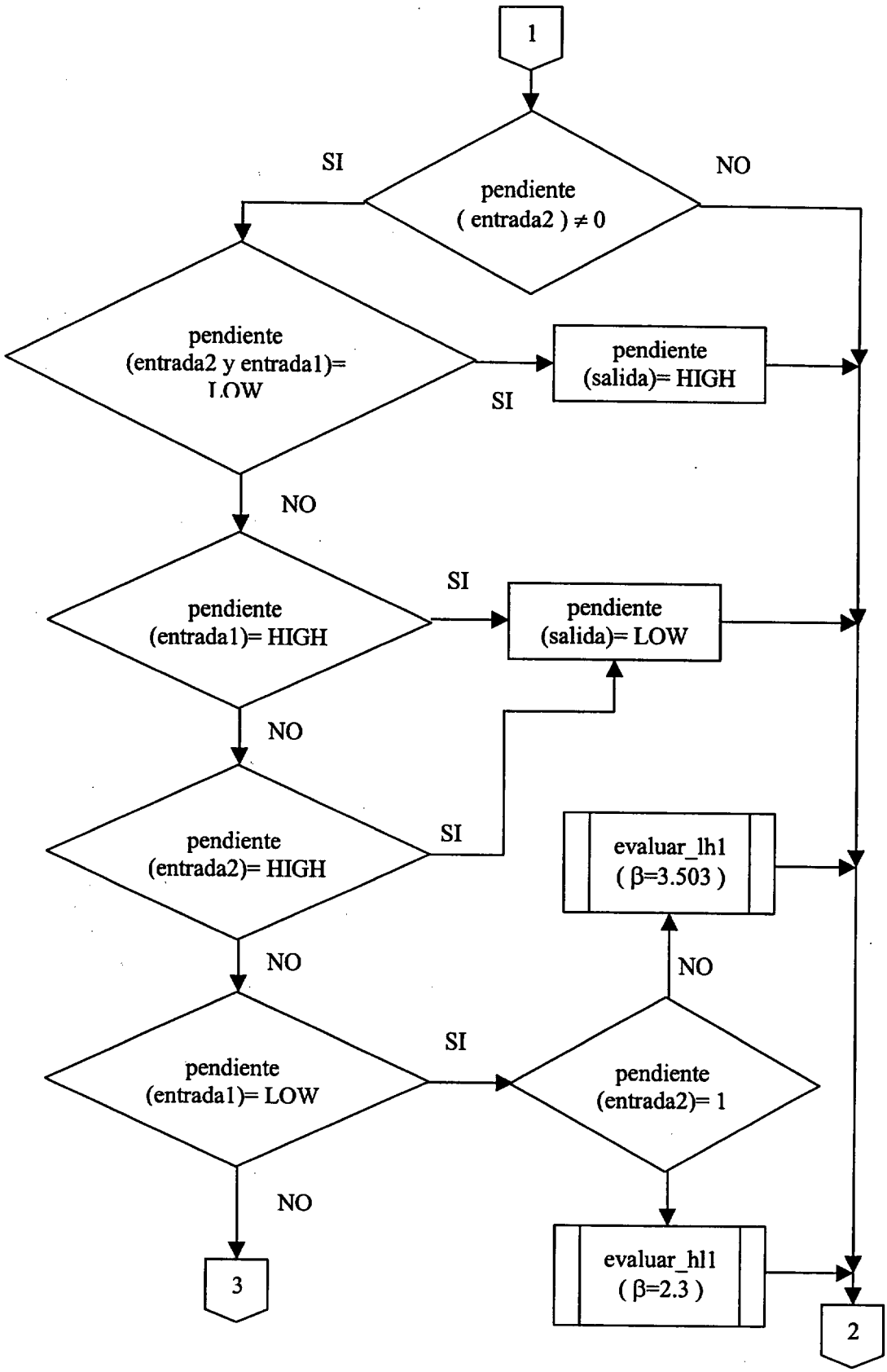
Se hace uso del procedimiento Buscar para obtener un puntero al elemento que representa a la segunda entrada de la puerta. Si ésta presenta valores de pendiente y retardo, campo tp, distintos de cero, se procede a evaluar la puerta NOR2.

Si ambas entradas se mantienen a nivel bajo, sus respectivos campos pendiente contienen el valor LOW, la salida permanece a nivel alto, asignándose el valor HIGH al campo pendiente del elemento de la lista sujeto de la evaluación. Si una de las entradas está a nivel alto la salida permanece a nivel bajo y se le asigna el valor LOW.

Cuando una de las entradas presenta una transición, su campo pendiente vale 1 o −1, y la otra entrada se mantiene a nivel bajo, el tiempo de propagación que presenta la puerta NOR se evalúa por aproximación de la puerta a un inversor. A la puerta NOR se le asigna como valor de la relación de aspecto para el cálculo 3.503, si su salida transita de bajo a alto, o 2.3 para transiciones de alto a bajo, atendiendo a lo expuesto en el punto 2.4 de la memoria. Para calcular el retardo se llama a la función evaluar_lh1 o evaluar_hl1 según el tipo de transición.

En el caso de que ambas entradas presenten transición en el mismo sentido puede existir colisión. La evaluación del timpo de propagación se realiza mediante llamada al procedimiento pos_col_sub si las entradas transitan del nivel alto al bajo, la salida de bajo a alto. Análogamente se llama a pos_col_baj si las entradas transitan del nivel bajo al alto, pasando la salida de alto a bajo. Si los sentidos de las transiciones son distintos, pendientes de distinto signo, la salida de la puerta permanece a nivel bajo. Este caso es el último en comprobarse y se detecta por el valor 0 que aún mantiene la pendiente de salida de la puerta en evaluación.





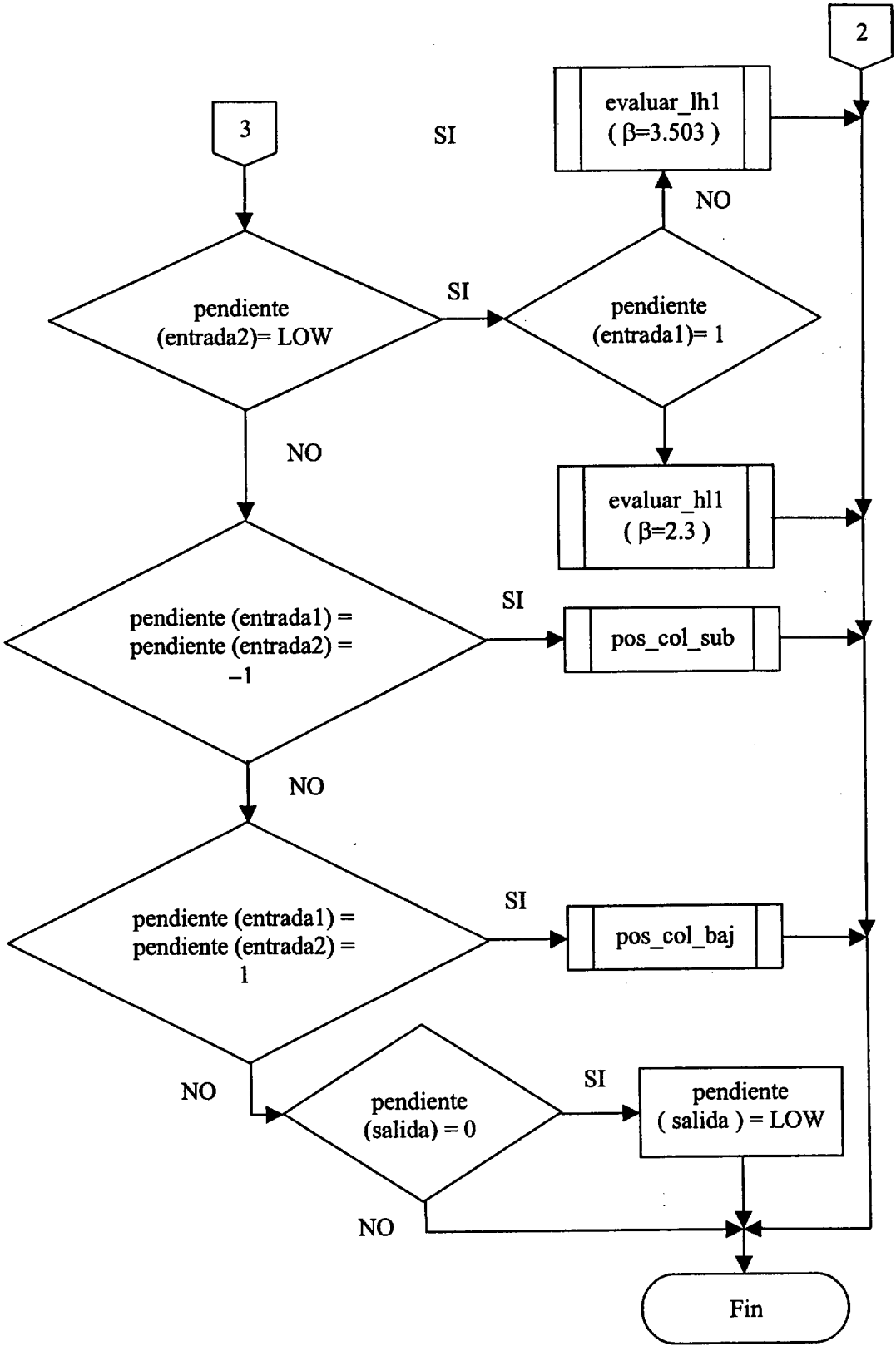


Figura 4. 29. Diagrama de flujos del procedimiento nor2.

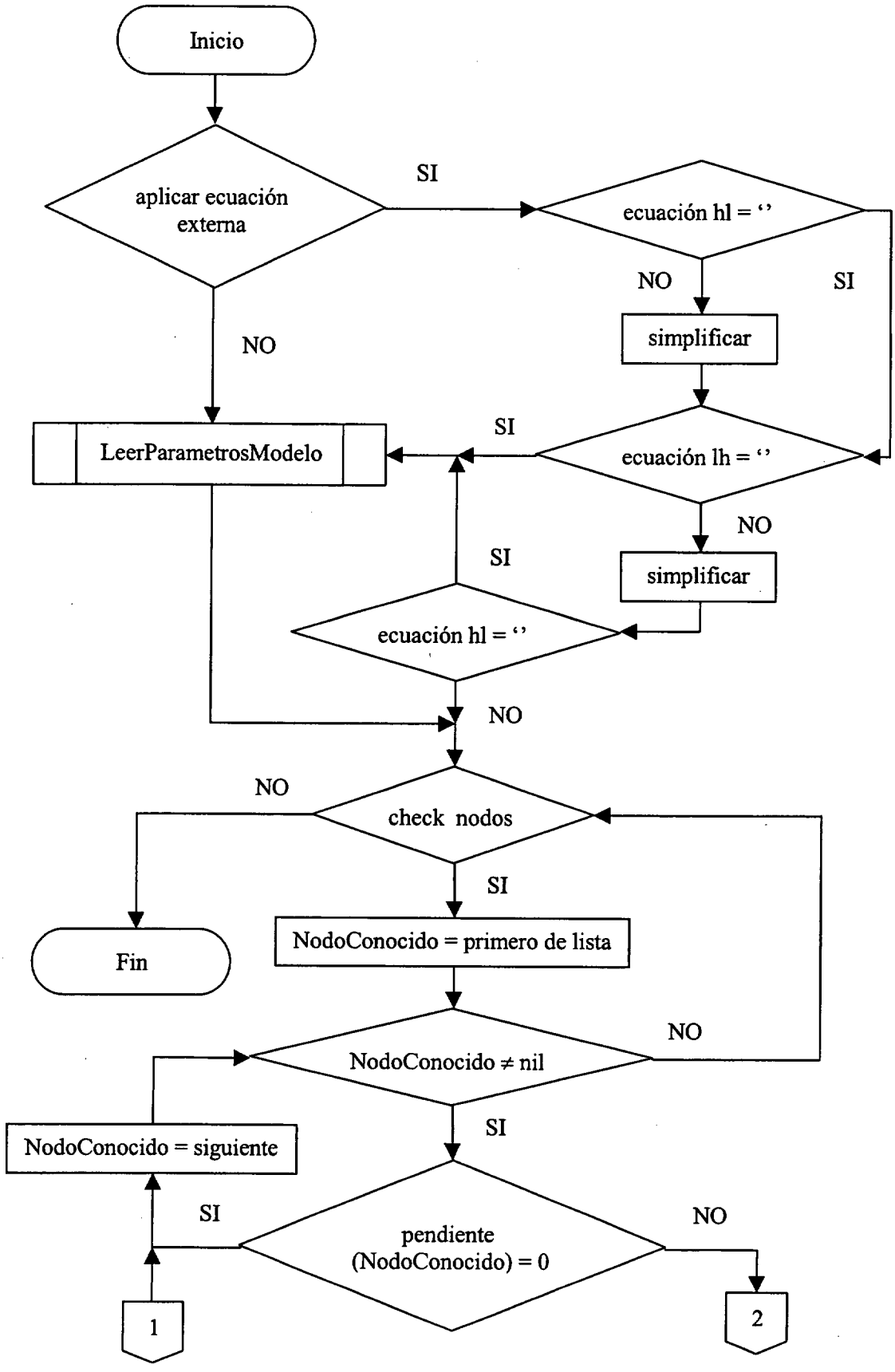
4.2.1.4.32. Procedimiento CalcularAtributos.

El procedimiento realiza la simulación del circuito representado por el *netlist*. Recibe como parámetros los punteros que soportan la lista que apuntan al comienzo y final de lista. Las posibles ecuaciones externas que el usuario de la aplicación haya propuesto también se reciben como parámetros.

Se determina qué ecuaciones van a ser usadas como modelo temporal; en el caso que sean las externas, éstas se simplifican mediante llamadas a la función TratarEcuacion.

La simulación concluye cuando el estado de todos los nodos que componen el circuito esté definido, es decir, se conozca la pendiente que presentan. La comprobación de este hecho se realiza mediante llamada a la función check_nodos, cuya respuesta de tipo lógico se usa como control del bucle de simulación.

El método de resolución del circuito consiste en la búsqueda de forma secuencial a lo largo de la lista de aquellos elementos definidos mediante un puntero que se ha llamado NodoConocido. Inicialmente sólo las entradas al circuito se encuentran definidas. Una vez hallado uno un segundo puntero, NodoDesconocido, realiza un nuevo recorrido completo a través de la lista. Se comprueba si el identificador del nodo de salida del elemento definido corresponde al nodo de entrada, o a uno de ellos en puertas NOR2, del elemento apuntado por NodoDesconocido. Si se produce tal circunstancia se verifica si el elemento apuntado por NodoDesconocido ya ha sido evaluado comprobando el valor que presenta su campo pendiente. Si el valor es cero el elemento apuntado por NodoDesconocido no ha sido evaluado y se llama al procedimiento tipo_puerta para su evaluación. En cualquier caso, se proseguirá con el recorrido a lo largo de la lista buscando nuevos elementos conectados a NodoConocido hasta llegar al final de la misma. Ambos punteros han de recorrer la lista en su totalidad. Este proceso se repite hasta que todos los elementos de la lista estén definidos.



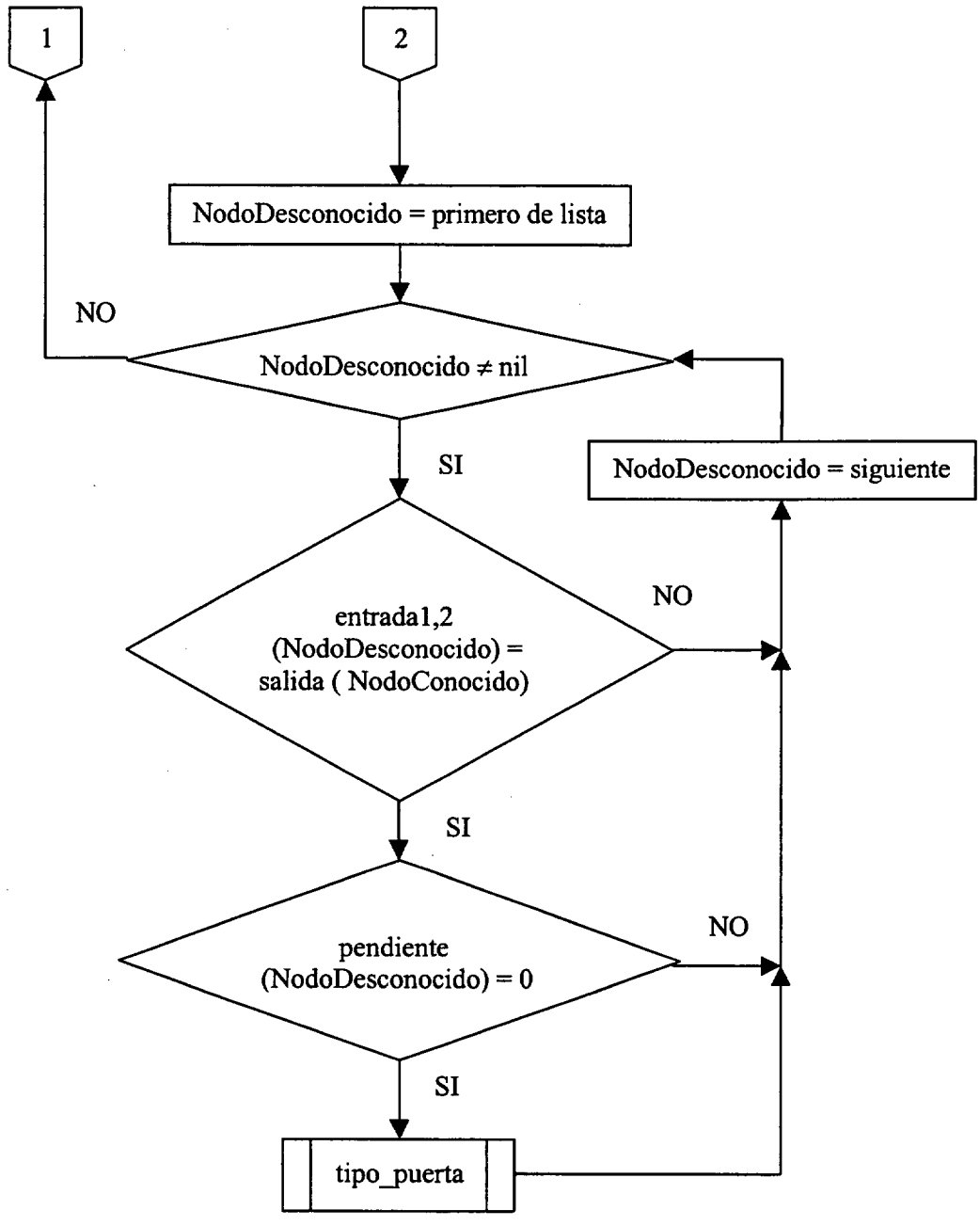


Figura 4. 30. Diagrama de flujos del procedimiento CalcularAtributos.

4.2.1.5. Unidad res4.

La unidad res4 contiene el código asociado a la ficha Resultado. Dicha ficha tiene carácter de cuadro de diálogo, y se utiliza como interfaz para el acceso a los resultados de la simulación por parte del usuario de la aplicación. La ficha se llama en modo modal. Sólo está habilitada su llamada si previamente ha concluido con éxito una simulación.

La ficha contiene etiquetas, componentes de tipo *Label*, y un componente de edición de tipo *Memo* especificado exclusivamente para lectura cuya función es mostrar los datos solicitados. En la etiqueta se muestra el tiempo de propagación, o el nivel bajo o alto si no existe transición, que presenta el nodo del circuito escogido por el usuario de la aplicación. El componente Memo está dotado de barras de desplazamiento para moverse a través del texto y presenta la información relativa al camino lógico que ha dado lugar a la transición en el nodo seleccionado.

El usuario de la aplicación cuenta con la posibilidad de escoger un nodo del *netlist* para obtener la información del retardo calculado en el mismo, o acceder directamente a la información de nodo crítico, que es aquél que presenta mayor tiempo de propagación de entre los que constituyen el *netlist*. Para seleccionar entre ambas opciones se ha dispuesto un cuadro de grupo con botones de activación, que corresponde a un componente de tipo *RadioGroup*. Se establece en tiempo de diseño como opción por defecto el nodo crítico.

Se ha añadido un cuadro de lista, componente de tipo *ListBox*, para contener el listado de los nodos del circuito que el usuario necesita seleccionar cuando quiera consultar los resultados de la simulación. Este componente sólo se encuentra habilitado para su uso cuando el usuario ha escogido la opción de seleccionar nodo. Contiene todos los nodos del *netlist* excepto aquellos que figuran como entradas del circuito.

Se ha situado en la ficha una casilla de activación, componente de tipo *CheckBox*. Este elemento permite al usuario configurar el origen de tiempos respecto al que se presentan los resultados obtenidos en el proceso de simulación. A través de este componente puede situar el origen en 0 o en el nodo inicial del camino correspondiente al punto del circuito que se consulta. La propiedad *Checked* de la casilla de activación adquiere el valor lógico verdadero cuando el usuario activa la opción de situar el origen en el primer nodo del camino, y el valor falso cuando la desestima.

Se emplea un componente *TChart*, llamado Chart1, para realizar una representación gráfica de los tiempos de propagación de los nodos correspondientes al camino relativo al nodo seleccionado por el usuario. Este componente proporciona la estructura básica para crear gráficos de datos siendo su contenedor visual. Se trata de un componente complejo,

que mediante sus diversas propiedades permite establecer varias configuraciones de representación gráfica.

La propiedad *View3D* se fija al valor falso, con lo que la representación tiene carácter bidimensional. Las propiedades *AllowZoom* y *AnimatedZoom* se emplean para definir la utilización de efectos de zoom sobre el gráfico. A la propiedad *AllowZoom* se le asigna el valor lógico verdadero para permitir el empleo del zoom. *AnimatedZoom* fija si el zoom es instantáneo o se realiza aplicando sucesivos zooms, habiéndose escogido esta segunda opción al asignar el valor verdadero a la propiedad. El número de zooms consecutivos se define mediante la propiedad *AnimatedZoomSteps*, que por defecto contiene el valor 8.

Un componente *TChart* cuenta con cuatro subcomponentes de la clase *TChartAxis*, que corresponden a los cuatro posibles ejes de la representación gráfica: izquierdo, derecho, superior e inferior. A través de la propiedad *Visible* de cada uno se establecen los ejes del gráfico. Se ha designado el eje *BottomAxis* (inferior) como eje x y el eje *LeftAxis* (izquierdo) como eje y. En ambos ejes se fija la propiedad *Automatic*, de tipo lógico, al valor verdadero. De esta forma la escala de valores de los mismos se establece en función de los valores x e y mínimos y máximos que contiene la serie a representar.

El gráfico de datos propiamente dicho es un objeto de la clase *TLineSeries*, derivado de *TChartSeries*, llamado Series1. Corresponde a la serie de datos a mostrar. La clase *TLineSeries* define un tipo de representación en la que los puntos de la misma se muestran unidos por una línea. Este objeto se asocia al componente de gráficos de datos, Chart1, en tiempo de diseño mediante el Editor de componentes de gráficos de datos del Chart. Sus propiedades *HorizAxis* y *VertAxis* determinan los ejes sobre los que se escala la serie, y se asocian a los ejes *BottomAxis* y *LeftAxis* del componente Chart1.

Una vez que el usuario escoge un nodo del circuito, los nodos y sus respectivos tiempos de propagación, correspondientes al camino relativo a ese nodo, se asignan como pares x-y de la serie para su representación gráfica.

Se ha situado en la ficha un botón para ordenar la presentación de los datos, y un segundo botón para cerrarla.

Se hace uso de las declaraciones de tipos de datos y subprogramas de la unidad sim4, para lo que se la incluye en la cláusula **uses** de la sección **interface** de la unidad. La cláusula uses permite a una unidad utilizar las declaraciones de tipos y procedimientos realizadas en otra unidad. La unidad sim4 se referencia en la sección interface de res4 pues las declaraciones de aquella son necesarias para las declaraciones de la parte interface de res4.

Se ha definido un tipo de datos para almacenar la información concerniente al camino que da lugar al tiempo de propagación que presenta un nodo determinado. El tamaño del *netlist* se desconoce a priori y es la causa por la que se ha usado una estructura de datos dinámica en forma de lista enlazada. Cada elemento de la estructura es de tipo registro, y consta de dos campos de información, uno de tipo entero y otro de tipo real, y de un campo puntero que enlaza con el siguiente elemento de la lista. La información que almacena el campo nod de tipo entero corresponde al identificador numérico de uno de los nodos que conforman el camino, y el campo retardo de tipo real contiene el tiempo de propagación de dicho nodo. Ambos valores se obtienen del elemento equivalente de tipo Puerta, declarado en sim4, que conforma la lista enlazada que representa al *netlist*.

```
pbuffer = ^bfer;
bfer = record
    nod : integer;
    retardo : real;
    enlace : pbuffer;
end;
```

Se han añadido cuatro campos y nueve métodos a la clase que describe la ficha. Dos de los campos son de tipo PunteroPuerta, declarado en la unidad sim4, y se usan para apuntar a los elementos inicial, PrinclistaN, y final, FinlistaN, de la lista doblemente enlazada que representa al *netlist*. Ambos punteros se declaran en la sección **interface** como **public** para acceder a ellos desde la ventana principal por el método que muestra a la ficha Resultado, el cual los inicializa al comienzo y final de la lista. Los otros dos campos se han declarado bajo la cláusula **private** y corresponden a un puntero de tipo pbuffer para sostener la lista que representa al camino, buffer, y un puntero, NodoEscogido, de tipo PunteroPuerta, que apuntará al elemento sobre el que el usuario de la aplicación desea

obtener información. Los métodos agregados se emplean en el método MosBtnClick para generar y presentar la información solicitada por el usuario.

Se describen a continuación los métodos contenidos en la unidad.

4.2.1.5.1. Método SeleccionClick.

Es el método gestor del evento OnClick del grupo de activación *Seleccion*. Según la casilla escogida habilita, si se selecciona Escoger Nodo, o inhabilita, eligiendo la casilla Nodo Crítico, el componente de cuadro de lista ListaNodos. Para determinar qué casilla se ha escogido consulta el valor de la propiedad *ItemIndex* del grupo de activación. La habilitación/inhabilitación del cuadro de lista se realiza asignando el correspondiente valor lógico a la propiedad *Enabled* del mismo. En caso de habilitación se fija el primer nodo de la lista como selección por defecto y se permite al usuario seleccionar a través del cuadro de lista el nodo del circuito cuyo retardo desee conocer. En el otro caso, se mostrará la información relativa al nodo crítico, que es el que presenta mayor retardo de los que componen el circuito.

4.2.1.5.2. Método CerBtnClick.

Método gestor del evento Onclick del botón CerBtn. Cierra la ficha, que al crearse de forma automática por el archivo del proyecto permanece oculta.

4.2.1.5.3. Método EncontrarNodoCritico.

El método EncontrarNodoCritico se ha añadido a la clase que describe la ficha. Busca en la lista doblemente enlazada, que contiene la información de las puertas que componen el circuito, el nodo que presenta el mayor retardo. Mediante un puntero auxiliar de tipo PunteroPuerta recorre la lista en su totalidad asignando la dirección del elemento de mayor retardo al puntero NodoEscogido. Si las entradas aplicadas al circuito son niveles

lógicos fijos, ninguna puerta presenta transición en su salida, con lo que se asigna el valor nil al puntero.

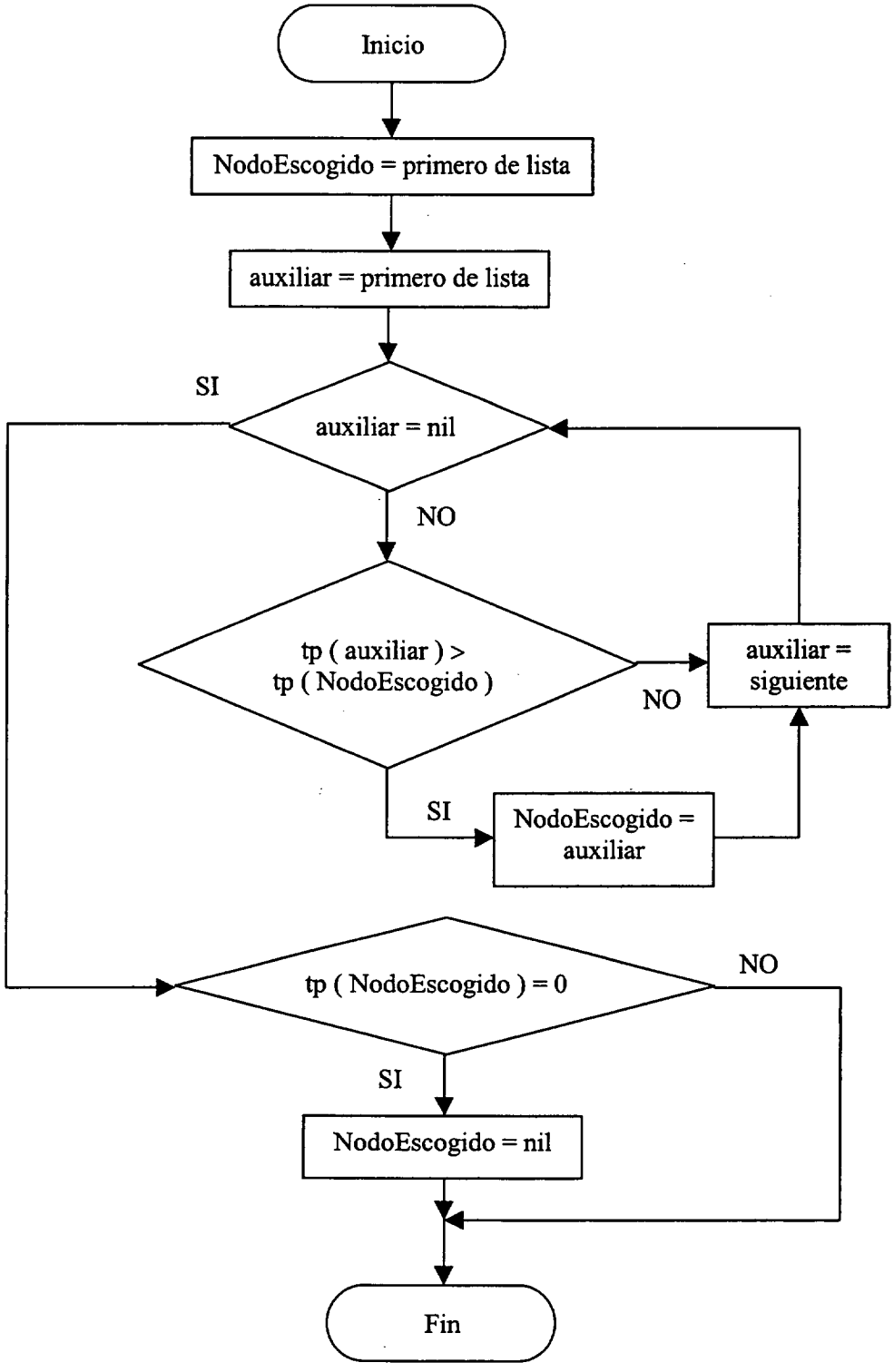


Figura 4. 31. Diagrama de flujos del método EncontrarNodoCritico.

4.2.1.5.4. Método Inbuffer.

Método añadido a la clase. Inserta un nuevo elemento en la lista enlazada apuntada por buffer. Dicho puntero apunta siempre al primer elemento de la lista. Los elementos nuevos se insertan al comienzo de la lista.

El elemento a insertar se crea por el procedimiento estándar *new* mediante el uso de un puntero auxiliar. La información de tipo entero correspondiente al identificador de nodo y de tipo real, tiempo de propagación, se recibe como parámetro por el método y se asigna al nuevo elemento.

Si la lista está vacía se hace apuntar el puntero buffer a dicho elemento y el campo enlace del elemento a nil como indicador de final de lista.

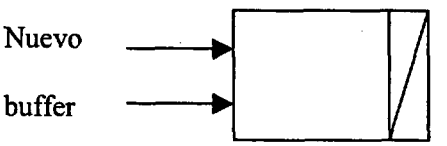


Figura 4. 32. Inserción e lista vacía.

Si la lista contiene más elementos se hace apuntar al campo enlace del nuevo elemento al elemento apuntado por buffer, primer elemento de la lista, y al puntero buffer se hace apuntar al nuevo elemento.

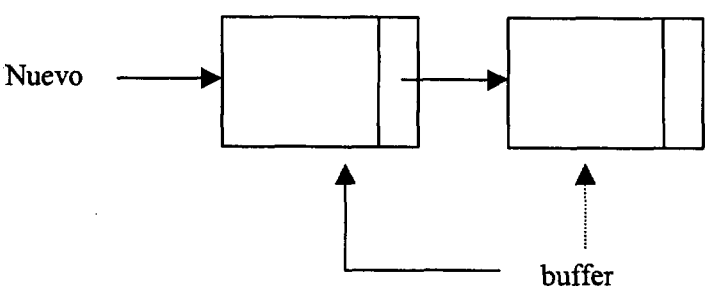


Figura 4. 33. Inserción por la cabeza de la lista.

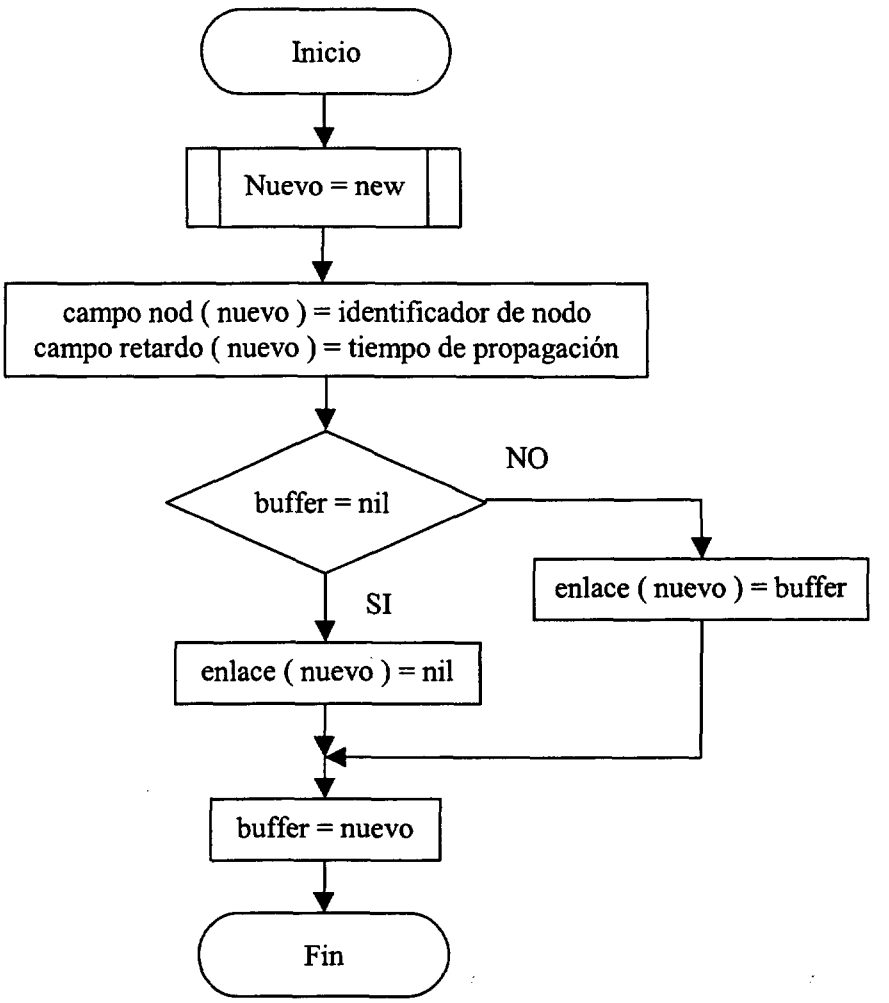


Figura 4. 34. Diagrama de flujos del método Inbuffer.

4.2.1.5.5. Método Borrarbuffer.

Es un método añadido a la clase que borra la lista apuntada por el puntero buffer. Los elementos que la forman son eliminados por la cabeza de la lista, es decir, borrando el elemento apuntado por buffer. La eliminación se realiza empleando el procedimiento estándar *dispose*. Se utiliza un puntero auxiliar que apunta al elemento que se pretende borrar. Antes de eliminar el elemento se asigna al puntero buffer la dirección del siguiente elemento en la lista, que pasa a ser la nueva cabeza de lista, con objeto de no perder el enlace que sostiene la lista. Se considera la lista borrada cuando el puntero buffer contenga la constante nil.

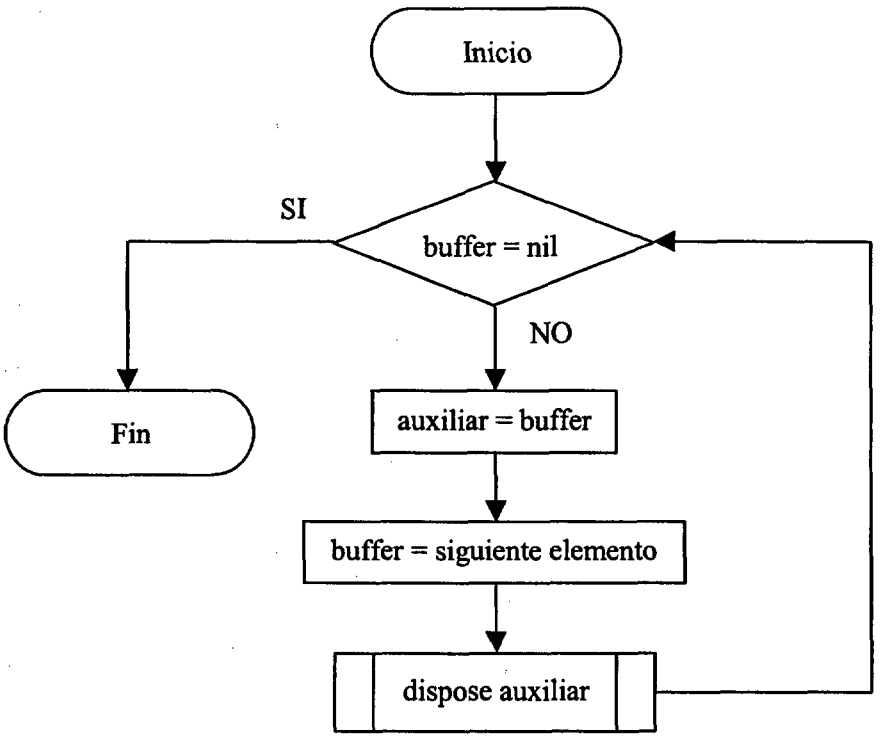


Figura 4. 35. Diagrama de flujos del método Borrarbuffer.

4.2.1.5.6. Método Camino.

El método Camino se ha añadido a la clase que describe la ficha. Crea una lista enlazada de elementos de tipo bfer soportada por el puntero buffer. La información que contiene corresponde a los identificadores de los nodos del circuito que relacionan a una entrada con el nodo cuyo tiempo de propagación desea conocer el usuario. Se construye el camino, o secuencia, de nodos que a partir de la excitación de entrada origina un tiempo de propagación en un nodo particular.

La lista se crea mediante llamadas al método Inbuffer. El último elemento de la lista, que es el primero en ser insertado, contiene el identificador del nodo solicitado por el usuario, que es el elemento apuntado por NodoEscogido. Los datos corresponden al identificador del nodo, cuyo valor se encuentra en el campo nodo_s, y el retardo que presenta. La información del nodo de entrada a la puerta representada por el elemento

apuntado por NodoEscogido que ha originado la respuesta temporal se encuentra almacenada en el campo respcambio.

Para construir la lista que constituye el camino se busca en la lista enlazada que representa al *netlist* el elemento de tipo Puerta cuyo nodo de salida corresponde al valor consignado en respcambio. Se utiliza el procedimiento Buscar declarado en la unidad sim4. Una vez encontrado se llama a Inbuffer para insertar este elemento en la lista que representa al camino. Este proceso se repite hasta que el campo respcambio del último nodo insertado sea cero, lo que indica que se trata de una entrada al circuito, finalizando la construcción de la lista.

Se remonta el circuito desde el nodo solicitado por el usuario hasta la entrada siguiendo los valores que contiene el campo respcambio de cada elemento integrante del camino. De esta forma se crea una estructura de tipo pila, en la que los elementos son introducidos en orden inverso, desde el nodo final hasta el nodo de entrada, para leerse después desde el nodo de entrada hasta el nodo final.

En la Figura 4.36 se muestra cómo se crea la lista:

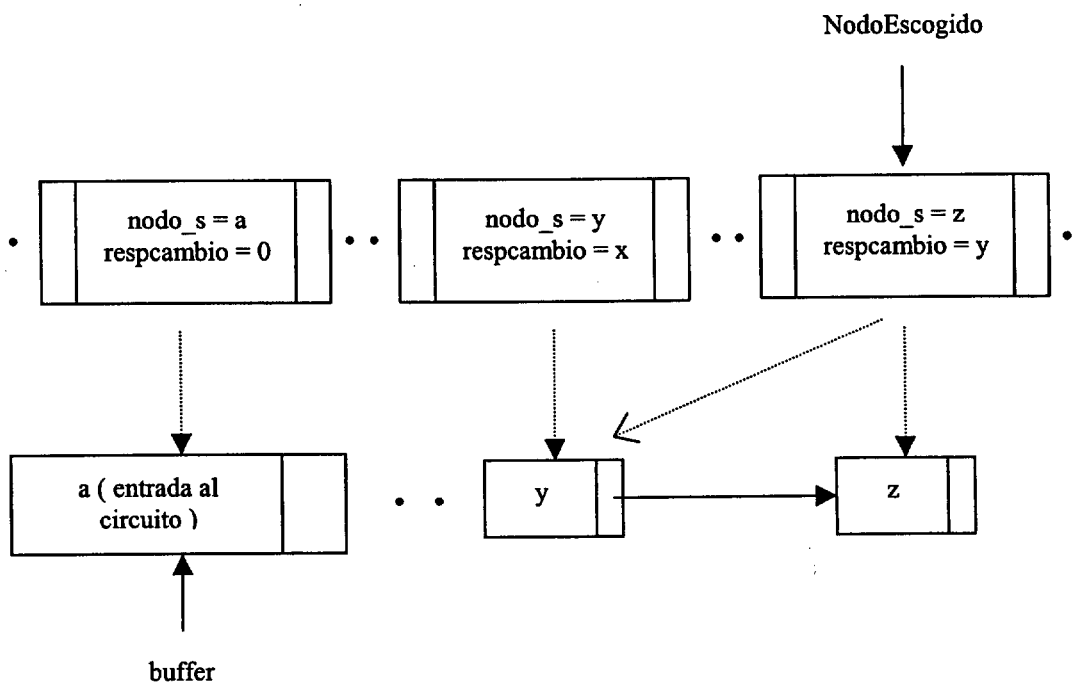


Figura 4. 36. Proceso de creación de la lista.

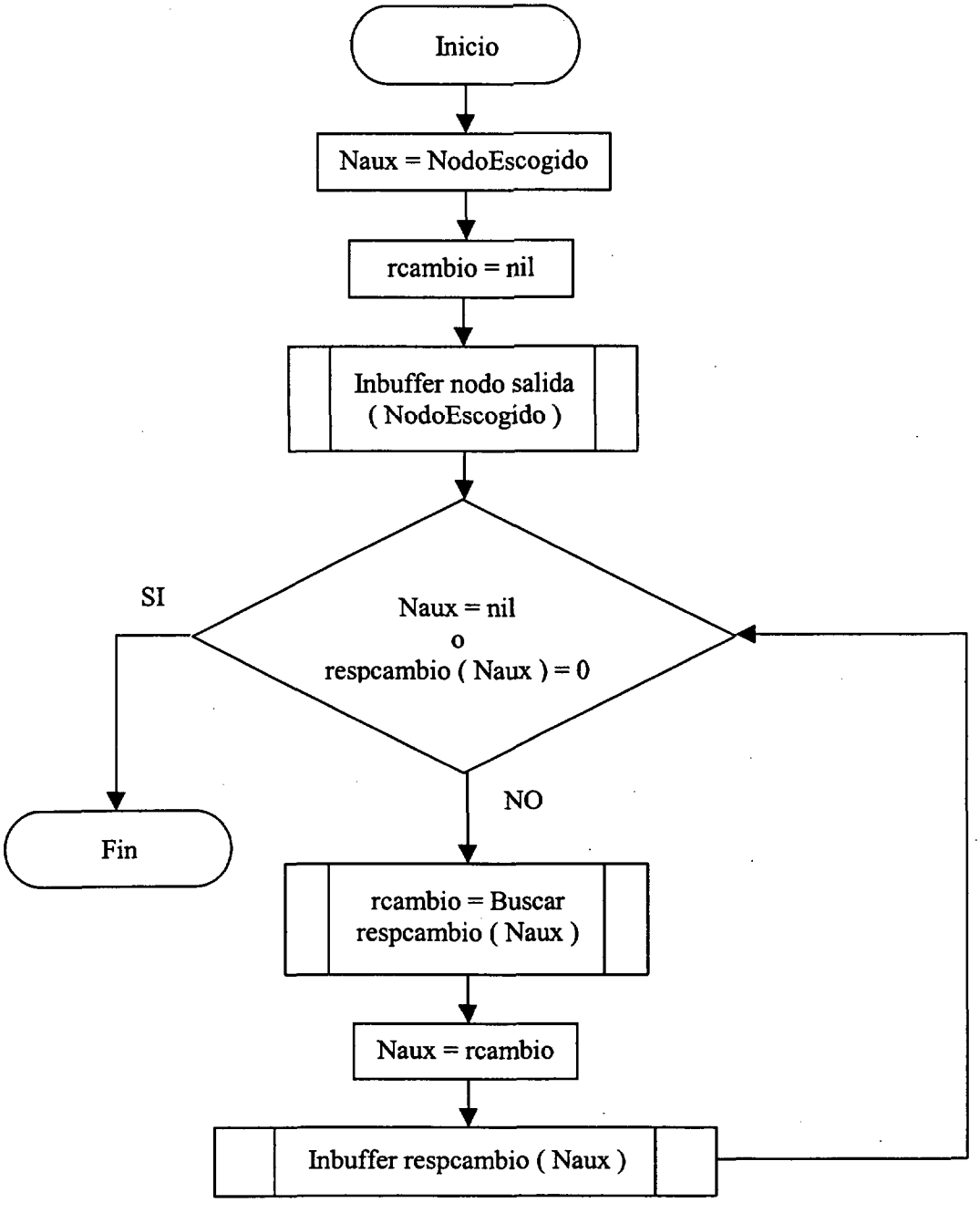


Figura 4. 37. Diagrama de flujos del método Camino.

4.2.1.5.7. Método Mostrar.

El método Mostrar recibe como parámetro, por referencia, una cadena que añade como nueva línea al texto contenido en el componente de edición CaminoM, de tipo *Memo*. La cadena se incorpora al elemento de edición haciendo uso del método *Add* de la clase *TStrings* a través de la propiedad *Lines*, de tipo *TStrings*, del componente de edición,

que permite manipular el texto en líneas individuales. La cadena recibida como parámetro se borra y se devuelve como cadena vacía.

4.2.1.5.8. Método RealACad.

El método RealACad es una función que devuelve un valor de tipo cadena de caracteres, *string*, por conversión del parámetro de tipo real que recibe. Dicho parámetro corresponde al tiempo de propagación de un nodo del circuito en picosegundos. Se llama a la función rcad, definida en la unidad sim4, que realiza la conversión de formato real a cadena de caracteres. La cadena obtenida se utiliza para presentar al usuario el dato del tiempo de propagación en el nodo que especifica. Se ha establecido que la parte decimal contenga dos dígitos, con lo que se eliminan los restantes caracteres correspondientes a la parte decimal que pueda tener la cadena. Para ello se detecta la posición del punto decimal, que se asigna a una variable índice, que se incrementa en dos unidades para borrar los caracteres finales de la cadena.

4.2.1.5.9. Método inicchart.

El método inicchart se ha añadido a la clase que define la ficha para inicializar el gráfico. Borra el título del gráfico y elimina los puntos de la serie que éste contiene.

El método comprueba mediante la propiedad *Title.Text*, de tipo *TStrings*, del componente Chart1 si contiene texto o está vacía. En caso de tenerlo se borra empleando el método *Delete* de la clase *TStrings*. El título del componente Chart1 se utiliza para especificar el nodo cuyo camino describe la serie que se muestra.

Para eliminar los datos existentes en la gráfica se verifica el número de puntos de la serie mediante la función *Count* de la clase *TChartSeries* de la que deriva el objeto Series1. Si el número de puntos es distinto de cero se eliminan todos los puntos de la serie usando el método *Clear* de *TChartSeries*. Si no se añaden nuevos puntos de datos a la serie el espacio del componente de gráficos de datos reservado para mostrarla permanece como un fondo blanco.

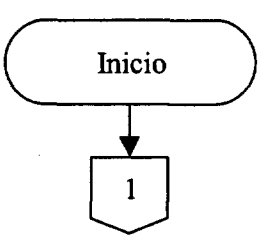
4.2.1.5.10. Método construirserie.

El método construirserie añade puntos de datos a la serie del gráfico Series1.

Inicialmente se asigna el título del gráfico a la propiedad *Title* del componente de gráficos de datos Chart1. Para ello se emplea el método *Add* de la clase *TStrings* a la que pertenece la subpropiedad *Text* de la propiedad *Title*, que va a contener verdaderamente el título.

Los datos para crear la serie corresponden a los nodos que forman el camino entre la entrada y el nodo solicitado por el usuario, y sus respectivos tiempos de propagación. Estos datos los contiene la lista enlazada que sostiene el puntero buffer. Mediante un puntero auxiliar se recorre la lista y se van añadiendo los datos a la serie. Los datos se agregan usando el método *AddXY* de la clase *TChartSeries*, en el que se especifican los valores x, nodo, e y, retardo, de cada punto, la etiqueta de cada valor x y el color del punto. Como etiqueta se utiliza una cadena equivalente a cada valor x de la serie que corresponde a un identificador de nodo. Se aplica la función predefinida de Delphi *IntToStr* para obtener dicha cadena a partir del valor de tipo entero que representa al nodo.

Los valores temporales obtenidos para cada nodo del camino se presentan respecto al origen de tiempos seleccionado por el usuario mediante la casilla de activación OrigenTemp, para lo que se comprueba el valor de la propiedad Checked de la misma. El puntero buffer apunta en todo momento al nodo inicial del camino. Si el usuario determina que se sitúe el origen en el primer nodo del camino, la propiedad Checked contiene el valor verdadero, se resta el contenido del campo retardo del primer nodo a los valores de todos los elementos de la lista antes de añadirlos a la serie. Si la propiedad Checked presenta el valor lógico falso el origen se sitúa en 0 sg, y los valores temporales recogidos en los elementos de la lista se insertan en la serie sin realizar modificaciones sobre ellos.



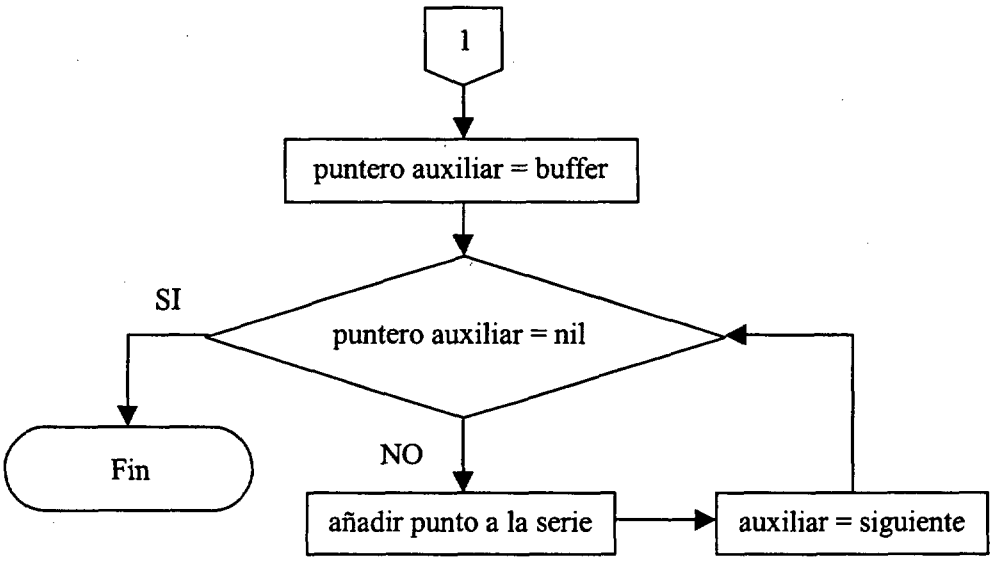


Figura 4. 38. Diagrama de flujos del método construirserie.

4.2.1.5.11. Método MostrarHint.

El método MostrarHint se incorpora a la ficha para presentar una descripción de un punto de la serie cuando el usuario sitúa el cursor del ratón sobre el mismo. Para ello se emplean las ayudas flotantes o sugerencias de la aplicación. Se trata de un texto que se muestra normalmente en un cuadro amarillo después de que el cursor del ratón permanezca estático sobre un control o componente durante una cantidad de tiempo fija. En su funcionamiento estándar basta con añadir el texto a la propiedad *Hint* del control o componente y asignar el valor lógico verdadero a la propiedad *ShowHints*.

Para el uso que se propone se debe configurar la manera en que se muestran las sugerencias. Esto se hace asignando un método al evento *OnShowHint* de la aplicación. La aplicación no usa para la gestión de sus eventos el **Inspector de Objetos de Delphi**. Éste es un elemento de Delphi que contiene una relación de los componentes y controles situados sobre la ficha, y que se emplea para modificar las propiedades de los mismos y fijar los eventos a que responden. Se utiliza también para los eventos y propiedades de la ficha. El método debe asignarse mediante código al evento OnShowHint del objeto *Application*, que Delphi crea de forma automática como instancia de la aplicación. Se hace cuando la ficha se crea en el método FormCreate.

El método MostrarHint debe tener tres parámetros concretos, definidos como parámetros por referencia, para poder asignarlo a *OnShowHint*. Se trata de una cadena de caracteres con el texto de la sugerencia, *HintStr*, un indicador lógico para su activación, *CanShow*, y una estructura con información adicional, *HintInfo*, de tipo *THintInfo*. Esta estructura define el aspecto y comportamiento de la sugerencia. Es un registro que contiene entre otros datos la posición en que se muestra la sugerencia y un indicador del control o componente de la ficha sobre el que se encuentra situado el cursor.

El método comprueba mediante el campo *HintControl*, de la clase *TControl*, del parámetro *HintInfo* si el componente sobre el que está el cursor es el componente de gráficos de datos Chart1. Es necesario comprobarlo pues aunque en la ficha Resultado sólo dicho componente tiene activa la ayuda flotante, su propiedad *ShowHints* vale verdadero, en la ficha principal de la aplicación los botones de la barra de herramientas también la tienen activa. Al ser la muestra de sugerencias un evento de la aplicación, no de la ficha, el código de MostrarHint afectaría a la visualización de las ayudas flotantes de la ficha principal impidiéndola.

Una vez verificado el control sobre el que se encuentra el ratón se asigna el valor falso al parámetro *CanShow*. Sólo se va a visualizar la sugerencia si se comprueba que el cursor se encuentra sobre un punto de la serie, no sobre el fondo del gráfico. Para ello se utiliza el método *GetCursorValueIndex* de la clase *TChartSeries*. Este método devuelve el índice del punto cercano a la posición del ratón. Si no hay punto cerca devuelve –1.

Si el cursor está sobre un punto de la serie se muestra la sugerencia. Para ello deben obtenerse los valores x, nodo, e y, tiempo de propagación, del punto para crear el texto, y calcularse las coordenadas de la pantalla donde situarlo.

Las propiedades *XValues* e *YValues*, de tipo *TChartValueList*, del objeto Series1 almacenan en tiempo de ejecución los valores de los puntos de la serie. Mediante el índice obtenido al aplicar *GetCursorValueIndex* se accede a los valores x e y del punto sobre el que se sitúa el cursor contenidos en *XValues* e *YValues*.

Se utilizan los métodos *XValueToText* e *YValueToText* pertenecientes a la clase *TChart* para convertir a formato de cadena de caracteres los valores x e y del punto. Con dichas cadenas se construye el texto de la sugerencia que se asigna al parámetro *HintStr*.

La propiedad *HintInfo.HintPos* del objeto *Application* determina el punto de la pantalla donde se muestra la ayuda flotante. Es una propiedad de tipo *TPoint*, que define un registro con dos campos de tipo entero, x e y, mediante los que se especifican las coordenadas de la pantalla donde colocar la sugerencia, que han de ser calculadas.

Para establecer la posición de la pantalla donde situar la sugerencia se toma como origen la esquina superior izquierda del componente de gráficos de datos Chart1. Las coordenadas correspondientes a este punto se obtienen de la propiedad *ClientOrigin*, de tipo *TPoint*, de Chart1. Esto va a permitir situar correctamente la sugerencia con independencia de la resolución de la pantalla.

La posición de la sugerencia se calcula sumando a dichas coordenadas las que definen la posición del punto sobre el que se sitúa el cursor. El método *GetCursorPos* de la clase *TChart* devuelve un valor de tipo *TPoint* correspondiente a la posición del cursor en el área del componente de gráficos de datos. Las coordenadas de este punto tienen como origen de referencia la esquina superior izquierda del componente Chart1. Sumando dichos valores a las coordenadas de la posición del componente en la pantalla se obtiene el punto donde situar la sugerencia, que se asigna al campo *HintPos* de la propiedad *HintInfo*.

Finalmente, se asigna el valor lógico verdadero al indicador *CanShow* para habilitar la visualización de la sugerencia.

4.2.1.5.12. Método FormCreate.

El método FormCreate responde al evento OnCreate, que tiene lugar cuando se crea la ficha. Se emplea este método para asignar el método MostrarHint como gestor del evento OnShowHint del objeto *Application*.

4.2.1.5.13. Método MosBtnClick.

Método gestor del evento OnClick del botón MosBtn. Muestra la información del nodo solicitado por el usuario. Consulta el valor de la propiedad *ItemIndex* del grupo de activación para determinar si pide la información del nodo crítico o selecciona directamente un nodo del circuito.

En el primer caso se realiza una llamada al método EncontrarNodoCritico, que busca el nodo que presenta mayor tiempo de propagación. En el segundo caso el nodo escogido por el usuario es el elemento activo en el cuadro de lista ListaNodos, identificado por su índice, el cual es contenido en la propiedad *ItemIndex* del cuadro de lista. Una vez se obtiene el identificador del nodo, el procedimiento Buscar, declarado en la unidad sim4, se usa para obtener la dirección del elemento de la lista que lo representa, que se asigna al puntero NodoEscogido.

Si el puntero NodoEscogido contiene el valor nil, ningún nodo del circuito presenta transición, lo cual se debe a que las entradas aplicadas sobre el circuito mantienen niveles fijos. A través de una etiqueta, componente de tipo *Label* situada en la ficha, se informa al usuario.

Si el puntero es distinto de nil se comprueba el valor del tiempo de propagación, campo tp, que presenta. Si es cero implica que el nodo permanece a nivel alto o bajo. Mediante la etiqueta referida se muestra el nivel presente en el nodo.

Si el tiempo de propagación es distinto de cero, su valor se convierte a cadena de caracteres por llamada al método RealACad y se presenta al usuario mediante una etiqueta. En este caso, se llama al método Camino que genera una lista enlazada soportada por el puntero buffer conteniendo el camino de nodos que conectan la entrada del circuito que origina la respuesta del nodo requerido con éste. Cada elemento de la lista contiene un campo de tipo entero que corresponde al identificador de un nodo del camino, y un campo de tipo real donde se consigna el tiempo de propagación de dicho nodo. El puntero buffer apunta al nodo inicial del camino. Para determinar el punto de referencia respecto al que se presenta el tiempo de propagación estimado se consulta el valor de la propiedad *Checked* de la casilla de activación OrigenTemp. Si contiene el valor lógico verdadero se toma

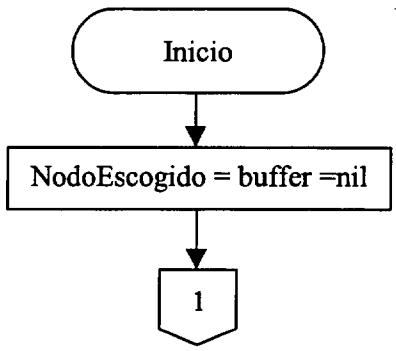
como origen de tiempos el primer nodo del camino, y se resta el valor del retardo que éste presenta al que se ha computado para el nodo seleccionado por el usuario.

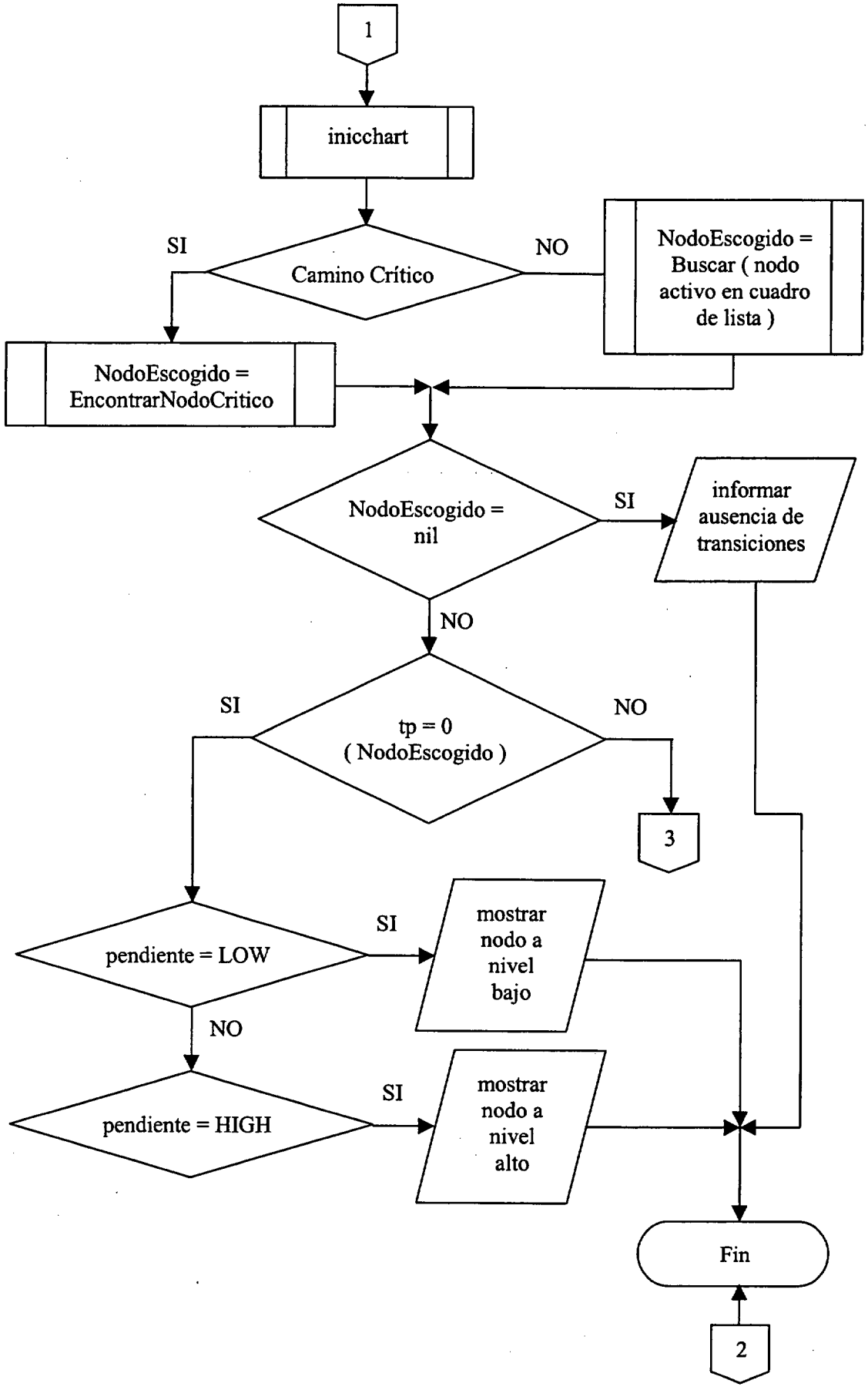
Los datos de la lista se muestran al usuario de la aplicación de dos formas diferentes: se presenta un listado de los nodos que constituyen el camino en forma de texto y se realiza una representación gráfica de los tiempos de propagación en cada nodo.

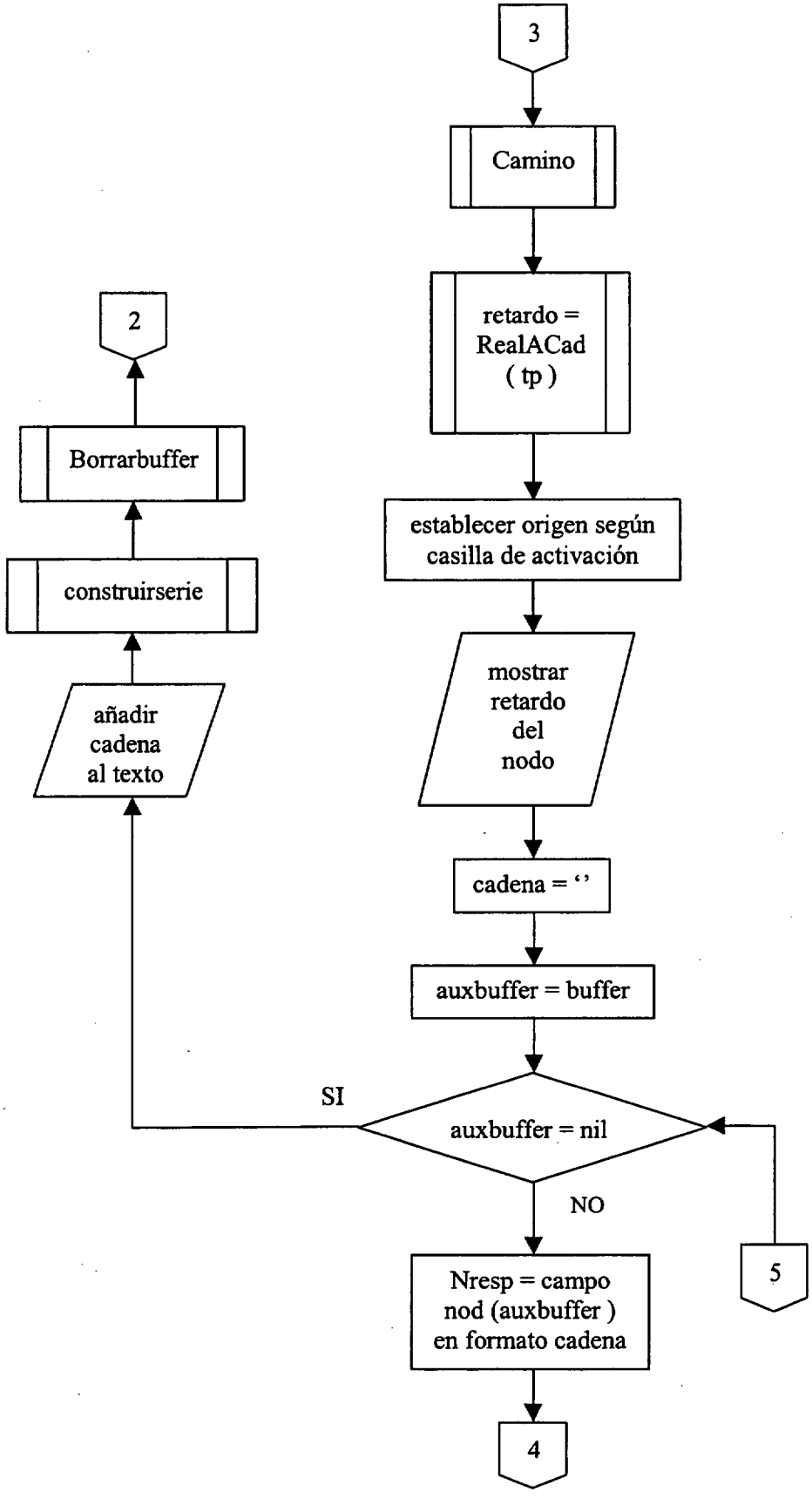
El camino se muestra como texto en un componente de edición, CaminoM, de tipo *Memo*. La información se estructura en líneas de texto de tipo cadena de caracteres, que contienen los identificadores de los nodos y flechas como elementos de unión. Para construir las líneas se utiliza un puntero auxiliar que recorre la lista enlazada extrayendo la información de cada elemento, que se inserta en la cadena de forma alterna con los caracteres que componen la flecha. Con objeto de conseguir que las líneas sean uniformes en tamaño, antes de insertar los caracteres en la cadena se compara su longitud con un valor establecido. Si el número de caracteres de la cadena supera dicha cifra se llama al método Mostrar, que añade la línea al texto presente en el componente de edición y se inicia una nueva línea con los caracteres no insertados. Este proceso de escritura concluye cuando se haya recorrido completamente la lista, es decir, cuando el puntero auxiliar apunte a nil.

La representación gráfica se realiza sobre el componente de gráficos de datos Chart1, para lo que se llama al método construirserie.

Finalmente se elimina la lista enlazada soportada por buffer utilizando el método Borrarbuffer y se sitúa el cursor del componente de edición al principio del texto escrito.







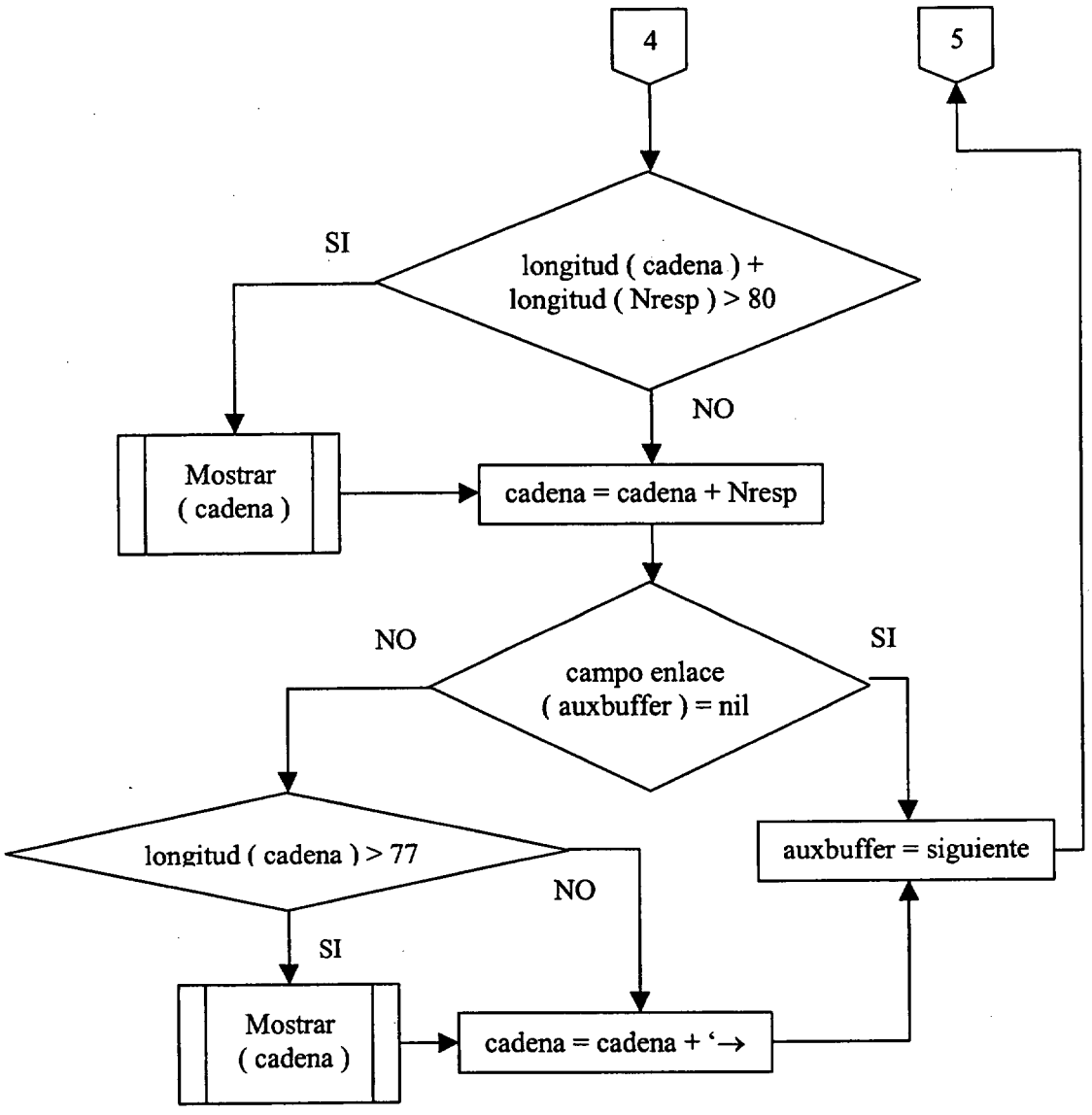


Figura 4. 39. Diagrama de flujos del método MosBtnClick.

4.2.1.6. Unidad princ5.

La unidad princ5 contiene la ficha principal de la aplicación, FormPFC, y el código asociado a la misma. El usuario de la aplicación puede modificar su tamaño. Sobre la ficha se ha situado una serie de componentes y controles cuyo cometido se detalla a continuación.

Se ha emplazado un componente de edición, tipo *RichEdit*, alineado con el área de cliente de la ficha. Se utiliza para la edición de los distintos tipos de ficheros con que trabaja la aplicación. La simulación tiene lugar sobre el fichero *netlist* (*.net) abierto sobre este componente.

Se ha dispuesto sobre la ficha un componente *MainMenu* para crear el menú principal de la aplicación. Contiene los siguientes menús desplegables: Archivo, Editar, Ver, Simular y Ayuda. Se ha establecido que cada elemento de menú cuente con el llamado acelerador gráfico, que es una letra subrayada del texto del elemento que permite su selección mediante el uso del teclado pulsando la tecla Alt más la letra subrayada.

Se ha creado una barra de herramientas emplazando un componente *Panel* en la parte superior del área de cliente de la ficha y situando varios botones rápidos en su interior, componentes *SpeedButton,* cuyo uso principal es en barras de herramientas. Estos botones van a realizar funciones propias de los menús desplegables Archivo y Editar. Los métodos creados para tales comandos de menú se asocian a los respectivos botones de la barra de herramientas.

El menú desplegable Archivo presenta elementos típicos de este tipo de menú como Abrir, Guardar, Guardar como y Salir. El comando Nuevo contiene un segundo nivel desplegable, con un elemento específico para la creación de ficheros de entradas y otro elemento para editar los restantes tipos de ficheros soportados por la aplicación. En cada caso se sondea si el archivo actual ha cambiado, guardándolo sólo si es cierto. Se invita al usuario a que guarde el archivo cada vez que el programa cree uno nuevo, abra otro ya existente o finalice.

El menú desplegable Ver consta de dos comandos de menú, cada uno de los cuales muestra u oculta respectivamente la barra de herramientas y la barra de estado. Inicialmente ambos comandos ocultan las barras, pues éstas, al arrancar la aplicación son visibles.

Los comandos del menú Editar realizan funciones propias de cualquier editor de texto, como son: copiar, cortar, borrar, pegar y seleccionar todo. Los elementos de menú Cortar, Copiar y Borrar, así como los botones equivalentes de la barra de herramientas,

sólo estarán habilitados cuando haya sido seleccionado texto presente en el componente de edición. En el caso del comando Pegar, y su correspondiente botón, cuando exista texto disponible en el portapapeles.

Para poder hacer uso del portapapeles se declara la unidad Clipbrd de Delphi en la cláusula **uses** de la sección **implementation** de princ5.

El menú desplegable Simular contiene cuatro elementos de menú, que corresponden a la opción de simular el fichero *netlist* abierto, la traducción de dicho fichero a formato SPICE, la traducción de un fichero SPICE al formato del modelo y el acceso a los resultados producto de la simulación. Este último comando de menú solamente es habilitado al concluir con éxito una simulación.

El menú desplegable Ayuda permite al usuario de la aplicación acceder a la ayuda del programa.

Se ha añadido un menú emergente, componente de tipo *PopupMenu*, el cual se asocia al componente Editor de tipo *RichEdit*. Para asignar el menú se fija la propiedad *PopupMenu* del componente de edición. Se activa al pulsar el botón derecho del ratón sobre el componente de edición. Realiza funciones de edición, las correspondientes a los comandos del menú Editar, con lo que se enlazan los métodos de éstos con los eventos generados por los elementos del menú emergente. La habilitación de los comandos del menú se ciñe a las mismas condiciones impuestas para los elementos del menú Editar.

Para disponer de una barra de estado se ha emplazado un componente *StatusBar* en la parte inferior del área de cliente de la ficha. Muestra información sobre el estado del programa y descripciones de determinados comandos de menú cuando el usuario sitúa el cursor del ratón sobre ellos. Éstas se realizan mediante el sistema de sugerencias de la aplicación, introduciendo el texto descriptivo como contenido de la propiedad *Hint* de cada elemento y manejando el evento OnHint de la aplicación.

Desde el código de la unidad se muestran las fichas secundarias descritas con anterioridad. Para poder hacer uso de ellas se declaran en la cláusula **uses** de la sección **implementation** las unidades que las contienen.

Se ha declarado un tipo de datos de utilización en las rutinas de conversión entre ficheros de formato SPICE y del modelo. El tipo PInfoS define un puntero al tipo InfoS. Éste identifica a un registro utilizado para crear una lista enlazada donde se almacenan los distintos valores de W_E, anchura del transistor de enriquecimiento, en el caso de traducción a SPICE, empleados para la declaración de subcircuitos, o de β, en la conversión inversa, que para cada tipo de puerta básica, inversor y NOR2, presenta el circuito. El valor se almacena en formato de cadena de caracteres en el campo bet. El campo nombre recoge el correspondiente nombre con que se identifica cada subcircuito definido en el fichero SPICE. Finalmente, el campo next de tipo PInfoS actúa como enlace entre los elementos de la lista enlazada.

```
PInfoS = ^InfoS;
InfoS = record
    nombre,bet: string;
    next: PInfoS;
end;
```

Se ha añadido a la clase que describe la ficha 5 campos y 12 métodos. Todos los campos han sido declarados bajo la cláusula **private**. Dos de dichos campos, NombreFich y FichEtradas, son de tipo cadena de caracteres, *string*, y tienen por utilidad almacenar los nombres de ficheros usados por el programa. Se ha declarado un campo de tipo lógico, *boolean*, llamado *valido*. Este campo refleja la validez de los datos presentes en el *netlist* sobre el que se realiza la simulación. Los dos campos restantes son de tipo PunteroPuerta, declarado en la unidad sim4, y apuntan a los elementos inicial y final de la lista doblemente enlazada que representa al *netlist*. La unidad sim4 se declara en la sección **interface**, bajo la cláusula **uses**, pues sus declaraciones se utilizan para las declaraciones de princ5. Los métodos Salvar, SalvarCambios y SalvarComo se emplean en el código asociado a los comandos de tratamiento de ficheros del menú Archivo. Los métodos InModelo, Entradas y ParSPICE se utilizan para la traducción del formato del modelo a SPICE. Localmente al método ConvertirClick se definen 13 procedimientos y funciones empleados en la traducción de un fichero SPICE al formato del modelo.

Se describen a continuación los métodos contenidos en la unidad.

4.2.1.6.1. Método Peg1Click.

Es el método gestor del evento OnClick del elemento de menú Peg1, 'Pegar'. El método se asocia al botón PegarBtn y al elemento del menú emergente PegPup1 como gestor de sus respectivos eventos OnClick. Hace uso del método de la clase *TRichEdit,* a la que pertenece el componente de edición, *PasteFromClipboard*, que copia el contenido del portapapeles en el componente de edición en la posición del cursor.

4.2.1.6.2. Método Bor1Click.

Método enlazado como gestor de los eventos OnClick sobre el elemento de menú desplegable Bor1, 'Borrar', el elemento del menú emergente BorPup1 y el botón BorrarBtn. Borra el texto seleccionado en el componente de edición. Para realizar dicha acción se utiliza el método de la clase *TRichEdit ClearSelection.*

4.2.1.6.3. Método Sel1Click.

Este método es el gestor del evento OnClick del elemento de menú Sel1, 'Seleccionar todo'. Se asocia como gestor del mismo evento del elemento del menú emergente SelPup1. Emplea el método de la clase *TRichEdit SelectAll* que selécciona todo el texto presente en el componente de edición.

4.2.1.6.4. Método Cop1Click.

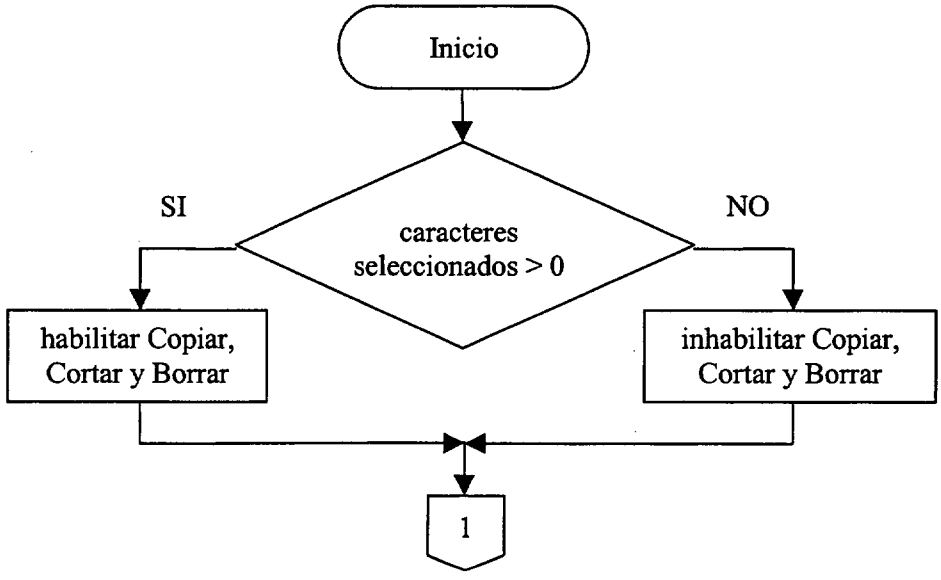
Gestor del evento OnClick del elemento de menú Cop1, 'Copiar', enlazado a dicho evento del botón CopiarBtn y del elemento del menú emergente CopPup1. Copia el texto seleccionado en el portapapeles. Se utiliza para ello el método *CopyToClipboard* perteneciente a la clase *TRichEdit*. Habilita el botón de la barra de herramientas PegarBtn asignando el valor lógico verdadero a su propiedad *Enabled*, toda vez que introduce texto en el portapapeles.

4.2.1.6.5. Método Cort1Click.

Gestiona el evento OnClick sobre el elemento de menú Cort1, 'Cortar'. Se ha fijado como gestor del mismo evento sobre el botón CortarBtn y el elemento del menú emergente CortPup1. Borra el texto seleccionado en el componente de edición tras haberlo copiado en el portapapeles. Dicha función la realiza el método *CutToClipboard* de la clase *TRichEdit*. Dado que inserta texto en el portapapeles habilita el botón de la barra de herramientas PegarBtn al fijar la propiedad *Enabled* del mismo al valor lógico verdadero.

4.2.1.6.6. Método Edit1Click.

Es el método gestor del evento OnClick sobre el menú desplegable Edit1, 'Editar'. El comando 'Seleccionar todo' está siempre habilitado, mientras que los demás comandos del desplegable se habilitan como respuesta al evento tratado, toda vez que sólo son accesibles al desplegar el menú. Para saber si hay texto seleccionado se consulta el valor contenido en la propiedad *SelLength* del componente de edición, que corresponde al número de caracteres seleccionados. Si dicha cantidad es mayor que cero se habilitan los elementos Copiar, Cortar y Borrar. La disponibilidad de texto en el portapapeles se comprueba mediante la función *HasFormat* del mismo, que devuelve un valor lógico verdadero en caso positivo. Se habilita de esta forma el comando Pegar.



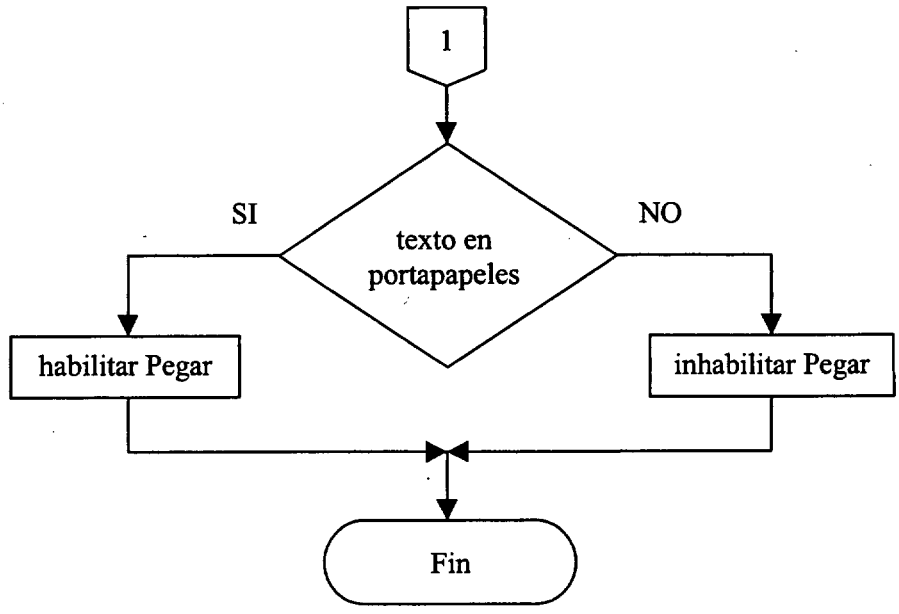
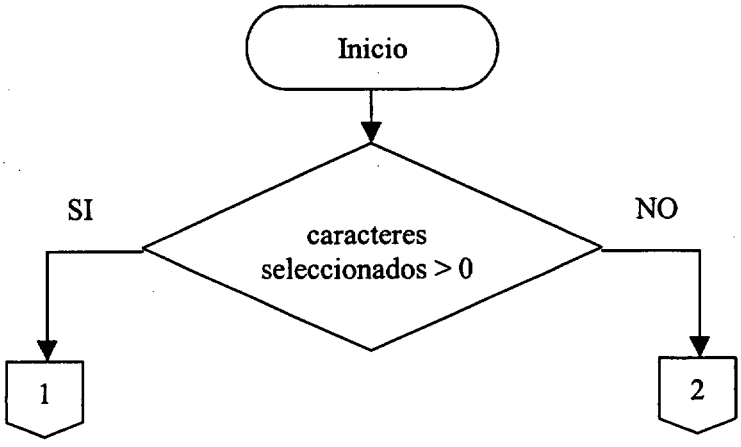


Figura 4. 40. Diagrama de flujos del método Edit1Click.

4.2.1.6.7. Método EditorSelectionChange.

Este método maneja el evento OnSelectionChange del componente de edición. El evento ocurre como respuesta a cambios en la selección de texto. El código creado habilita los botones de la barra de herramientas Copiar, Cortar y Borrar, que están activos cuando el usuario selecciona texto presente en el elemento de edición. Se comprueba la propiedad *SelLength* del componente editor y se actúa sobre la propiedad *Enabled,* de tipo lógico, de cada uno de los botones, habilitándolos si aquella registra una cantidad mayor que cero.



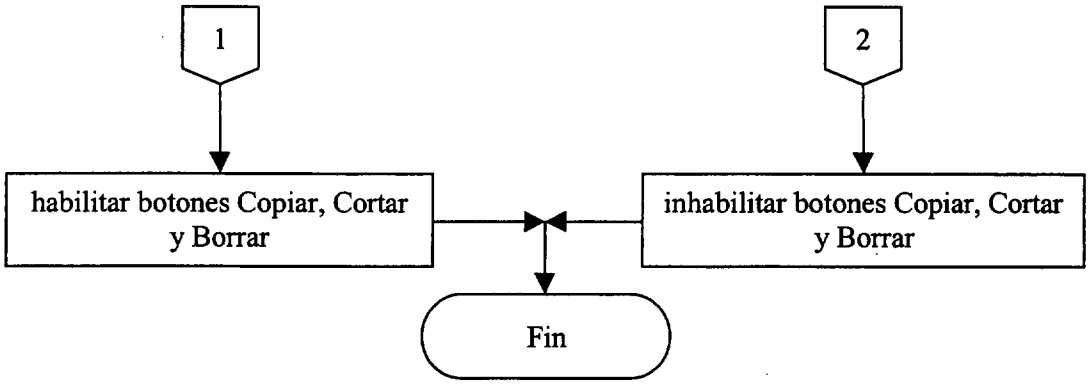
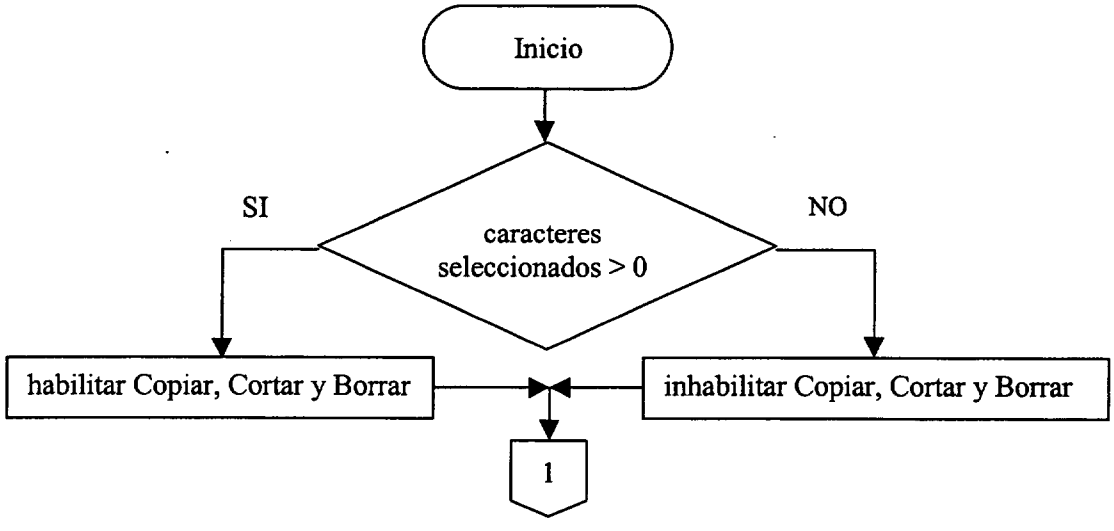


Figura 4. 41. Diagrama de flujos del método EditorSelectionChange.

4.2.1.6.8. Método PopupM1Popup.

El método gestiona el evento OnPopup correspondiente al menú emergente. Al pulsar el botón derecho del ratón sobre el componente de edición se muestra el menú, provocando el envío del evento OnPopup a la aplicación. El código asociado a este evento se ha creado para habilitar los comandos del menú emergente. Los elementos Copiar, Cortar y Borrar se habilitan si se ha seleccionado texto en el componente de edición, lo cual se verifica consultando la propiedad *SelLength* del mismo. La habilitación del comando Pegar tiene lugar si se dispone de texto en el portapapeles, condición que se comprueba a través de la función *HasFormat* del portapapeles, *Clipboard*.



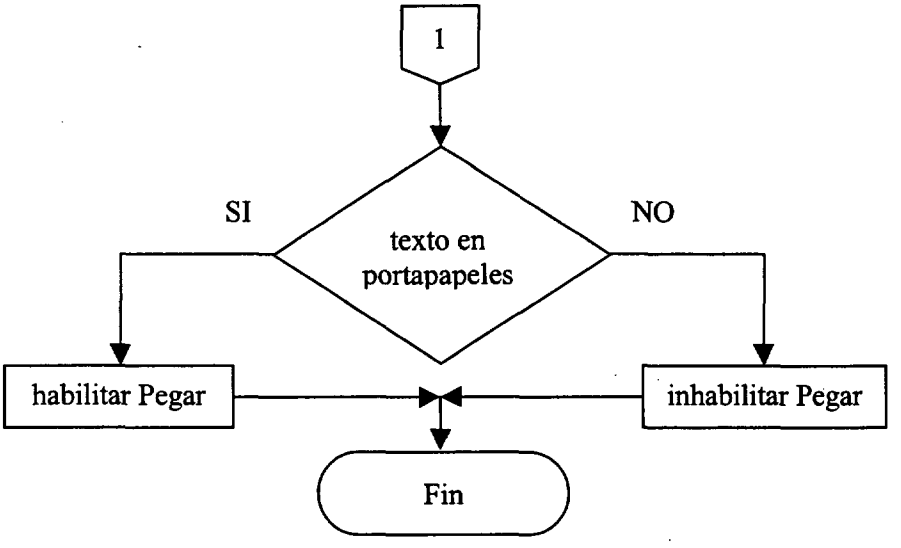


Figura 4. 42. Diagrama de flujos del método PopupM1Popup.

4.2.1.6.9. Método BH1Click.

El método maneja el evento OnClick del elemento de menú BH1, perteneciente al menú desplegable Ver. Oculta o muestra la barra de herramientas de la aplicación, invirtiendo el estado de la propiedad *Visible* de la barra. Modifica el texto del elemento de menú, contenido en la propiedad *Caption,* para indicar la acción contraria a la última realizada.

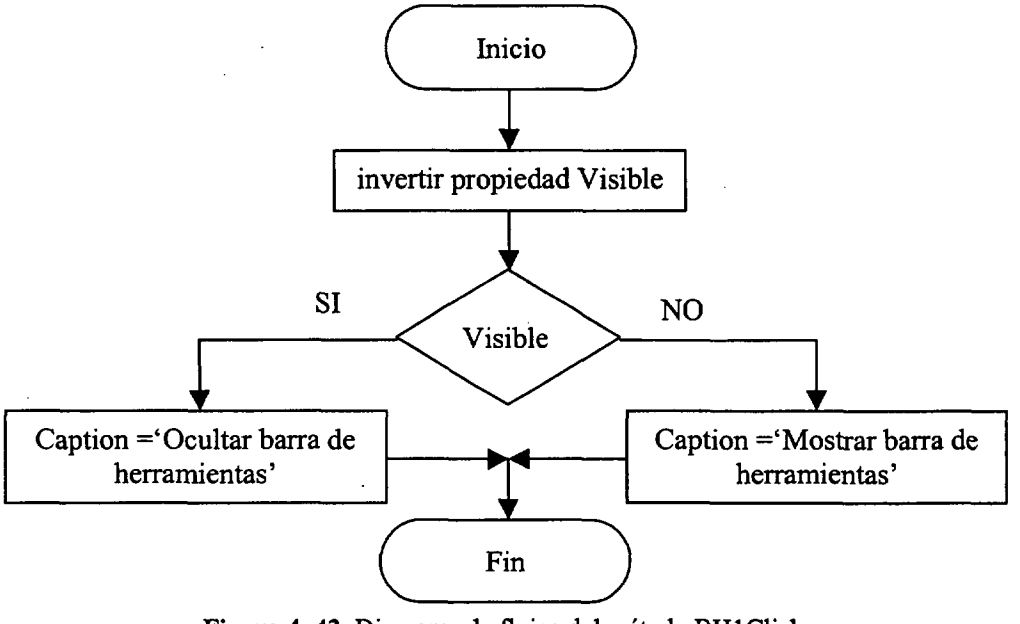


Figura 4. 43. Diagrama de flujos del método BH1Click.

4.2.1.6.10. Método BE1Click.

El método BE1Click gestiona el evento OnClick del elemento de menú BE1 perteneciente al menú desplegable Ver de forma análoga al método BH1Click de la barra de herramientas. Oculta o muestra la barra de estado de la aplicación al invertir el estado de la propiedad *Visible* de la misma. Se modifica el texto del elemento de menú, propiedad *Caption*, para indicar la acción contraria a la última efectuada.

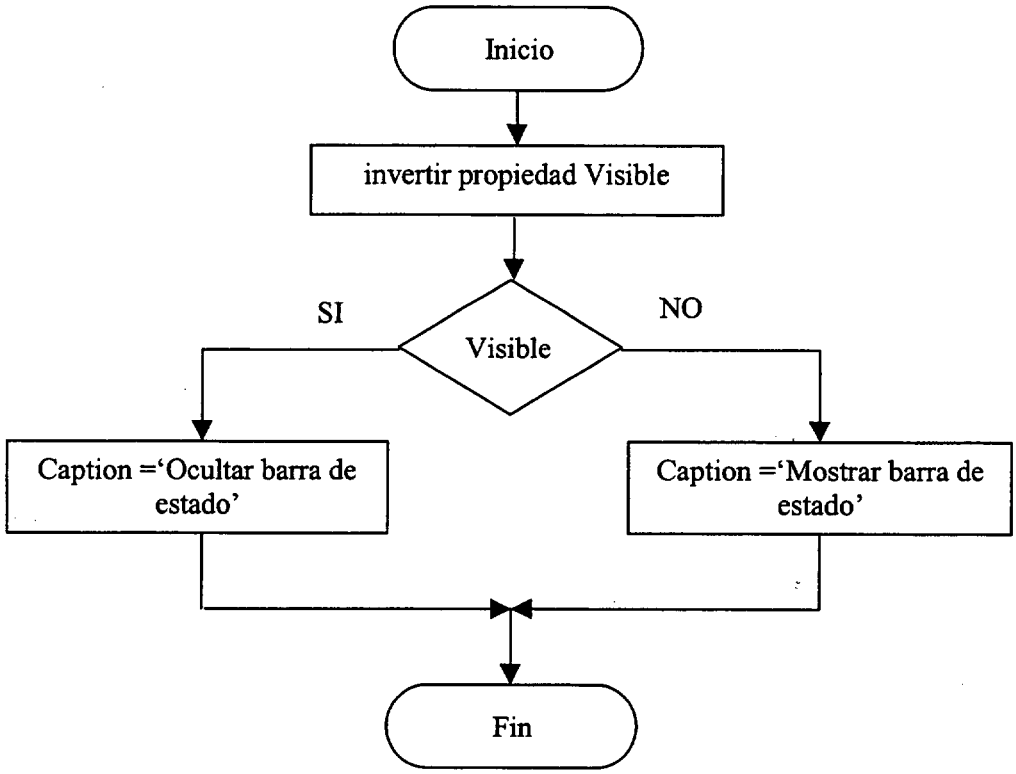


Figura 4. 44. Diagrama de flujos del método BE1Click.

4.2.1.6.11. Método Liberar.

Método agregado a la clase que describe la ficha. Verifica si la lista enlazada que representa al *netlist* está vacía. Para ello comprueba la dirección a la que apunta el puntero PlistaNodos, comienzo de lista. Si contiene un valor distinto de la constante nil la lista no está vacía y se llama al procedimiento Borrarlista, declarado en la unidad sim4, que borra la lista, liberando el espacio de memoria que ésta ocupaba.

4.2.1.6.12. Método SalvarComo.

El método se ha añadido a la ficha. El código generado realiza una llamada al cuadro de diálogo predefinido SaveDialog1, componente de tipo *SaveDialog*, para que el usuario seleccione el nombre con que guardar el fichero actual. El método *Execute* de la clase *TSaveDialog* devuelve el valor lógico falso si el usuario cancela la operación o verdadero si pulsa el botón Guardar. La propiedad *Filename* del cuadro de diálogo registra, en este último caso, el nombre escogido, que se asigna al campo NombreFich. Se produce una llamada al método Salvar que guarda el fichero con el nombre designado.

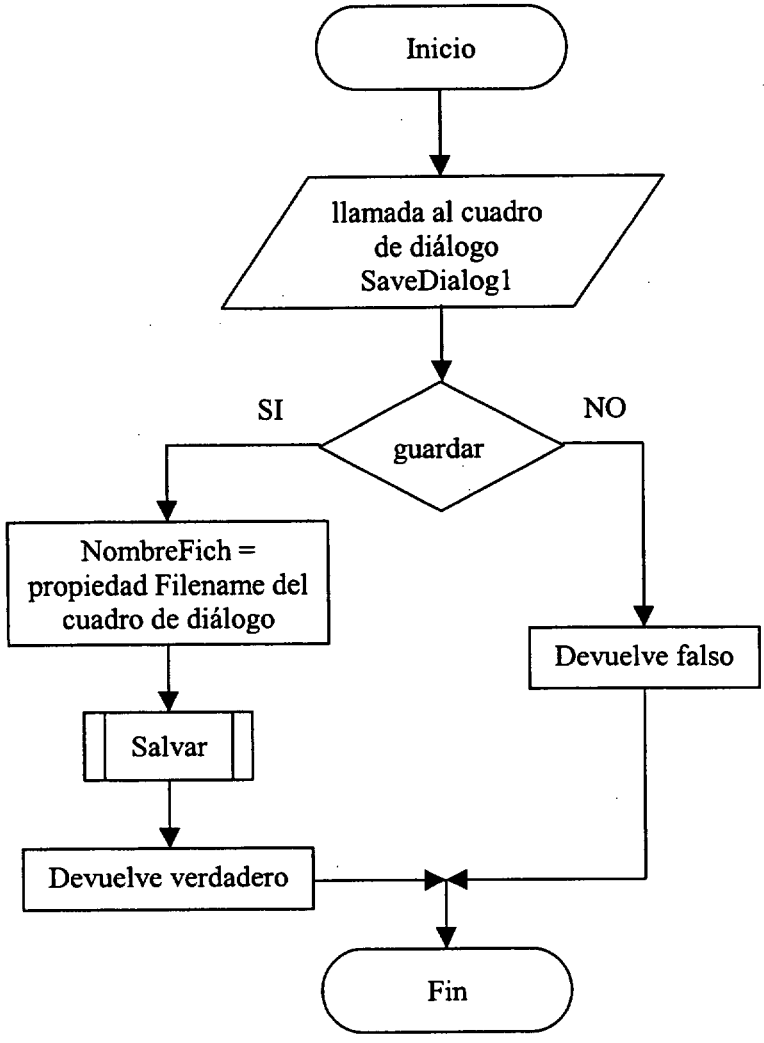


Figura 4. 45. Diagrama de flujos del método SalvarComo.

4.2.1.6.13. Método Salvar.

Método incorporado a la clase que describe la ficha en forma de función. Guarda el archivo si ya tiene un nombre de archivo apropiado, o pide al usuario de la aplicación que introduzca un nombre llamando al método SalvarComo. Si se ha asignado un nombre para el fichero, consignado en el campo NombreFich, se guarda el contenido del componente de edición utilizando el método *SaveToFile*. Este método es propio de la clase *TStrings*, a la cual pertenece la propiedad *Lines* del componente de edición. A dicho método se le pasa como parámetro el nombre del fichero. El método Salvar asigna el valor lógico falso a la propiedad *Modified*, que registra si ha habido cambios en el texto del elemento editor, inicializándola, y devulve el valor verdadero.

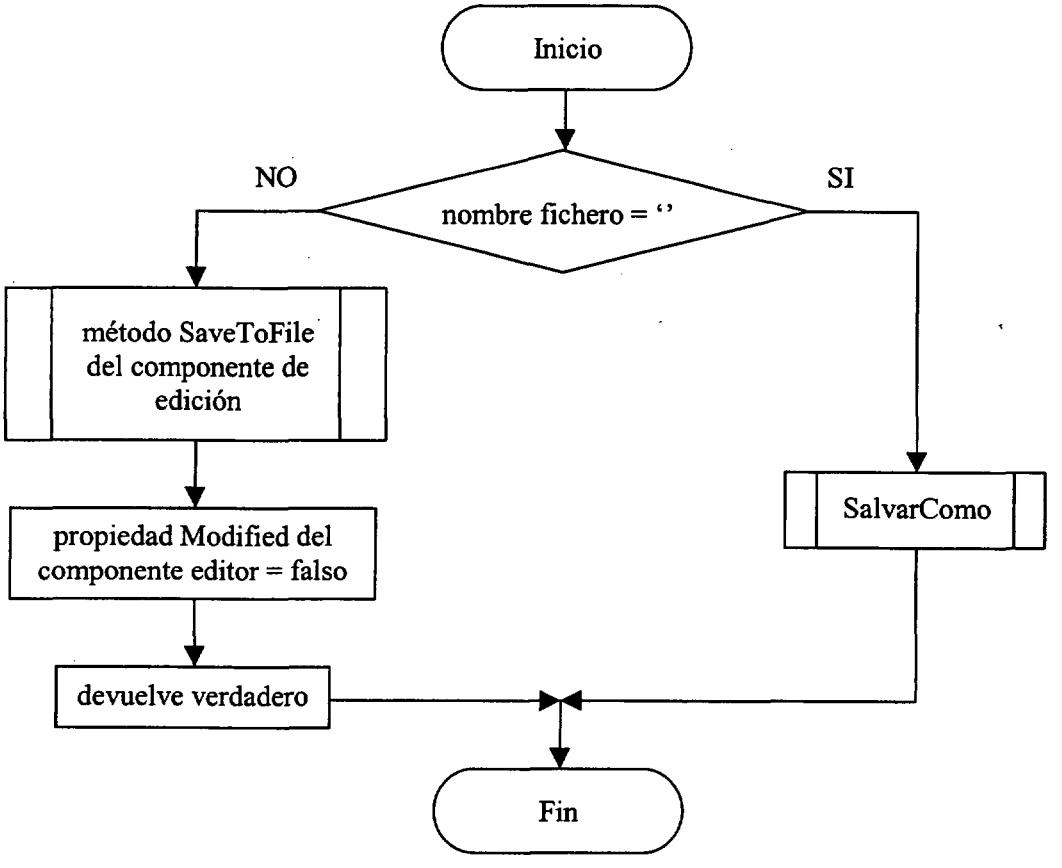


Figura 4. 46. Diagrama de flujos del método Salvar.

4.2.1.6.14. Método SalvarCambios.

Este método se ha añadido a la clase que describe la ficha. Se trata de una función que devuelve un valor lógico. Pregunta al usuario si desea guardar los cambios en el fichero activo, descartarlos o saltar la operación en curso. Para ello, se utiliza la ficha secundaria SelSalv, unidad sec4, caracterizada como cuadro de diálogo que se llama en forma modal. A sus botones se les ha signado valor de retorno, *ModalResult*, y al ser pulsados se cierra la ficha y se devuelve el valor correspondiente. Si el usuario selecciona el botón Cancelar la función devuelve el valor falso. Si selecciona No, el archivo no se guarda, pero devuelve el valor verdadero indicando que la acción requerida, como crear un nuevo archivo, puede realizarse. Si se selecciona Sí, se produce una llamada al método Salvar y se devuelve el resultado, también de tipo lógico, que éste entrega.

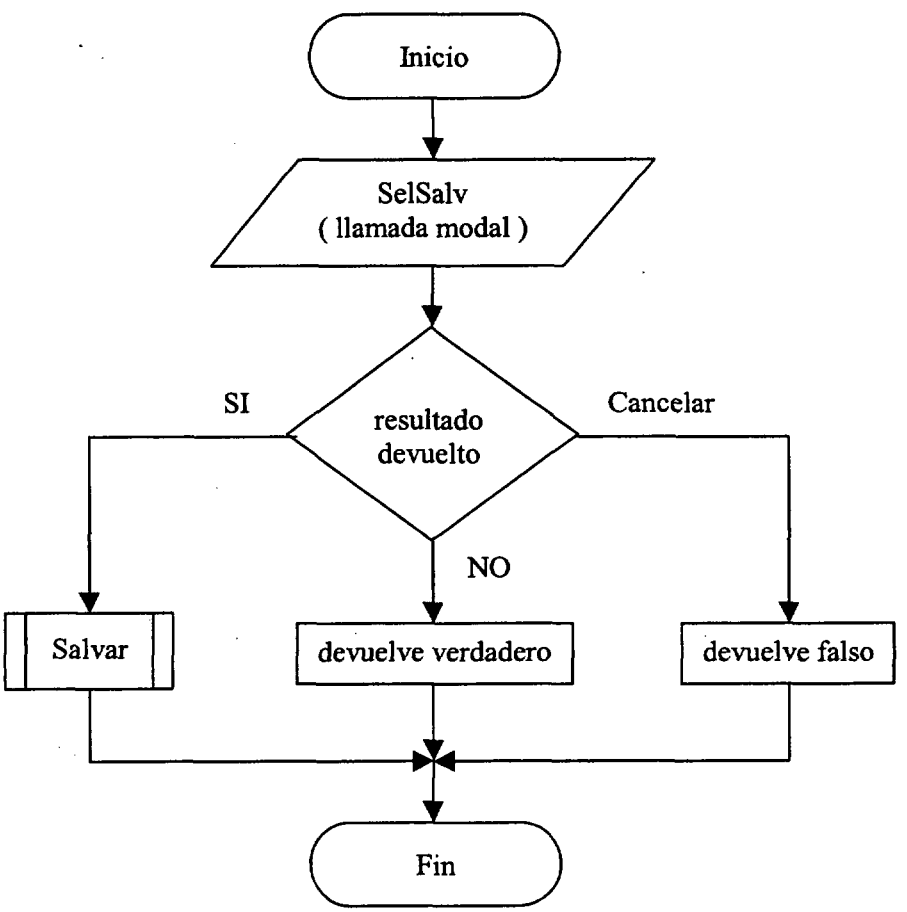


Figura 4. 47. Diagrama de flujos del método SalvarCambios.

4.2.1.6.15. Método Sal1Click.

El método maneja el evento OnClick sobre el elemento del menú Archivo Sal1, Salir. Procede al cierre de la ficha. Al tratarse de la ficha principal de la aplicación el cierre de ésta supone la finalización de la aplicación. Dicha acción se realiza mediante llamada al método predefinido *Close*.

4.2.1.6.16. Método FormCloseQuery.

El evento OnCloseQuery que maneja el método se produce cada vez que el usuario intenta cerrar la ficha. Se trata de un evento perteneciente a la ficha, no a los controles usados para cerrarla.

El cierre de la ficha, y por tanto de la aplicación, se establece fijando el valor del parámetro *CanClose*, de tipo lógico, propio del método. Asignándole el valor falso se detiene el proceso de finalización.

Si el archivo actual no ha sido modificado se permite el cierre de la aplicación, asignando el valor verdadero al parámetro *CanClose*. La existencia de cambios en el fichero se detecta consultando el valor de la propiedad *Modified*, de tipo lógico, del componente de edición. Si se ha modificado el archivo *Modified* valdrá verdadero y se llama al método definido como función SalvarCambios. Dicha función pregunta al usuario de la aplicación si desea guardar los cambios, descartarlos o cancelar la operación en curso. Devuelve un valor lógico verdadero en los dos primeros supuestos y falso en el tercero. Este valor se asigna al parámetro *CanClose*, con lo que solamente en caso de seleccionar la opción cancelar se impide el cierre de la ficha.

Si el parámetro *CanClose* recibe el valor verdadero se llama al método Liberar para borrar la lista enlazada que representa al *netlist*.

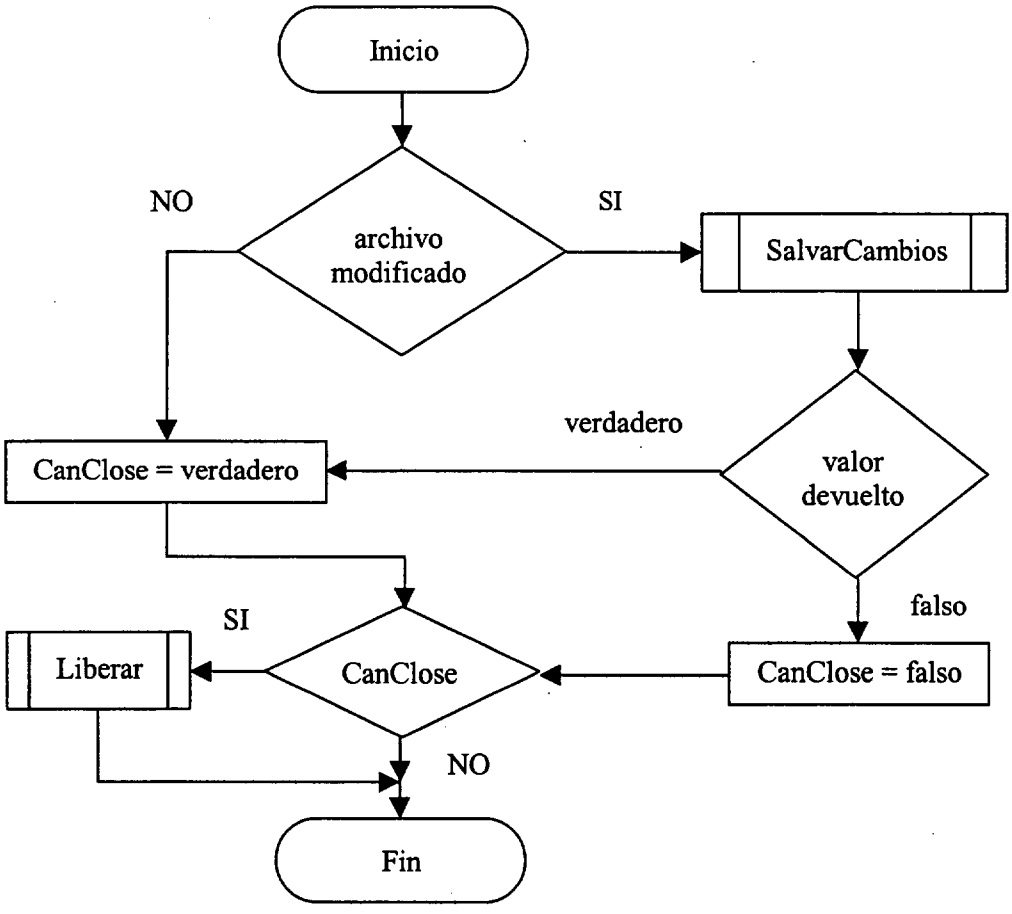


Figura 4. 48. Diagrama de flujos del método FormCloseQuery.

4.2.1.6.17. Método ShowHint.

Este método se ha incorporado a la clase para manejar el evento OnHint de la aplicación. Es un evento de la aplicación, no de un acto sobre un componente o control. El método debe asignarse al evento mediante código, y se realiza en el método FormCreate mediante asignación directa al evento OnHint del objeto Application.

El método muestra en la barra de estado una descripción de un comando de menú cuando el ratón se mueve sobre él. El texto descriptivo se asigna a la propiedad *Hint* del elemento de menú en tiempo de diseño. Cuando se sitúa el ratón sobre el comando se copia temporalmente dicho texto en la propiedad *Hint* de la aplicación. Se asigna dicho contenido a la propiedad *SimpleText* de la barra de estado que ha sido declarada como panel simple. Esta propiedad *SimpleText* muestra en la superficie de la barra de estado el texto recibido.

4.2.1.6.18. Método FormCreate.

Cuando la ficha se crea se recibe el evento OnCreate, que puede ser utilizado para inicializar campos de la ficha, así como propiedades de ésta o de los componentes que contiene.

El método creado para gestionar este evento inicializa cinco campos agregados a la clase que representa a la ficha. Los campos NombreFich y FichEntradas, de tipo *string*, son inicializados como cadenas vacías; el campo *valido*, de tipo lógico, se inicializa a falso; y los campos PlistaNodos y FlistaNodos, punteros de tipo PunteroPuerta, declarado en la unidad sim4, se inicializan a la constante nil.

Tal como se indicó al describir el método ShowHint se asigna dicho método al evento OnHint de la aplicación.

4.2.1.6.19. Método Guard1Click.

Es el método gestor del evento OnClick sobre el comando del menú Archivo Guard1, Guardar. Se asocia al evento OnClick sobre el botón GuardarBtn de la barra de herramientas. Las acciones a realizar por el código del método se ejecutan si el archivo actual ha sido modificado. La propiedad *Modified* del componente de edición adquiere el valor verdadero como consecuencia de la modificación del texto que contiene. En este caso se llama al método Salvar, definido como función, cuyo valor de retorno se ignora. Este método guarda en el fichero cuyo nombre contiene el campo NombreFich el texto presente en el elemento de edición.

Si el fichero es de tipo *netlist* el indicador *valido* es devuelto a su estado inicial, es decir, se le asigna el valor falso ya que el archivo ha sido modificado y los datos que contiene no han sido verificados en cuanto a formato. Se llama al método Liberar, que elimina la lista enlazada, pues si existe, ésta no contiene los nuevos datos incorporados al fichero. Se inhabilita el elemento de menú ReSim1, Resultados de la simulación, del menú Simular, pues la lista enlazada, que contiene los resultados de la simulación se ha eliminado.

4.2.1.6.20. Método GComo1Click.

El método maneja el evento OnClick sobre el elemento GComo1, Guardar como, del menú Archivo. Llama al método SalvarComo que ha sido declarado como función, y cuyo valor de retorno se ignora. Dicho método solicita al usuario el nombre con que ha de guardarse el archivo actual. De igual forma que en el método Guard1Click si se trabaja con un fichero *netlist*, se devuelven a sus valores iniciales el campo *valido* y la propiedad *Enabled* del elemento de menú ReSim1, y se llama al método Liberar para borrar la lista enlazada que representa al *netlist*.

4.2.1.6.21. Método Abr1Click.

El método se asigna al evento OnClick del elemento Abr1, Abrir, del menú Archivo. Asimismo se establece como método para el mismo evento sobre el botón AbrirBtn de la barra de herramientas. Si el fichero actual ha sido modificado se pide al usuario de la aplicación que lo guarde antes de abrir otro para evitar la pérdida de cambios, cuya existencia se detecta a través de la propiedad *Modified* del componente de edición. Si se comprueba la presencia de cambios en el fichero se llama al método SalvarCambios, definido como función, que pregunta al usuario si desea guardar los cambios, no hacerlo o cancelar la operación de apertura de un nuevo fichero. Si el usuario selecciona una de las dos primeras opciones SalvarCambios devuelve el valor lógico verdadero.

Se comprueba el tipo de fichero que contiene el componente de edición. Si es de tipo *netlist* se inhabilita la opción de menú ReSim1, se asigna el valor falso al indicador *valido* y se llama al método Liberar, tal como se realiza en los métodos Guard1Click y GComo1Click.

Se utiliza el cuadro de diálogo estándar de apertura de ficheros OpenDialog1. Se ejecuta el método *Execute* del mismo y su propiedad *FileName* recoge el nombre del archivo que el usuario selecciona, el cual se asigna al campo NombreFich.

El método *LoadFromFile* de la clase *TStrings*, a la cual pertenece la propiedad *Lines* del elemento de edición, se aplica para cargar el archivo. Esta acción supone que la

propiedad *Modified* del editor reciba el valor verdadero, con lo que se le debe restituir el valor falso.

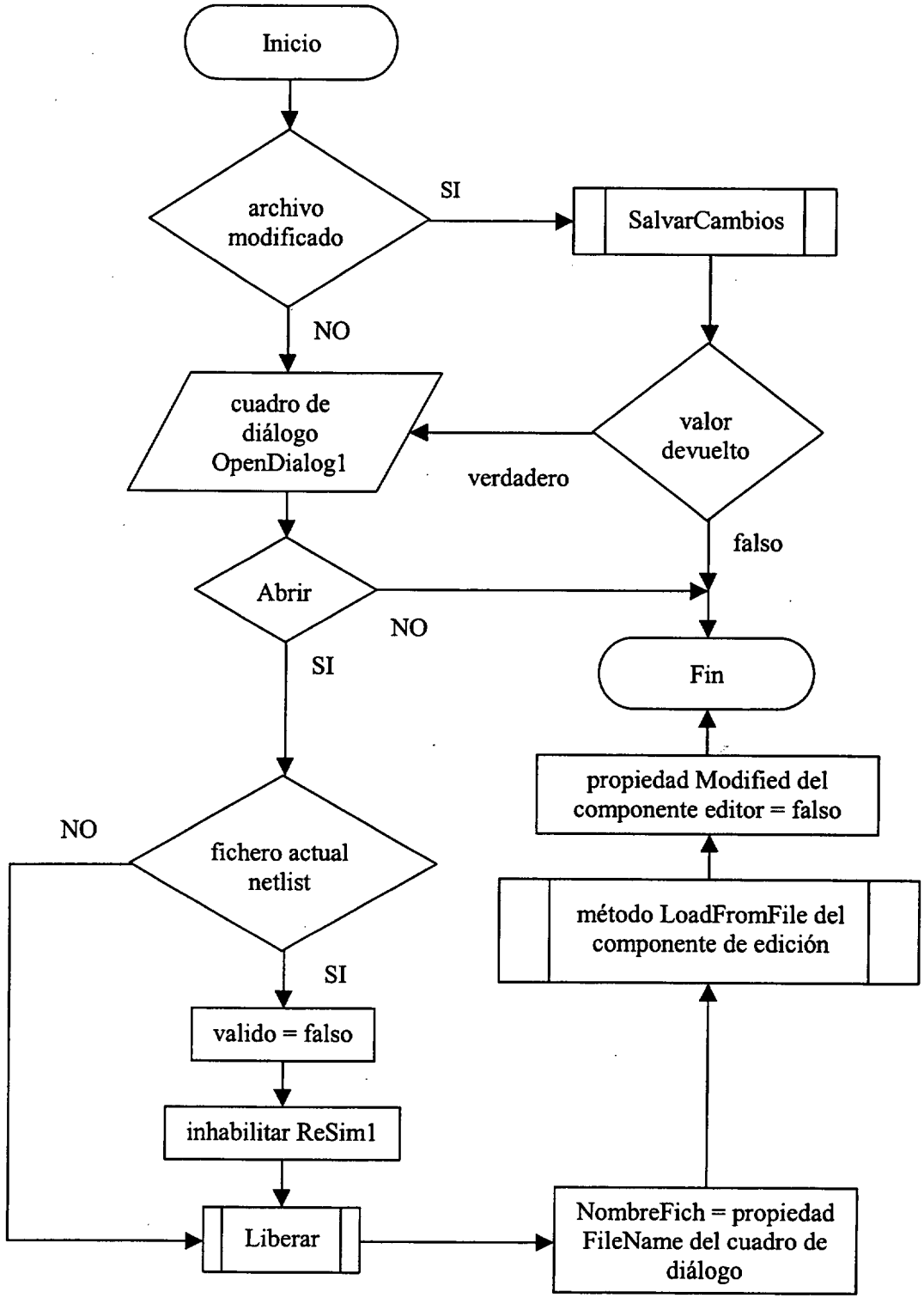


Figura 4. 49. Diagrama de flujos del método Abr1Click.

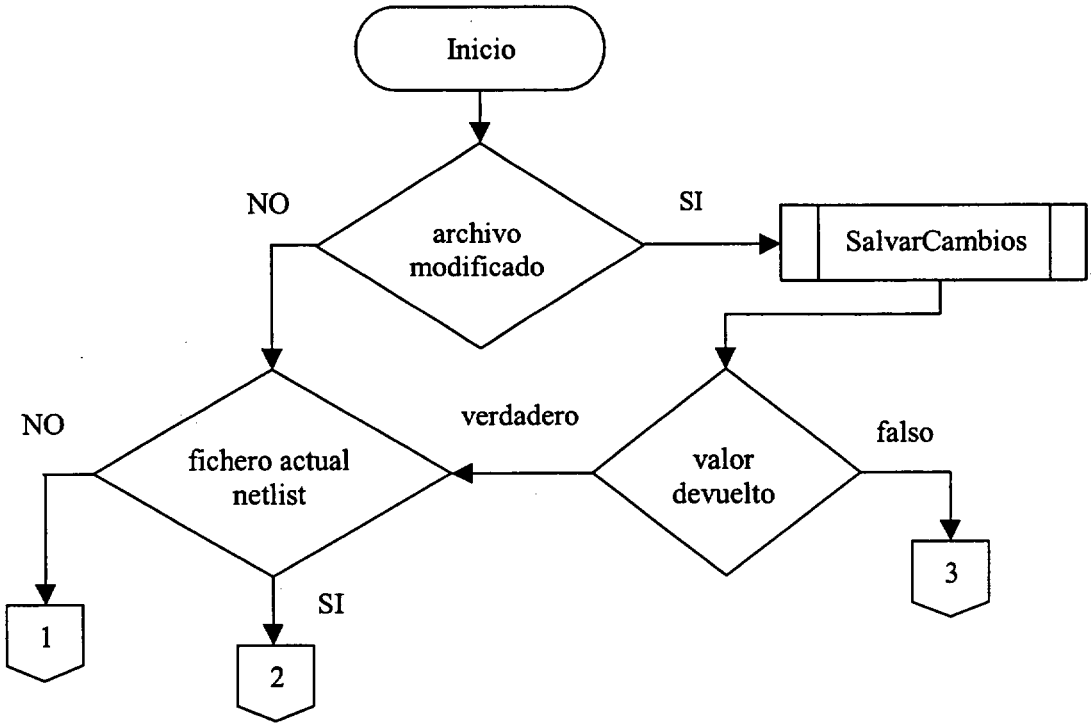
4.2.1.6.22. Método NuevNetClick.

Método que maneja el evento OnClick del elemento de menú NuevNet, Nuevo, del menú desplegable Archivo. Se asigna como gestor del mismo evento sobre el botón NuevoBtn de la barra de herramientas.

El método crea un nuevo archivo. Comprueba si el texto presente en el componente de edición ha sido modificado, llamando en ese caso al método SalvarCambios, que pregunta al usuario si quiere guardar los cambios, desecharlos o no realizar la operación.

Si se confirma la creación de un nuevo archivo se comprueba que el fichero actual sea de tipo *netlist*. Si es así tienen lugar las mismas acciones que en los métodos Abr1Click, Guard1Click y GComo1Click, es decir, se asigna falso al campo *valido* y a la propiedad *Enabled* del elemento de menú ReSim1 inhabilitándolo, y se llama al método Liberar.

Se inicializan el campo NombreFich y el componente de edición aplicando el valor de cadena vacía, en el caso de elemento editor a su propiedad *Text*.



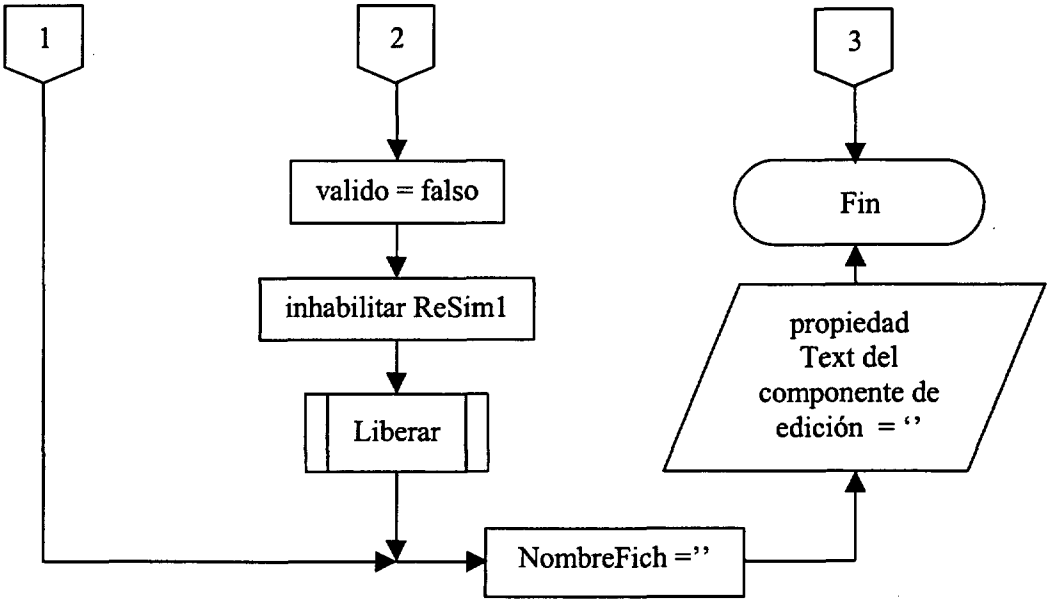


Figura 4. 50. Diagrama de flujos del método NuevNetClick.

4.2.1.6.23. Método NuevEntClick.

Este método se ha asociado al evento OnClick del comando NuevEnt, Nuevo fichero de entradas, del menú desplegable Archivo y también al citado evento del botón NuevEntBtn de la barra de herramientas de la aplicación.

Crea un nuevo fichero de entradas realizando las mismas operaciones que el método NuevNetClick. Se comprueba si el archivo actual ha sido modificado y se solicita al usuario que salve los cambios a través del método SalvarCambios. Si progresa la creación del nuevo fichero se comprueba que el archivo actual sea de tipo *netlist* y se procede a la inicialización de los campos de la ficha relacionados con este tipo de fichero, así como a la inhabilitación del elemento de menú ReSim1, perteneciente al menú desplegable Simular.

Al campo NombreFich, que contiene el nombre del fichero actual, y a la propiedad *Text* del elemento de edición se les asigna el valor cadena vacía a modo de inicialización de ambos.

En la primera línea del componente de edición se escribe en forma de comentario, texto precedido por el carácter '*', el tipo de información que debe contener con objeto de orientar al usuario de la aplicación. La escritura se realiza asignando a la línea 0 del editor el texto citado utilizando la propiedad *Lines* del mismo, que permite el tratamiento del texto en líneas individuales.

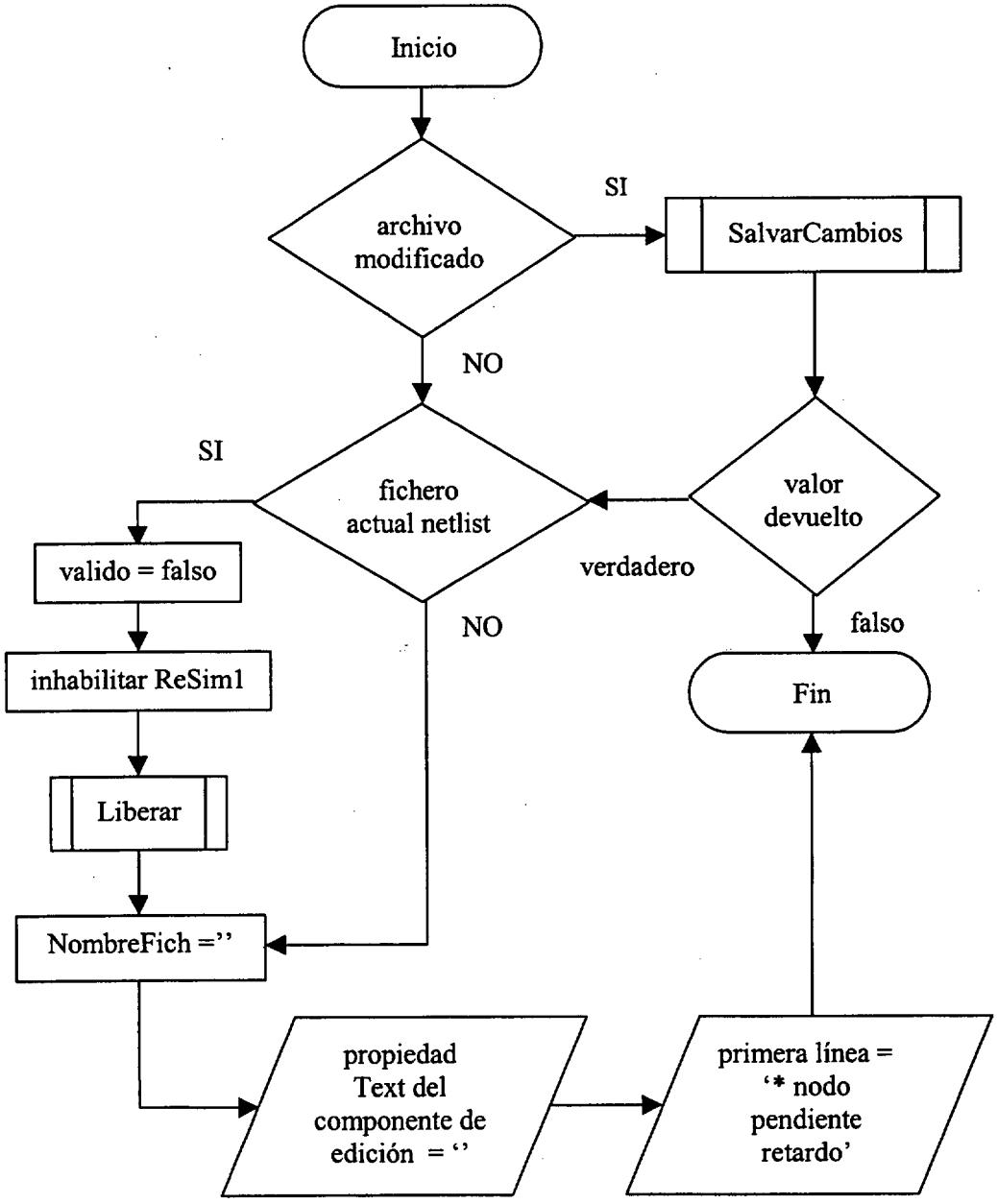


Figura 4. 51. Diagrama de flujos del método NuevEntClick.

4.2.1.6.24. Método SimRet1Click.

Es el método gestor del evento OnClick del elemento SimRet1, Simular retardo, del menú desplegable Simular. SimRet1 es el comando que el usuario de la aplicación ha de seleccionar para simular el fichero *netlist* presente en el componente de edición.

Se comprueba que el componente de edición contiene texto comparando el valor cadena vacía con el contenido de la propiedad *Text* del mismo. Si hay texto se prosigue con el proceso de simulación. Si no es así se informa al usuario de esa circunstancia, para lo que se dispone de un cuadro de mensaje predefinido. Se ha utilizado una llamada a *MessageDlg* que permite la personalización del cuadro agregando botones y fijando el tipo de cuadro de mensaje por asignación de valores a distintos parámetros del mismo. Se ha definido el cuadro como cuadro de mensaje de error al asignar al parámetro que define su tipo el valor *mtError*. Se ha dispuesto un botón de tipo *mbOK*, Aceptar.

Se verifica si el fichero ha sido modificado. Si es así se llama al método Liberar y el indicador *valido* recibe el valor lógico falso. La modificación del *netlist* supone que la lista enlazada que haya podido representarlo hasta ese momento deje de tener validez. Si se trata de la primera simulación sobre el *netlist* la llamada al método Liberar no tiene ningún efecto toda vez que no se ha constituido la lista enlazada, y el campo *valido* mantiene el estado que tenía, valor falso.

Si el indicador *valido* contiene el valor falso se inhabilita el elemento de menú ReSim1. Para crear la lista enlazada que representa al circuito se utilizan los procedimientos ObDatNet y ListNetlist declarados en la unidad sim4. Los punteros PlistaNodos y FlistaNodos, de tipo PunteroPuerta, también declarado en la unidad sim4, soportan la lista. Mediante un bucle se extrae cada línea del texto contenido en el componente de edición utilizando la propiedad *Lines* del mismo, que permite el acceso individual línea a línea. Esta cadena de caracteres se pasa como parámetro al procedimiento ObDatNet, que comprueba si se trata de una línea de información o de comentarios. En el primer caso extrae de la línea los datos que ésta contiene: tipo de puerta, nodos de entrada, nodo de salida y valor de la relación de aspecto; y genera un código de error comprobante de la validez de dichos datos en cuanto a su formato. Si no hay error la información de la puerta se incorpora a la lista enlazada mediante llamada al

procedimiento ListNetlist. Las sentencias presentes en el cuerpo del bucle se repiten hasta que se haya tratado todo el texto contenido en el elemento de edición o se detecte error en el mismo.

Si existe error en los datos del *netlist* se llama al método Liberar con objeto de eliminar los elementos, ya no válidos, incorporados a la lista enlazada. Se informa al usuario de dicho error mediante un cuadro de mensaje de error, de tipo *MessageDlg*, cuyo texto refleja la causa del error cometido.

Si no hay error el indicador *valido* adquiere el valor verdadero, indicando de esta forma que la simulación puede realizarse al ajustarse el *netlist* al formato establecido. Se llama al procedimiento CalcFanout, declarado en la unidad sim4, que calcula el fan-out según el concepto explicado en el capítulo 2. Se ha situado esta llamada en este bloque de código dedicado a la adquisición de los datos a emplear, y no en el siguiente, que realmente ejecuta la simulación, debido a que una vez validado el *netlist* el cómputo que realiza el procedimiento se mantiene invariable para todos los ficheros de entradas que se deseen simular sobre el mismo.

En el caso de que el campo *valido* contenga el valor lógico verdadero se muestra al usuario de la aplicación el cuadro de diálogo DatSim, definido en la unidad cuadro_sim, en llamada modal. Se emplea este cuadro de diálogo para que el usuario seleccione el fichero de entradas (*.net) que desea simular sobre el *netlist* y escoja el fichero con los parámetros de modelado (*.dat) que quiere sean aplicados en la simulación. Para la evaluación de los tiempos de propagación el usuario cuenta con la posibilidad de introducir nuevas ecuaciones desde teclado. Dos componentes de edición situados en el cuadro de diálogo permiten al usuario la entrada de las ecuaciones externas. La ficha se cierra cuando el usuario pulsa uno de los dos botones que se han dispuesto sobre ella, y el valor de retorno de la misma, *ModalResult*, se usa para determinar si continúa, se pulsa el botón Aceptar, con valor de retorno *mrOK,* o se aborta la simulación en curso, pulsando la opción Cancelar.

Antes de proseguir con el proceso de simulación el método verifica que el usuario haya especificado ambos ficheros. En caso afirmativo también se comprueba que éstos

existan. Si se produce un error en cualquiera de los dos casos se interrumpe la simulación y se informa al usuario mediante un cuadro de mensaje de error de tipo *MessageDlg*.

Se inhabilita el comando de menú ReSim1 del desplegable Simular, que sólo se habilitará si concluye con éxito la simulación. Se comprueba si se ha realizado previamente otra simulación sobre el *netlist*. Se hace verificando el valor que presenta el campo pendiente del primer elemento de la lista enlazada. Si es distinto de cero indica que se ha ejecutado una simulación anteriormente. En este caso se llama a los procedimientos InicRetardos y BorrarEntradas, ambos declarados en la unidad sim4. InicRetardos inicializa a cero los campos tp, respcambio y pendiente de cada elemento de la lista enlazada para realizar la simulación. De esta forma se eliminan los resultados de la simulación anterior sobre el *netlist*. BorrarEntradas elimina de la lista enlazada los elementos que corresponden a las entradas de la simulación anterior.

El procedimiento LeerEntradas, declarado en la unidad sim4, extrae la información contenida en el fichero de entradas y la incorpora a la lista enlazada que representa al *netlist*. Dicho procedimiento genera un código de error correspondiente al contenido del fichero.

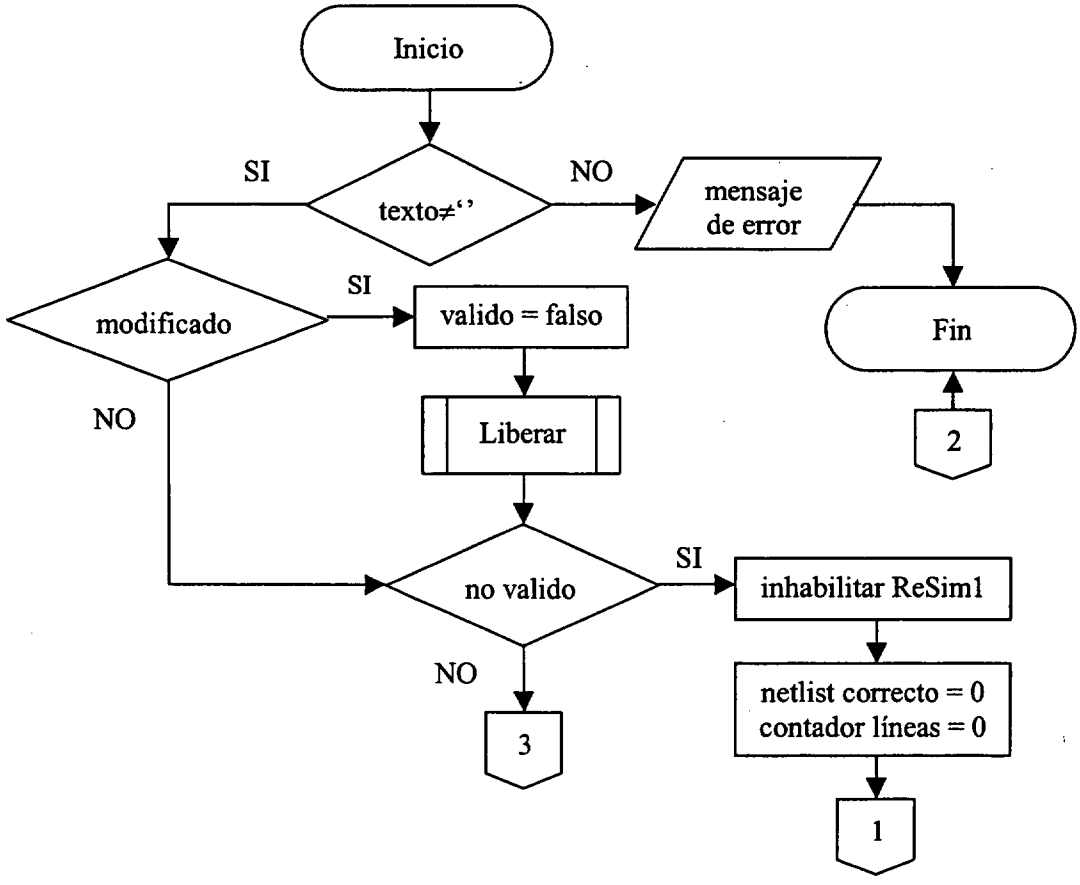
Si se detecta error en el archivo se informa al usuario mediante un cuadro de mensaje de error de tipo *MessageDlg*, y se llama al procedimiento BorrarEntradas, declarado en la unidad sim4, toda vez que la información del fichero carece de validez.

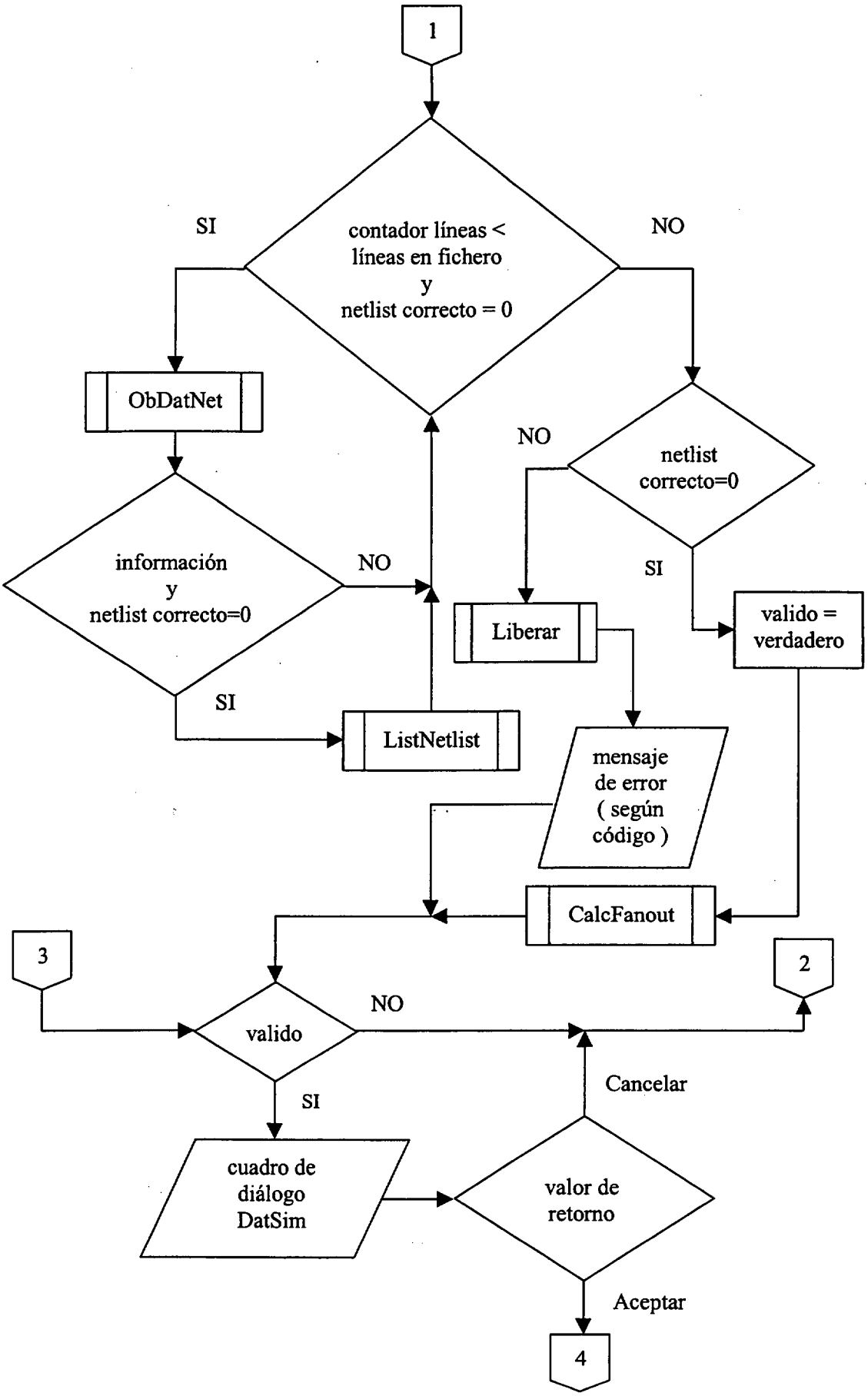
Una vez que la lista enlazada contiene los datos definidos en cada archivo se comprueba que en el fichero de entradas se haya especificado una señal para cada nodo de entrada al circuito, condición necesaria para poder desarrollar la simulación. Para ello se llama a la función ConexCorrecta, definida en la unidad sim4, que devuelve un valor de tipo lógico. Esta función también detecta errores en la numeración de los nodos, que pudieran dejar desconectada alguna de las puertas definidas en el *netlist*. Si se detecta error se detiene el proceso de simulación iniciado. Se llama al método Liberar que elimina la lista enlazada y se asigna el valor falso al indicador *valido*. A través de un cuadro de mensaje de error de tipo *MessageDlg* se informa al usuario de la existencia del error indicando la puerta que lo sufre mediante su nodo de salida.

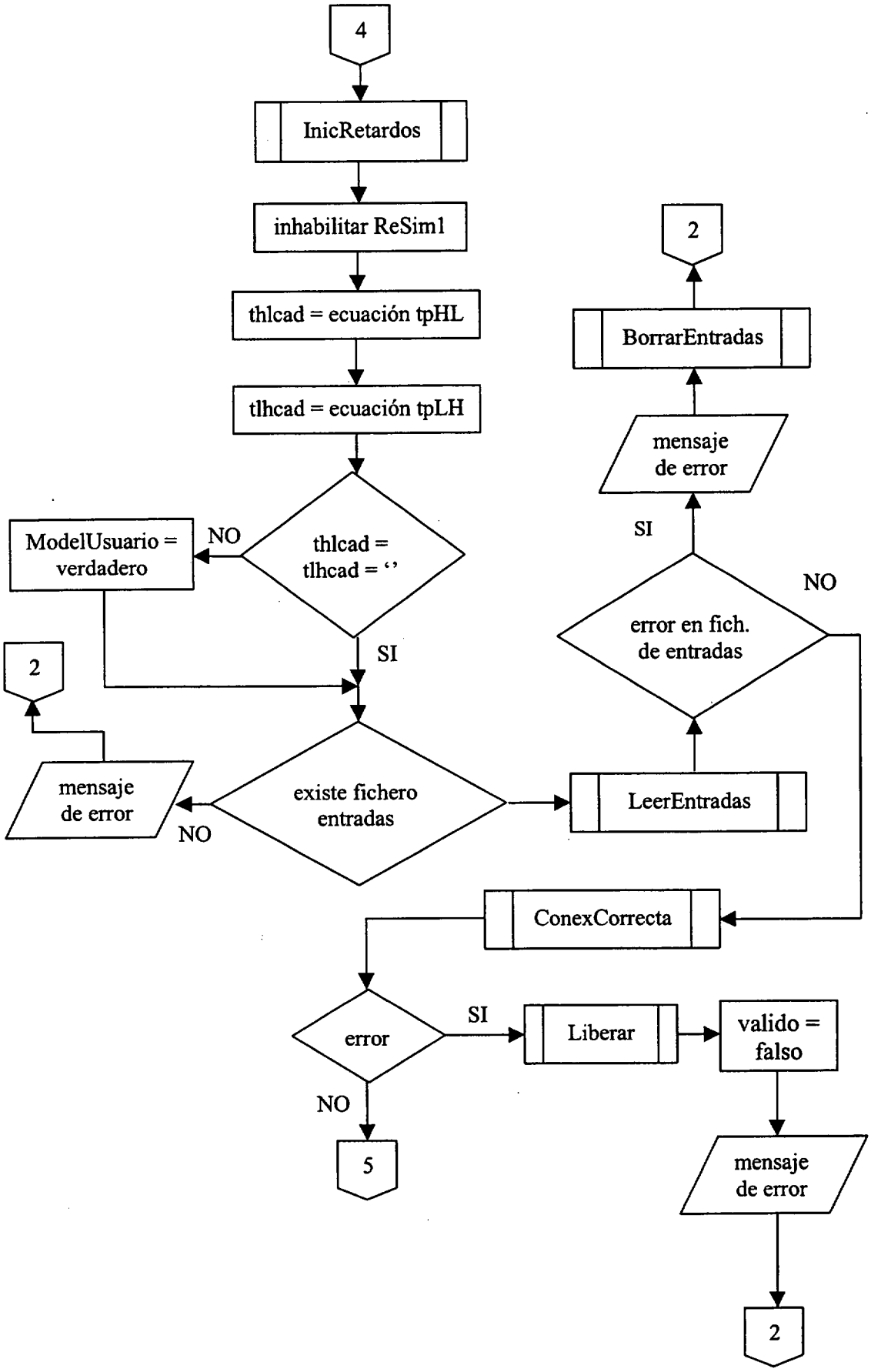
Se llama al procedimiento CalcularAtributos, declarado en la unidad sim4, que realiza la simulación del *netlist*. Se le pasan como parámetros los punteros PlistaNodos y FlistaNodos que soportan la lista enlazada y la variable ArchTecnolog con el nombre del fichero de la tecnología a emplear. También recibe como parámetros una variable de tipo lógico, ModelUsuario, que determina qué ecuaciones se utilizan y dos variables locales de tipo cadena de caracteres, thlcad y tlhcad, que contienen, en su caso, las ecuaciones introducidas desde teclado. ModelUsuario presenta el valor lógico verdadero si se ha definido al menos una expresión de modelado externa.

Tras la simulación se habilita el elemento de menú ReSim1, que permite al usuario consultar los resultados.

Finalmente, se informa al usuario de la finalización del proceso de simulación mediante un cuadro de mensaje. Se llama a *MessageBox*, que desarrolla un cuadro de mensaje en el que se puede escoger el símbolo que acompaña al mensaje y el título de la ventana. Se ha utilizado el símbolo de exclamación, *MB_ICONEXCLAMATION*, y se ha incorporado un botón de tipo *MB_OK*, Aceptar.







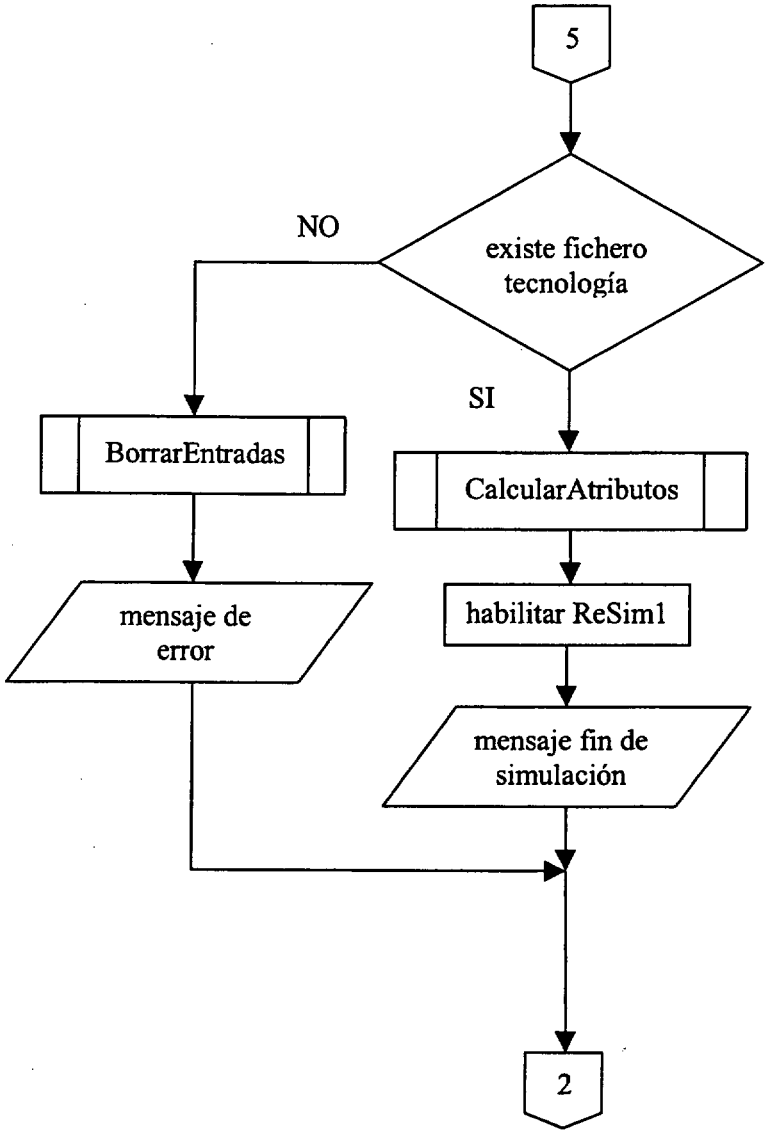


Figura 4. 52. Diagrama de flujos del método SimRet1Click.

4.2.1.6.25. Método PrepararLista.

Método incorporado a la clase que describe la ficha. Se utiliza para inicializar el contenido del cuadro de lista, ListaNodos, de la ficha Resultado, declarada en la unidad res4, antes de que ésta se muestre.

Entre llamadas a la ficha, ésta no se destruye, sino que se oculta. Por esta razón el cuadro de lista retiene los elementos que contiene, que corresponden a los nodos de la última simulación, y se precisa eliminarlos e incorporar los identificadores de los nodos correspondientes a la nueva simulación.

Para eliminar los elementos del cuadro de lista se emplea el método *Delete* de la propiedad *Items* del cuadro de lista, siendo esta propiedad de la clase *TStrings*. Dicho método requiere el índice, de tipo entero, del elemento a borrar. Se ha fijado el valor cero como índice, con lo que se borra el elemento que secuencialmente ocupa el primer lugar en el cuadro de lista.

Los nuevos elementos se añaden al cuadro de lista aplicando el método *Add* de la propiedad *Items*. El citado método agrega al final de la lista la cadena de caracteres que se le pasa como parámetro. Para ello a la propiedad *Sorted* del cuadro de lista se asigna el valor falso. Esta cadena corresponde al identificador del nodo de salida de una puerta del *netlist*, convertido a dicho formato por el método predefinido *IntToStr*.

Este identificador de nodo corresponde al contenido del campo nodo_s de un elemento de la lista enlazada que representa al *netlist*. Se extraen dichos valores realizando un recorrido completo de la lista enlazada, con lo cual se añaden de forma ordenada al cuadro de lista. Los nodos que corresponden a entradas al circuito, identificados porque sus campos nodoe1 y nodoe2 valen cero, no se incluyen en el cuadro de lista.

4.2.1.6.26. Método ReSim1Click.

Es el método gestor del evento OnClick del elemento ReSim1, Resultados de la simulación, del menú desplegable Simular. Inicializa distintos campos y propiedades de componentes pertenecientes a la ficha Resultado, asociada a la unidad res4, y la muestra en forma modal. A través de esta ficha el usuario de la aplicación puede consultar los resultados de la simulación efectuada.

La ficha se crea de forma automática por el archivo de proyecto, con lo que el evento OnCreate sólo se produce una vez. La inicialización de los campos y componentes debe realizarse cada vez que se muestra la ficha, no solamente como respuesta a la creación de la misma. Por este motivo se realizan dichas operaciones en el método ReSim1Click.

Este método asigna las direcciones a que apuntan PlistaNodos y FlistaNodos a los punteros PrinclistaN y FinlistaN respectivamente, con lo que soportan la lista enlazada que representa al *netlist*. PrinclistaN y FinlistaN, de tipo PunteroPuerta, pertenecen a la ficha Resultado.

Se llama al método PrepararLista que establece el contenido del cuadro de lista ListaNodos de la ficha asignando los identificadores de los nodos de salida de las puertas que constituyen el circuito como elementos de dicho componente.

Se fija como opción inicial por defecto del grupo de activación *Seleccion* la que permite al usuario acceder a la información del nodo crítico, casilla Nodo Crítico, asignando el valor cero a la propiedad *ItemIndex* del mismo.

El componente Etiqueta, de tipo *Label*, utilizado para mostrar el retardo se inicializa asignando una cadena vacía a su propiedad *Caption*. La propiedad *Text* del elemento de edición CaminoM, empleado para presentar el camino de nodos que origina el retardo en el nodo seleccionado por el usuario se inicializa también como cadena vacía.

Se llama al método inicchart de la clase TResultado, definido en la unidad res4, para inicializar el componente de gráficos de datos Chart1 de la ficha Resultado.

4.2.1.6.27. Método ParSPICE.

El método ParSPICE se añade a la ficha en forma de función. Devuelve un valor de tipo cadena de caracteres equivalente a la cadena que recibe como parámetro y que ha de representar a un número, después de multiplicar el valor recibido por una potencia de 10 para expresarla en diferentes unidades de medida. Esta operación equivale a desplazar el punto decimal y añadir o eliminar ceros en la cadena que representa al número en función del exponente. El desplazamiento a realizar se le especifica al método como parámetro de tipo entero.

El método se emplea en la definición del modelo de los transistores para SPICE a partir de los parámetros de modelado del fichero de la tecnología especificado por el usuario. Así el parámetro RS del modelo correspondiente al transistor de enriquecimiento es:

$$RS = 100 \cdot rse$$

El método ParSPICE recibe en este caso como parámetros la cadena con el valor de rse en el fichero de la tecnología y el valor 2 como desplazamiento.

Si el número al que representa la cadena es entero existen dos posibilidades según el desplazamiento especificado. Si éste es positivo se añaden al final de la cadena tantos ceros como indique el desplazamiento. Si el desplazamiento es negativo se compara con la longitud de la cadena, dándose los siguientes casos:

Si (longitud – desplazamiento) > 0, se inserta el punto decimal en la posición de la cadena (longitud – desplazamiento) + 1.

Si (longitud – desplazamiento) ≤ 0, se añaden al comienzo de la cadena tantos ceros como indique el valor resultante, y luego la cadena '0.'.

Si el número representado por la cadena tiene parte decimal y el desplazamiento es positivo:

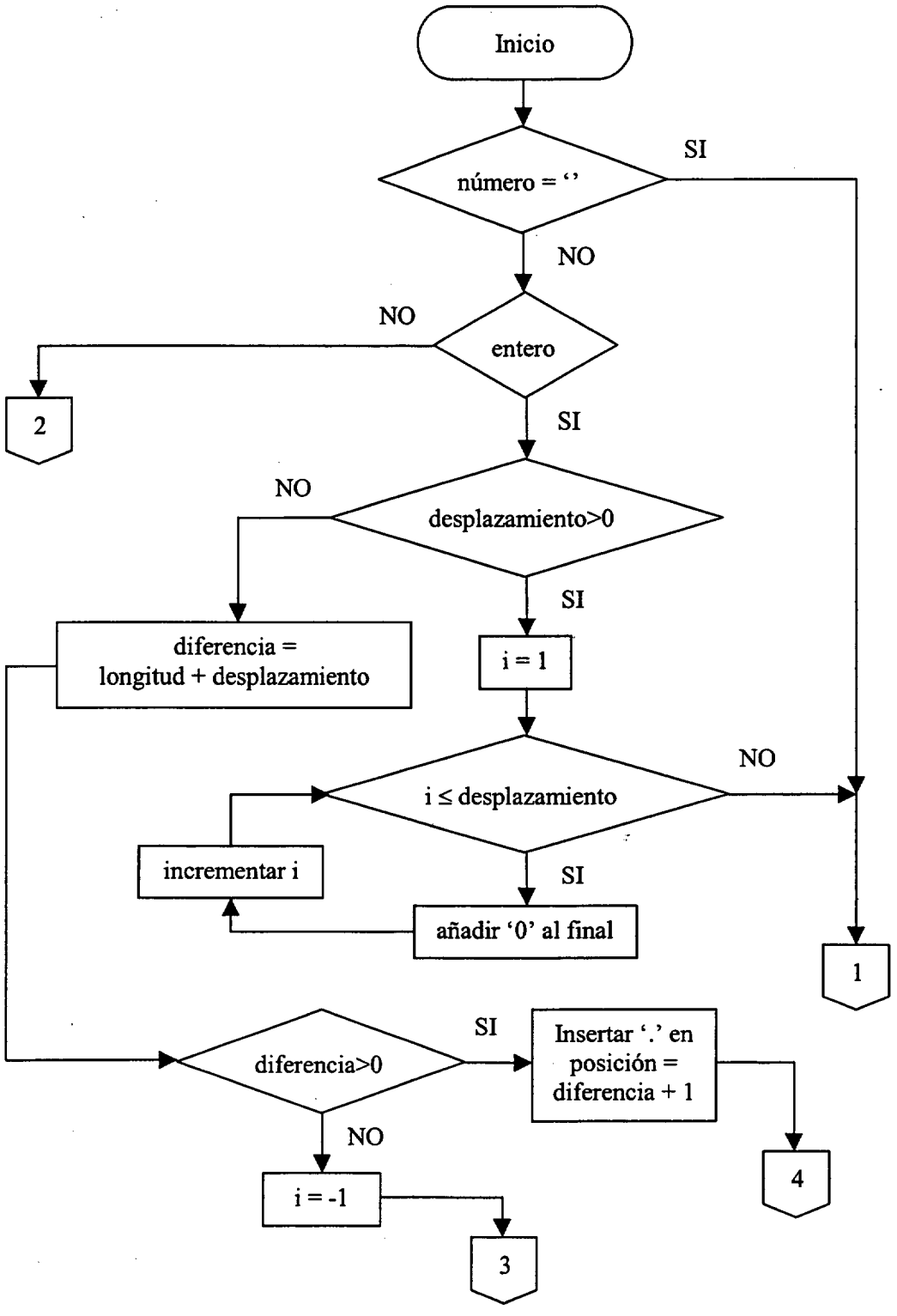
Si desplazamiento ≥ (longitud – posición punto decimal), se borra el punto decimal y se añaden tantos ceros a la derecha como mayor sea.

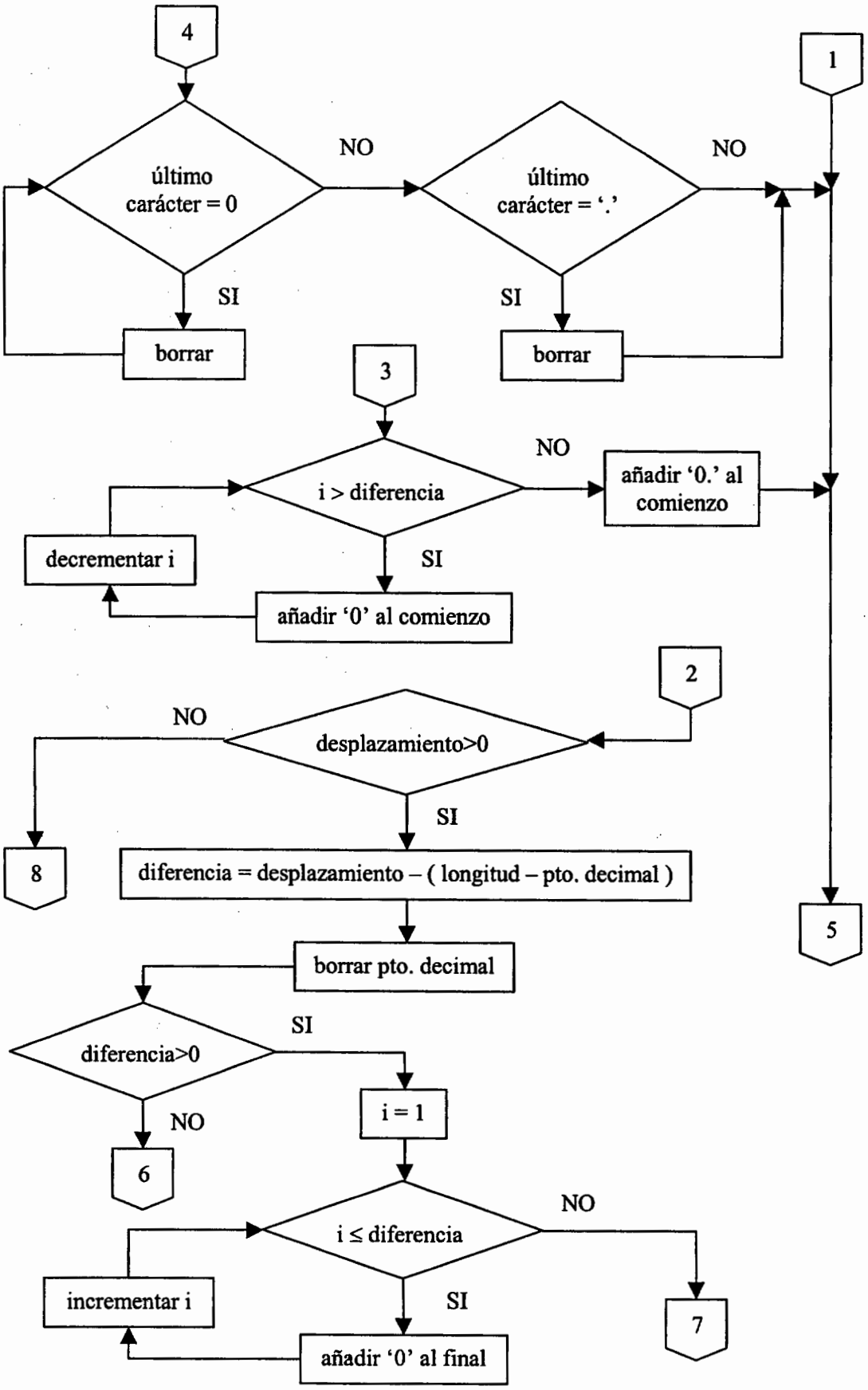
Si desplazamiento < (longitud – posición punto decimal), se sitúa el punto en la posición pto. decimal + desplazamiento.

Si el desplazamiento es negativo:

Si (posición punto decimal – desplazamiento) > 1, se sitúa el punto decimal en la posición que resulta de la resta.

Si (posición punto decimal – desplazamiento) ≤ 1, se añaden al inicio de la cadena tantos ceros como la diferencia + 1, y '0.', habiendo borrado el punto decimal.





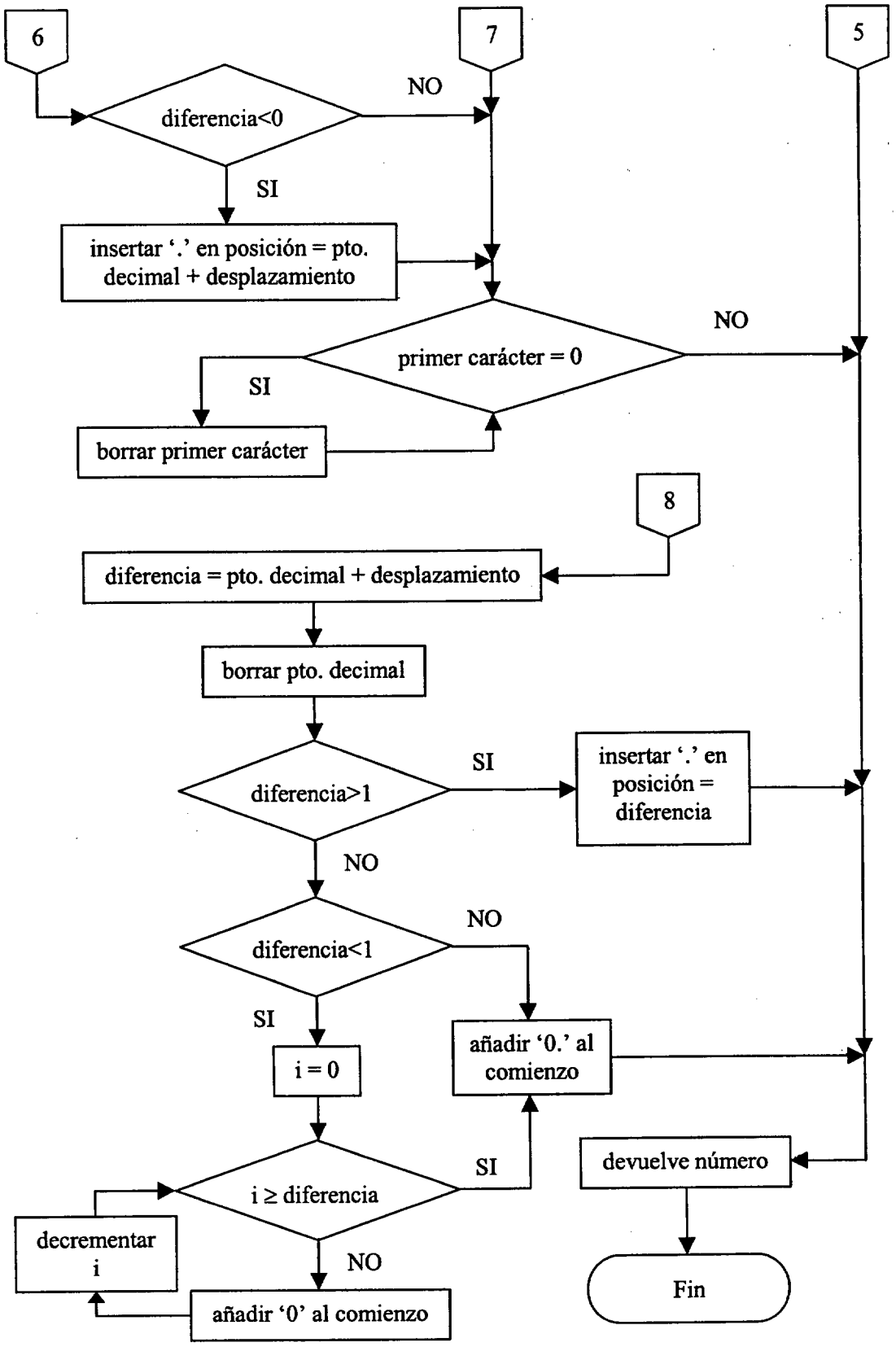


Figura 4. 53. Diagrama de flujos del método ParSPICE.

4.2.1.6.28. Método InModelo.

El método InModelo se ha agregado a la clase que describe a la ficha para añadir al fichero SPICE la información de los modelos de los transistores. Los parámetros que el modelo basado en el análisis de la sensibilidad considera sensibles se obtienen del fichero de la tecnología que el usuario haya seleccionado. Tanto el fichero de la tecnología como el fichero SPICE se pasan como parámetros al método. El fichero SPICE se abre mediante el procedimiento predefinido *Append*, que permite añadir información al final del archivo.

Los parámetros del fichero de la tecnología que se añaden al modelo de los transistores se extraen del mismo utilizando la función valorfich, declarada en la unidad sim4. Si el parámetro debe expresarse en diferentes unidades se llama a la función ParSPICE, que realiza la conversión de unidades necesaria.

4.2.1.6.29. Método Entradas.

El método Entradas se ha añadido a la clase que describe a la ficha. Inserta en el fichero SPICE la declaración de las entradas a simular sobre el circuito. Recibe como parámetro el nombre del fichero con las entradas (*.ent) que especifique el usuario, a partir del cual se generan las entradas SPICE.

Se lee cada línea del fichero hasta llegar a su fin, que se reconoce por la palabra clave end. Se desechan las líneas de comentarios, y de las de información se obtienen los datos correspondientes al nodo sobre el que se aplica la señal de entrada, la transición de la misma y, en su caso, el retardo que presenta. Para cada entrada al circuito se crea una entrada SPICE.

Si la entrada corresponde a un nivel lógico alto o bajo, la entrada SPICE creada contiene la palabra identificadora 'DC'. Si se trata de una entrada a nivel alto se le asigna el valor '1' y '0' si se encuentra a nivel bajo.

Si la entrada presenta una transición se crea una entrada SPICE de tipo Pwl, que permite definir una señal de tipo escalón. La entrada Pwl se define especificando el valor,

0 o 1, que presenta en los instantes de tiempo marcados en la figura 4.53. Los tiempos (a) y (b) se establecen a partir del retardo que presenta la entrada leída del fichero de entradas, sumando sendos valores temporales para definir el escalón. Se utiliza el procedimiento predefinido val para obtener los valores de retardo en formato numérico, y la función rcad, declarada en la unidad sim4, para convertir a cadena de caracteres los valores temporales obtenidos. A la última referencia de tiempo, (c), se le asigna el valor especificado por el usuario para la duración del análisis transitorio de la simulación SPICE.

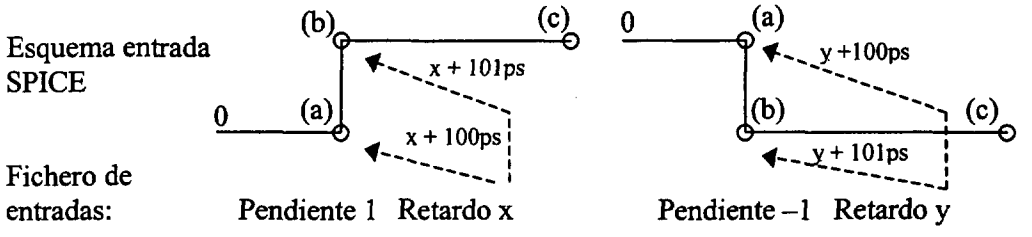


Figura 4. 54. Creación de entradas SPICE a partir de las del modelo.

La información de las entradas a simular se añade al fichero SPICE empleando los métodos propios del componente de edición de la ficha principal. El texto se inserta utilizando el método *Insert*, que pertenece a la propiedad *Lines* de clase *TStrings*, del editor. Este método inserta una cadena de caracteres en la posición que se especfica.

4.2.1.6.30. Método Eliminar.

El método Eliminar se añade a la clase para borrar la lista enlazada apuntada por un puntero de tipo PInfoS que recibe como parámetro. Los elementos que forman la lista se eliminan por la cabeza de la misma. Se utiliza el procedimiento predefinido *dispose* para borrar cada elemento. Un puntero auxiliar apunta al elemento a destruir. Al puntero recibido como parámetro, que soporta la lista enlazada, se le asigna la dirección del elemento de la lista que sigue al que se va a borrar para no perder el enlace que mantiene la lista. Cuando el puntero recibido alcance el valor nil se considera que la lista ha sido borrada.

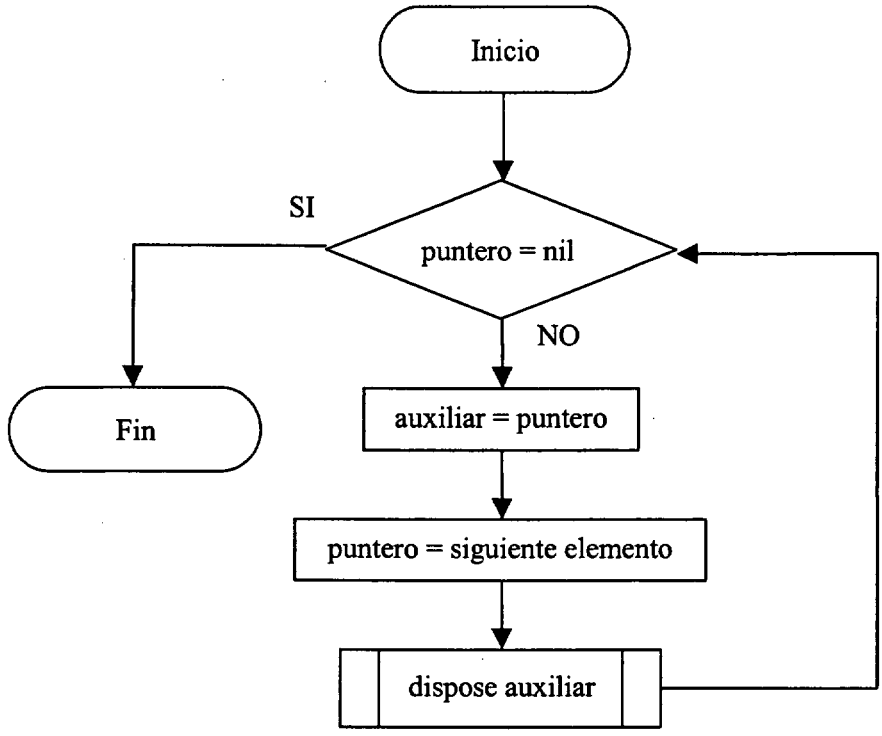


Figura 4. 55. Diagrama de flujos del método Eliminar.

4.2.1.6.31. Método InsertarBetayNombre.

El método InsertarBetayNombre, añadido a la clase que describe a la ficha, agrega a la lista enlazada apuntada por el puntero de tipo PInfoS que recibe como parámetro un nuevo elemento. El puntero apunta siempre al primer elemento de la lista y los elementos nuevos se insertan al cominezo de la lista.

El elemento a insertar se crea mediante el procedimiento estándar *new*, para lo que se usa un puntero auxiliar de tipo PInfoS. La información que se asigna al nuevo elemento corresponde al nombre con que se identifica a un tipo de subcircuito en un fichero SPICE, y al valor de la relación de aspecto, β, o de la anchura del transistor de enriquecimiento, W_E, según se esté traduciendo de SPICE al modelo, o viceversa, que define a ese tipo de subcircuito. Ambos datos se pasan al método como parámetros.

Si la lista está vacía el puntero apunta al nuevo elemento creado, a cuyo campo de enlace, next, se le asigna la constante nil. Si la lista contiene más elementos el campo next

del nuevo elemento apunta al primer elemento de la lista, y sólo entonces el puntero que sostiene la lista apuntará al nuevo elemento, que pasa a encabezar la lista enlazada.

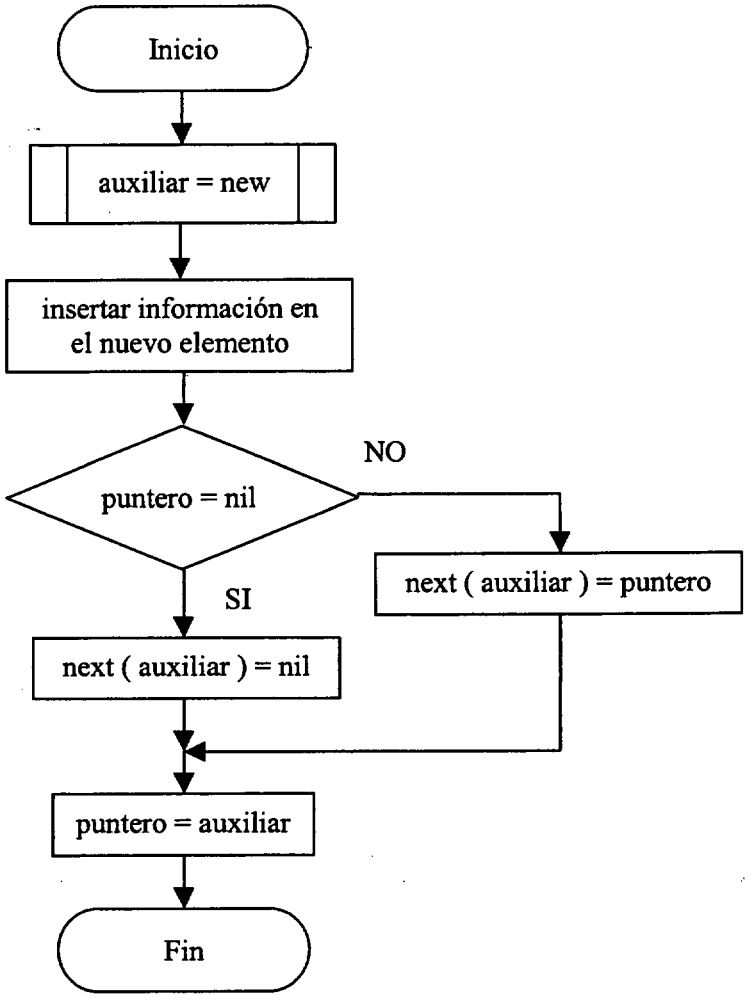


Figura 4. 56. Diagrama de flujos del método InsertarBetayNombre.

4.2.1.6.32. Función SubNombre.

La función SubNombre se define anidada al método Trad1Click. Devuelve una cadena de caracteres que corresponde al nombre con que se declara un tipo de subcircuito. Dicho nombre lo busca en la lista enlazada de elementos de tipo InfoS que contiene los nombres y los valores de W_E definidos para un tipo base de puerta, inversor o NOR2. El nombre identifica al subcircuito cuyo valor de la anchura del transistor de enriquecimiento la función recibe como parámetro. Mediante un puntero de tipo PInfoS se recorre la lista comparando el valor recibido, después de convertirlo a formato de cadena, con el

contenido del campo bet de cada elemento de la lista. Si ningún elemento contiene el valor buscado la función devuelve como resultado una cadena vacía.

4.2.1.6.33. Procedimiento Circuito.

El procedimiento Circuito, definido localmente al método Trad1Click, añade al fichero SPICE en creación la declaración del circuito a simular. Recibe como parámetros el fichero SPICE, y dos punteros de tipo PInfoS, que soportan las listas que almacenan la información de los subcircuitos, siendo cada puntero para un tipo base de puerta, NOR2 e inversora. También recibe un parámetro de tipo lógico que verifica la validez de los datos introducidos por el usuario como información para el análisis transitorio. El circuito se declara definiendo las puertas que lo constituyen.

El usuario en la declaración de análisis transitorio precisa los nodos del circuito cuya simulación desea conocer. Si el último nodo que determina corresponde a una salida del circuito es necesario conectar como cargas varias puertas para poder realizar la simulación. En este caso se inserta una cadena de 3 inversores. Para no modificar las condiciones de simulación respecto al modelo se precisa que el fan-out, punto 2.4.1 de la memoria, calculado entre la puerta de salida y la carga sea 1. Por lo tanto, los inversores tendrán el mismo valor de relación de aspecto, β, que la puerta de salida. Se ha definido una variable de tipo lógico llamada cargas para determinar si se añade la cadena de inversores según el nodo a observar corresponda a una salida del circuito o no.

El contador contx, de tipo entero, enumera cada puerta que se añade a la declaración conforme al formato SPICE. Cada puerta se define especificando los nodos a que se conecta y el tipo de subcircuito a que pertenece.

Para obtener dichos datos se lee cada línea del fichero *netlist* activo en el componente de edición de la ficha. Se accede a cada línea utilizando la propiedad *Lines*, de tipo *TStrings*, del elemento editor. Si la línea contiene comentarios se añade al archivo SPICE.

Si contiene datos, éstos se extraen de la cadena utilizando la función CopiaSubCad, declarada en la unidad sim4, y corresponden al tipo de puerta, los nodos a que se conecta y el valor de la relación de apecto. A partir de éste último se obtiene la anchura del transistor de enriquecimiento multiplicando su valor por cinco para definir un tipo de subcircuito. Sendas listas enlazadas, una para cada tipo base de puerta, contienen los valores obtenidos que definen cada subcircuito. Cada clase de subcircuito se identifica asignándole como nombre un valor numérico, que se obtiene mediante el incremento de un contador específico para cada tipo de puerta, que se añade a la cadena formada por las siglas del tipo básico de puerta, INV o NOR2.

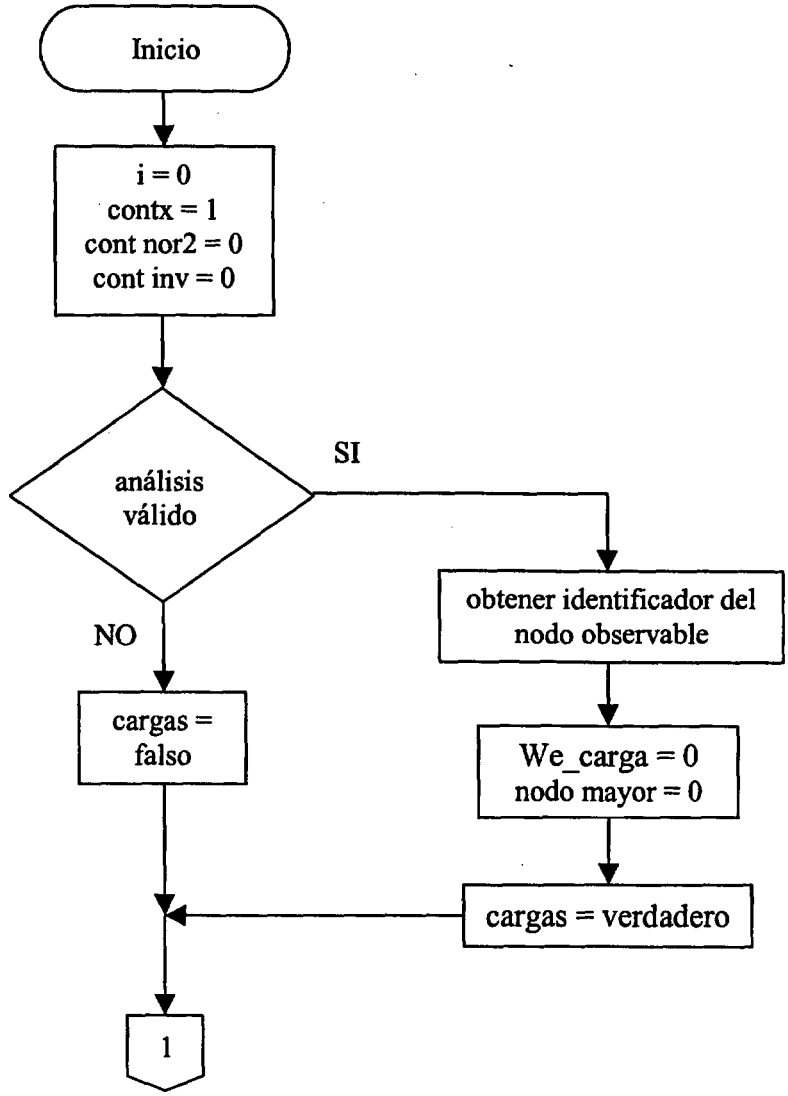
Cuando se añade la declaración de una puerta al fichero SPICE se recorre la correspondiente lista enlazada, en función del tipo base de la misma, comprobando si el valor de la anchura del transistor de enriquecimiento, W_E, de la puerta a incorporar es uno de sus elementos. Esta labor la realiza la función SubNombre, que devuelve el nombre con que se declara el subcircuito que presenta ese valor de W_E. Si no lo encuentra se inserta en la lista como nuevo elemento llamando al método InsertarBetayNombre.

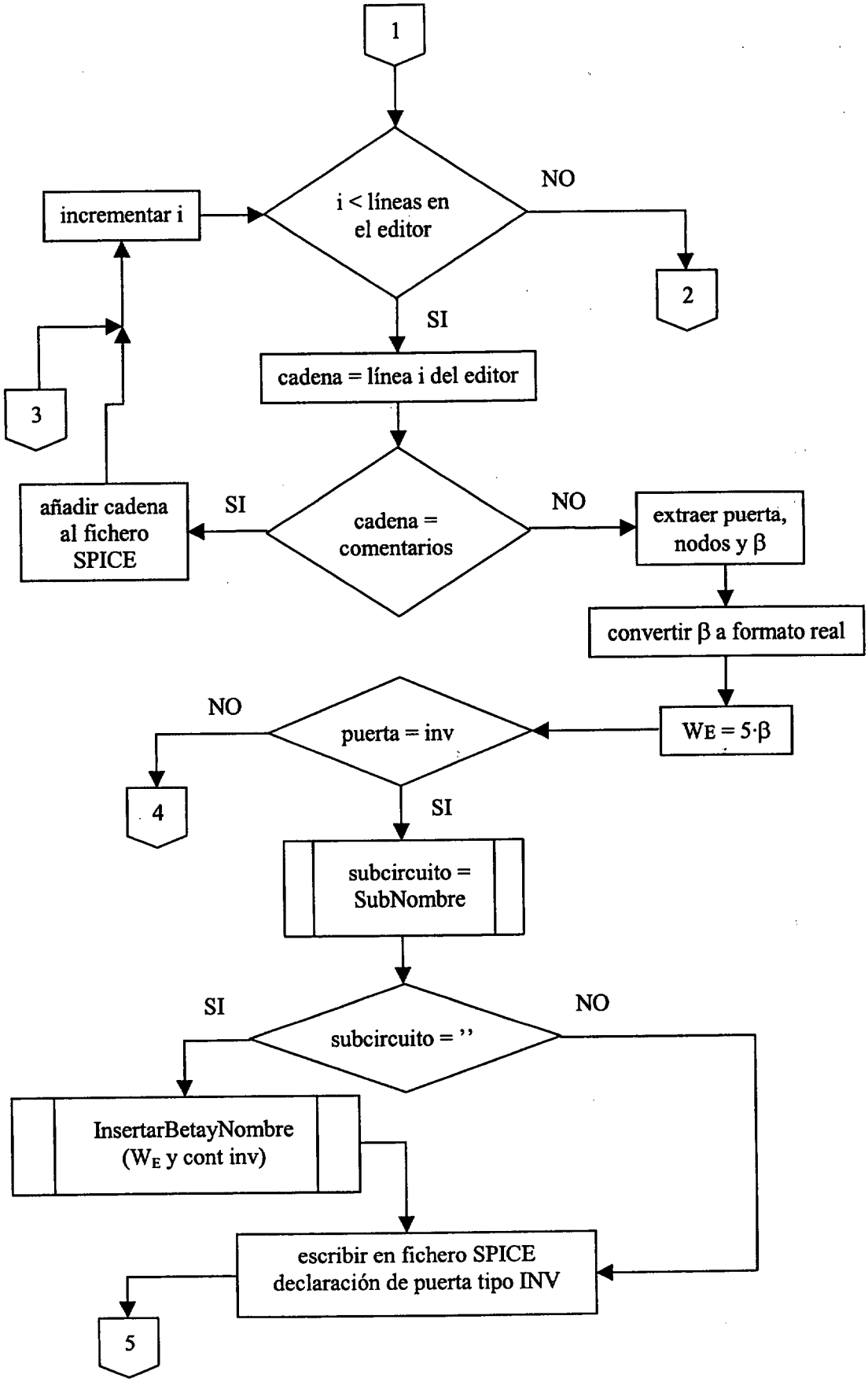
Una línea de fichero SPICE declarando una puerta consta de la numeración de la puerta, carácter 'X' seguido del valor del contador contx, los nodos de entrada y salida de la misma y el tipo de subcircuito a que pertenece. El carácter 'X' es el asignado para nombrar a los transistores HFETs en el fichero SPICE [4]. La línea se inserta utilizando el procedimiento predefinido *writeln* para la escritura en ficheros de texto.

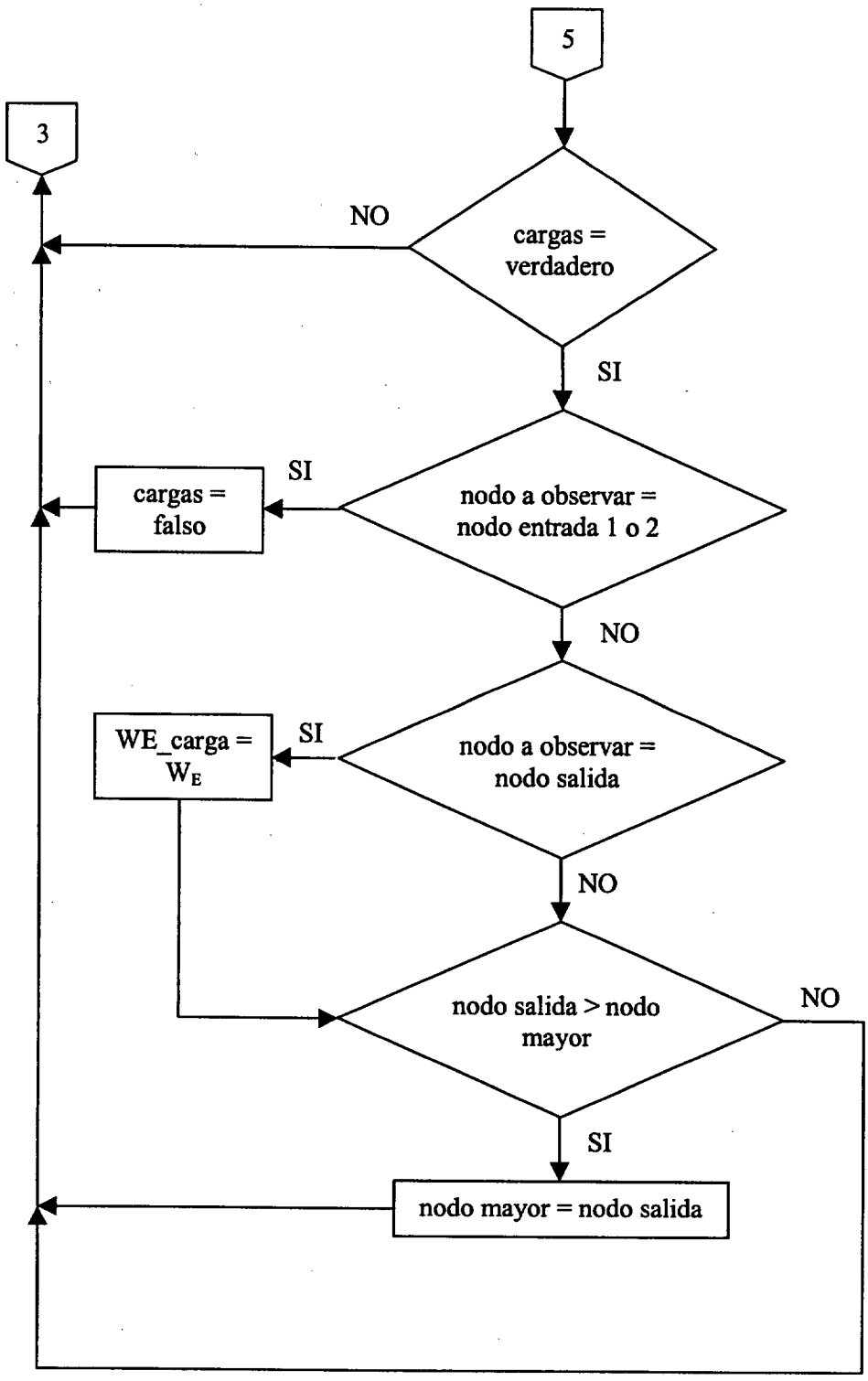
La varible lógica cargas recibe inicialmente el valor verdadero si la información de análisis introducida por el usuario es correcta. Siempre que mantenga dicho valor, para cada línea de información del fichero *netlist* se compara el nodo especificado con los nodos de entrada de la puerta para determinar si aquél corresponde a una salida del circuito. Si es igual se asigna el valor falso a la variable cargas para evitar incluir la cadena de inversores. Si es distinto se compara con el nodo de salida de la puerta para almacenar el valor de la anchura del transistor de enriquecimiento si corresponde a la selección del usuario. Se obtiene el valor más alto de nodo de salida del circuito para asignar como identificadores de los nodos de la cadena de inversores valores superiores al mismo.

Al concluir la lectura del fichero *netlist* y la declaración del circuito se añade la cadena de inversores al nodo especificado si la variable cargas presenta el valor verdadero. Se emplea la función SubNombre para buscar en la lista enlazada correspondiente a los inversores el nombre del subcircuito cuyo valor de W_E sea igual al del nodo a observar. Si no se encuentra el nodo seleccionado corresponde a la salida de una puerta NOR2, y se crea un nuevo tipo de subcircuito inversor con dicho valor de W_E. Mediante un bucle for se añaden los inversores incrementando el identificador del nodo mayor encontrado en el circuito.

El proceso de declaración de puertas concluye al finalizar la lectura completa del archivo *netlist*.







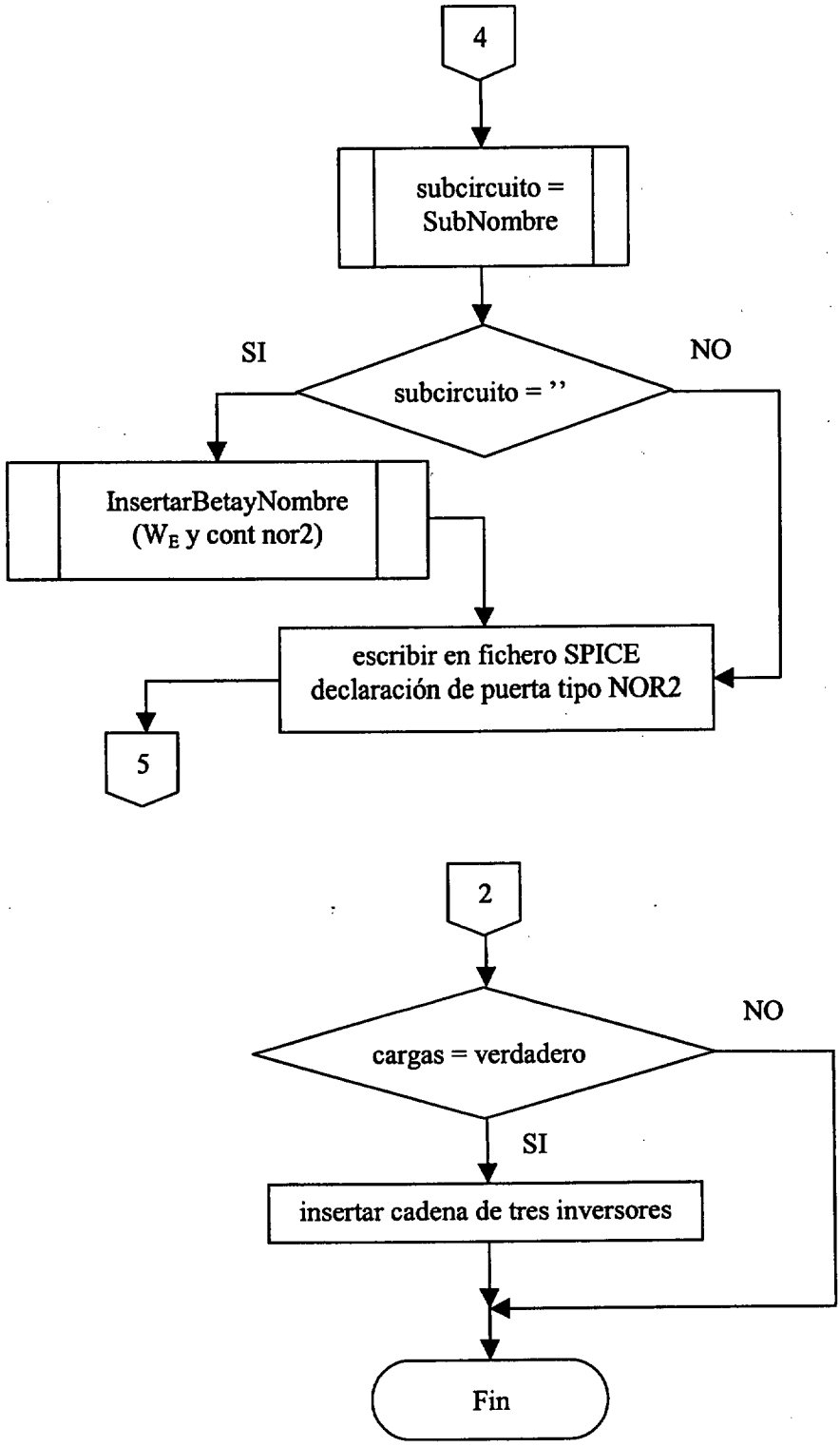


Figura 4. 57. Diagrama de flujos del procedimiento Circuito.

4.2.1.6.34. Método Trad1Click.

El método Trad1Click es el gestor del evento OnClick del elemento Trad1, Traducir a SPICE, del menú Simular. Convierte a formato SPICE el archivo *netlist* actual.

El fichero SPICE que se crea se guarda con el mismo nombre que el archivo *netlist* y se le asigna la extensión SPI. El nombre se obtiene del campo NombreFich de la ficha.

Para añadir la información de análisis al fichero SPICE, mediante la cual el usuario especifica los nodos del circuito cuya respuesta desea conocer de la simulación y los tiempos de paso y análisis transitorio, se dispone del cuadro de diálogo TAS asociado a la unidad cuadro_t. Se trata de una ficha secundaria caracterizada en forma de cuadro de diálogo que se muestra en forma modal. Antes de mostrarla se inicializan al valor cadena vacía los dos campos de edición, componentes de tipo *Edit*, que ofrece la ficha para la introducción de los datos de análisis por parte del ususario. El cuadro de diálogo se emplea también para especificar el fichero de entradas (*.ent) y el fichero de la tecnología (*.dat) necesarios para completar la información que precisa el fichero SPICE para realizar la simulación.El cuadro dispone de botones a los que se ha signado valor de retorno, *ModalResult*, que se comprueba para decidir si se realiza la traducción del fichero a SPICE. Si el valor de retorno es mrOK, botón Aceptar, se ejecuta, mientras que si se pulsa el botón Cancelar el proceso de traducción queda abortado.

El código desarrollado verifica que los ficheros de entradas y de la tecnología seleccionados existen. Si no es así se informa al usuario del error cometido mediante un cuadro de mensaje estándar de tipo *MessageDlg*, y se detiene la creación del archivo SPICE.

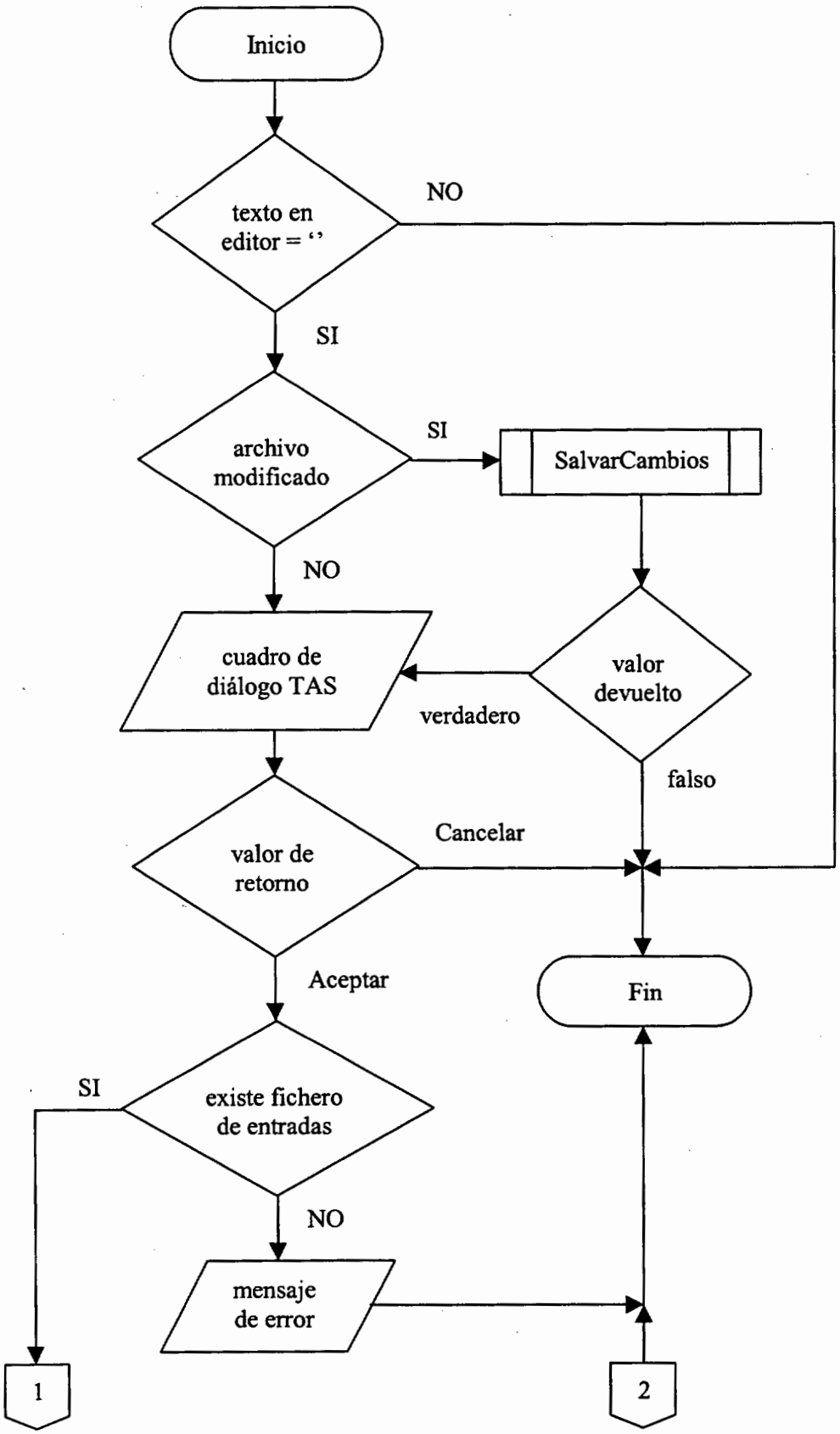
Se llama a los métodos InModelo y Entradas, y al procedimiento Circuito para incluir en el fichero SPICE la información de los modelos de los transistores, las entradas a simular y el circuito. El modelo de los transistores se extrae de el fichero de la tecnología, las entradas a simular se obtienen del fichero de entradas, y el fichero *netlist* contiene la definición del circuito.

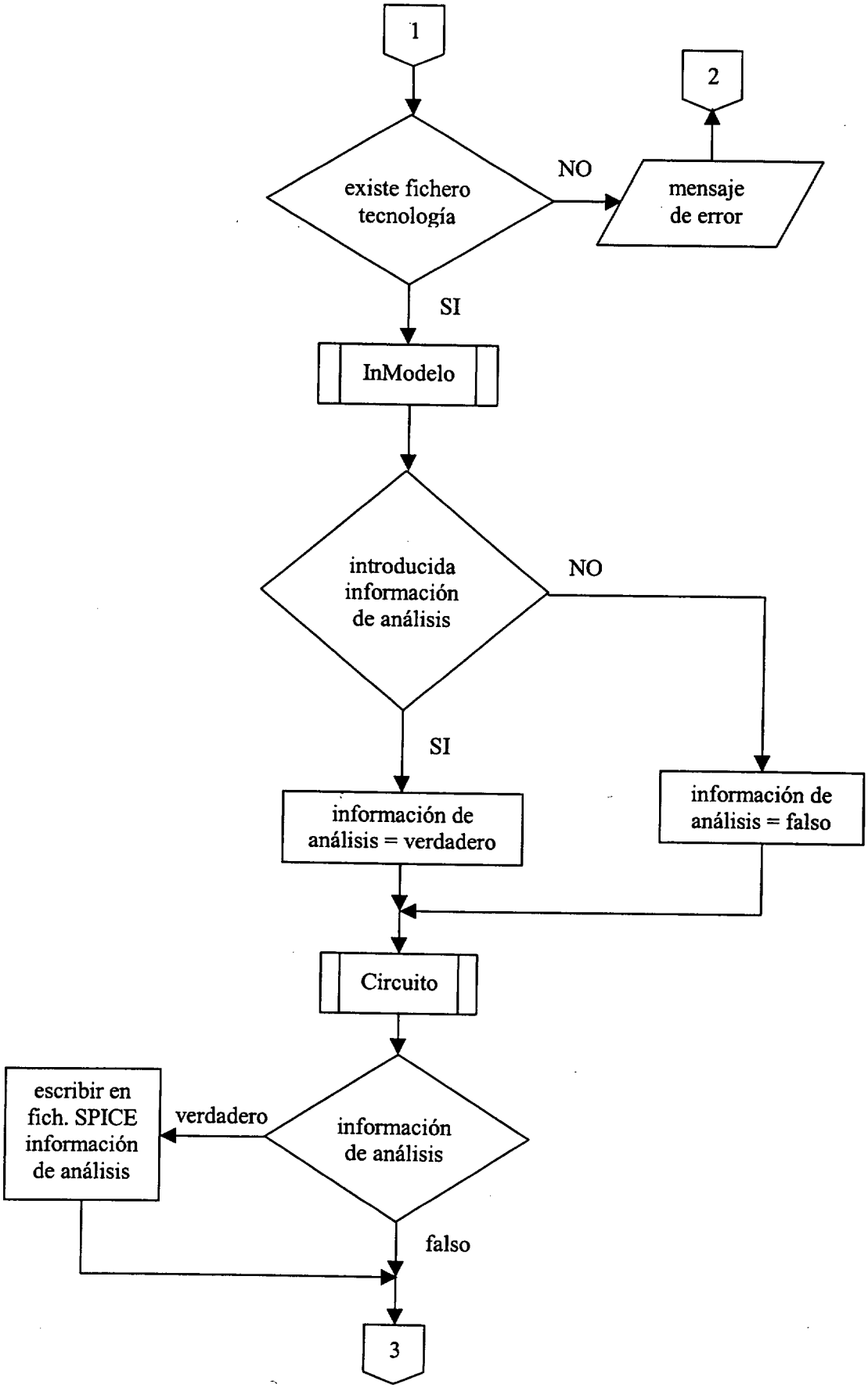
Se ha definido un tipo de datos, lista enlazada de elementos de tipo InfoS, específico para contener los diversos valores de WE que presentan las puertas del circuito. Se usa esta información para la definición de los subcircuitos, que se realiza con posterioridad a la declaración del circuito. Los diferentes valores de WE se memorizan a medida que se declaran las puertas en la ejecución del procedimiento Circuito, creando una lista enlazada para cada tipo base de puerta, inveror y NOR2. Si se generase primero la información de los subcircuitos habría que realizar una segunda lectura del fichero *netlist* o disponer de estructuras de datos de mayor ocupación en memoria para almacenar los datos necesarios para especificar el cicuito.

El texto que define los subcircitos se incluye desde el componente de edición, mientras que el método InModelo y el procedimiento Circuito emplean los procedimientos predefinidos de tratamiento de ficheros. Para añadir el texto se utiliza el método *Insert* de la clase *TStrings* a la que pertenece la propiedad *Lines* del elemento de edición. Este método inserta una cadena de caracteres en la posición que se especfica, lo que no sería posible con los procedimientos de tratamiento de ficheros de que dispone el lenguaje. Se sitúa precediendo a la declaración del circuito.

En la definición de cada subcircuito se especifican las conexiones de los transistores que conforman la puerta y la anchura de los mismos. La anchura correspondiente al transistor de deplexión se fija al valor 5, y la del transistor de enriquecimiento se obtiene de la lista enlazada correspondiente según sea el tipo básico de la puerta que se declara. También se especifica la tensión de alimentación, que se obtiene del fichero de la tecnología utilizando la función valorfich, declarada en la unidad sim4, y se definen los nodos de entrada y salida de la puerta que especifica el subcircuito.

Se recorre la lista que contiene los distintos valores de WE encontrados en el *netlist* para las puertas del tipo tratado, definiendo un subcircuito por cada valor consignado. Se añade el valor, de tipo cadena de caracteres y contenido en el campo bet, al final de la línea que define al transistor de enriquecimiento del subcircuito que se declara. El nombre del elemento de la lista que lo contiene se incluye en la línea que identifica el tipo de subcircuito. Finalmente se eliminan las dos listas enlazadas, soportadas por los punteros invlista y nor2lista, mediante sendas llamadas al método Eliminar.





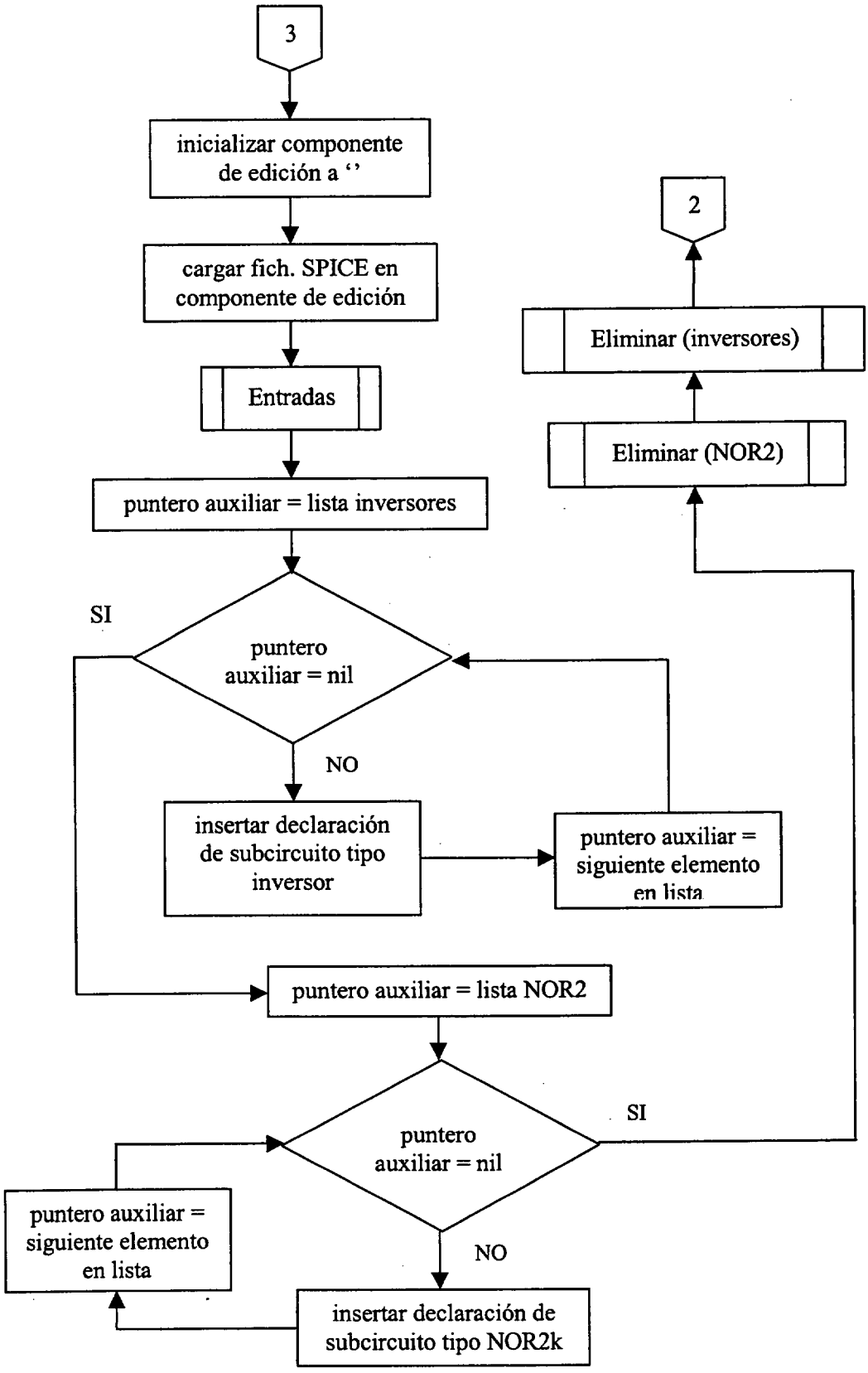


Figura 4. 58. Diagrama de flujos del método Trad1Click.

4.2.1.6.35. Método inicficheros.

El método inicficheros crea e inicializa los ficheros *netlist*, de entradas y de la tecnología que se generan al traducir al formato del modelo un fichero SPICE. Los archivos se crean mediante llamadas al procedimiento predefinido de tratamiento de ficheros *Rewrite*. En el caso del fichero de la tecnología se inserta la información correspondiente al valor nominal del retardo en las transiciones alta-baja, tphl0, y baja-alta, tplh0, información que no contiene el fichero SPICE.

4.2.1.6.36. Procedimiento buscaparametrosspi.

El procedimiento buscaparametrosspi se ha definido anidado al método ConvertirClick. Se utiliza en la traducción desde SPICE al modelo. Busca en la cadena de caracteres que recibe como parámetro el valor de los parámetros de modelado considerados como sensibles. La cadena que recibe corresponde a una línea de fichero SPICE encabezada por '.MODEL' o por el carácter '+'.

Se ha definido de forma local al método ConvertirClick un tipo de datos llamado parametrotx. Se trata de una estructura en forma de registro para almacenar los parámetros de modelado que se obtienen del fichero SPICE, y que pertenecen a uno de los dos tipos de transistor que se declaran. Consta de tres campos.

Un campo de tipo string, cadena de caracteres, llamado nombre, al que se asigna el nombre con que se designa al tipo de transistor en el fichero SPICE. Un campo de tipo byte, cuenta, se utiliza como contador de los parámetros que han sido introducidos en la estructura.

Finalmente un campo de tipo vectorcad llamado partros. Este tipo también ha sido definido, y constituye un array de siete elementos de tipo string. Se emplea para contener los parámetros de modelado obtenidos del fichero SPICE en formato de cadena de caracteres. En la estructura del array se ha establecido una posición fija para cada parámetro.

Para cada tipo de transistor el modelo considera que cinco parámetros son sensibles como puede verse en la tabla 3.II. Tres de dichos parámetros son coincidentes para ambos tipos de transistores: tensión umbral, resistencia de fuente y transconductancia, que en el formato SPICE se designan por el mismo nombre pese a referirse a distintos tipos de transistor. Los otros dos parámetros difieren según el tipo de transistor que se define.

En el momento del proceso de conversión en que se leen las declaraciones de los parámetros en el fichero SPICE el tipo de cada transistor declarado se desconoce. Sólo se determina el tipo cuando se comprueban las conexiones en la declaración de subcircuitos. Debido a esto se buscan los valores de los siete identificadores en formato SPICE: los tres comunes y los dos diferentes para cada tipo. Al tratar la sección del fichero SPICE en que se declaran los subcircuitos se determina el tipo de transistor que define cada declaración '.MODEL'. El procedimiento construirfichtec se encarga de insertar los parámetros adecuadamente en el fichero de la tecnología una vez establecido el tipo de cada modelo. Seguidamente se presenta la definición de los tipos de datos implementados:

```
vectorcad=array [1..7] of string;
parametrotx=record
        partros:vectorcad;
        nombre:string;
        cuenta,tipotx:byte;
        end;
```

El procedimiento buscaparametrosspi recibe como parámetro un registro de este tipo.

En el ejemplo que sigue se muestra la sección de un fichero SPICE en la que se definen los modelos de los transistores. Se han resaltado en negrita los parámetros sensibles que se insertan en los registros, uno para cada modelo de transistor, y subrayados los que se añaden al fichero de la tecnología una vez determinado el tipo de transistor que representa cada modelo. EFET03 define el modelo correspondiente al transistor de enriquecimiento y DFET03 el modelo del transistor de deplexión.

```
.MODEL EFET03 NMF VC=0.5495 VSB=5 LAMBDA=0.05 RD=600 RS=600 +RG=0.3
MFACT=1  GAMMA=4 IS=1e-13 DXL=10 VS=0.4 CDVC=0.00025
+ CDVSB=0 DELTA=0 NP=1.5 BETA=2.95 ALFA=5 FC=1 VST=1 CGS0=2e-16
+ CGS1=6e-16 VAT=-0.4 VBT=0.15 CDS=2e-16 RGS=800 RGD=800 TD=2e-12
**********
.MODEL DFET03 NMF VC=-0.1 VSB=0.78 LAMBDA=0.15 RD=500 RS=500
+RG=0.3 MFACT=2  GAMMA=2 IS=1e-14 DXL=3 VS=0.4 CDVC=0.00017
+CDVSB=0.00012 DELTA=1.9 NP=1.35 BETA=2.35 ALFA=3.5 FC=0.7 VST=1
+CGS0=1.8e-16 CGS1=4e-16 VAT=-1.1 VBT=-0.4 CDS=2e-16 RGS=900
+RGD=900 TD=2e-12
```

De forma local al procedimiento se ha definido una constante de tipo array. Es lo que se conoce como constante con tipo. Los elementos del array, de tipo cadena de caracteres, son los nombres en el formato de SPICE de los parámetros de modelado sensibles.

Un bucle controlado por el número de parámetros que contiene el registro, campo cuenta, y por un índice de tipo entero para acceder a cada elemento de la constante de tipo array se emplea para buscar en la línea del fichero SPICE los parámetros de modelado sensibles. Las sentencias presentes en el cuerpo del bucle se repiten hasta que el registro adquiera todos los parámetros, o se han buscado todos en la línea.

En el cuerpo del bucle se comprueba si el parámetro a buscar ya se encuentra contenido en el registro. Si no es así, se determina mediante la función predefinida *Pos* si la cadena fuente contiene la especificación de dicho parámetro. En ese caso se copia su valor a una variable auxiliar de tipo cadena de caracteres. Si el parámetro corresponde a uno de los que requieren realizar conversión de unidades, RS o CDVC, tanto para el transistor de deplexión como el de enriquecimiento, se utiliza el método ParSPICE para efectuarla. El valor del parámetro se asigna a la posición adecuada del campo de tipo array partros del registro, cuyo índice coincide con la posición de la cadena que lo representa en la constante de tipo array.

4.2.1.6.37. Procedimiento inicregistro.

El procedimiento inicregistro, definido localmente al método ConvertirClick, inicializa todos los campos del parámetro de tipo parametrotx que recibe. Asigna el valor

cero al campo cuenta de tipo byte, y el valor cadena vacía al campo nombre y a todos los elementos de tipo string del campo partros.

4.2.1.6.38. Función extraer.

La función extraer se declara de forma anidada al método ConvertirClick. Devuelve un valor de tipo real equivalente a su representación en formato de cadena de caracteres. Los caracteres que representan al número se encuentran al final de la cadena que la función recibe como parámetro, y representan la anchura de un transistor definida en una declaración de subcircuito. Si el valor no es convertible devuelve cero. A través de un índice, de tipo entero, se accede al carácter final de la cadena. Dicho índice se decrementa hasta que apunta al carácter inicial del valor de la anchura del transistor. Así se puede extraer de la cadena fuente el valor de la anchura utilizando la función de tratamiento de cadenas *Copy*. La cadena de caracteres obtenida se convierte a formato real mediante el procedimiento predefinido *Val*.

4.2.1.6.39. Procedimiento lineapto.

El procedimiento, definido localmente al método ConvertirClick, se utiliza para tratar las líneas del fichero SPICE encabezadas por el carácter '.'. El tipo de datos que contiene se identifica por la palabra que sigue a dicho carácter.

La palabra MODEL designa que el texto contiene el inicio de la declaración de los parámetros que definen a un transistor empleado en la simulación SPICE. En esta línea se incluye el nombre con que se identifica el tipo de transistor que se define, y una serie de parámetros. El procedimiento recibe dos registros de tipo parametrotx, definido para contener los parámetros de modelado que se encuentran en el fichero SPICE. Se comprueba consultando el valor contenido en el campo cuenta qué registro está vacío, y se asigna al campo nombre del mismo la cadena que identifica al tipo de transistor en la declaración .MODEL. Se llama al procedimiento buscaparametrosspi, al que se pasan el registro y la línea del fichero SPICE, para que inserte en aquél los parámetros sensibles que ésta contenga.

La palabra SUBCKT identifica el comienzo de una declaración de subcircuito. El nombre del subcircuito se obtiene de la línea del fichero SPICE empleando la función CopiaSubCad, definida en la unidad sim4. Se asigna a un parámetro del procedimiento de tipo cadena de caracteres. Se usará cuando se haya determinado la relación de aspecto que define al subcircuito para incluir ambos datos en una lista enlazada de elementos de tipo InfoS, que contendrá los valores de β de los subcircuitos definidos en el fichero y sus nombres. Dos parámetros de tipo real se inicializan a cero, y corresponden a dos variables locales de ConvertirClick definidas para contener los valores de las anchuras de los transistores declarados en el subcircuito. Dos parámetros de tipo lógico se inicializan con el valor verdadero. Uno de ellos indica que las siguientes líneas del fichero SPICE se incluyen en el bloque de declaración de un subcircuito, y pasa a valer falso cuando se detecte el final de la definición del subcircuito. El otro parámetro lógico se emplea para determinar el tipo de puerta que define el subcircuito, indicando el valor verdadero que se trata de un inversor. El procedimiento lineaz, que se emplea para tratar las líneas de fichero que definen los transistores del subcircuito le asigna falso si determina que el subcircuito representa a una puerta NOR2.

La palabra ENDS indica que la línea del fichero SPICE contiene el final de la declaración de un subciruito, con lo que los dos parámetros de tipo real contienen los valores de las anchuras de los transistores del subcircuito actual. El valor de β se calcula al dividir la anchura del transistor de enriquecimiento entre la correspondiente al transistor de deplexión.

$$\beta = \frac{W_E}{W_D}$$

Un parámetro de tipo lógico indica si el subcircuito definido corresponde a una puerta lógica inversora o NOR2. Según el tipo base de la puerta el valor de la relación de aspecto y el nombre identificador del subcircuito se deben insertar en las listas enlazadas que se han definido para cada tipo básico de puerta. El procedimiento recibe como parámetros los punteros de tipo PInfoS que mantienen ambas listas. Para incluir la información se llama al método InsertarBetayNombre.

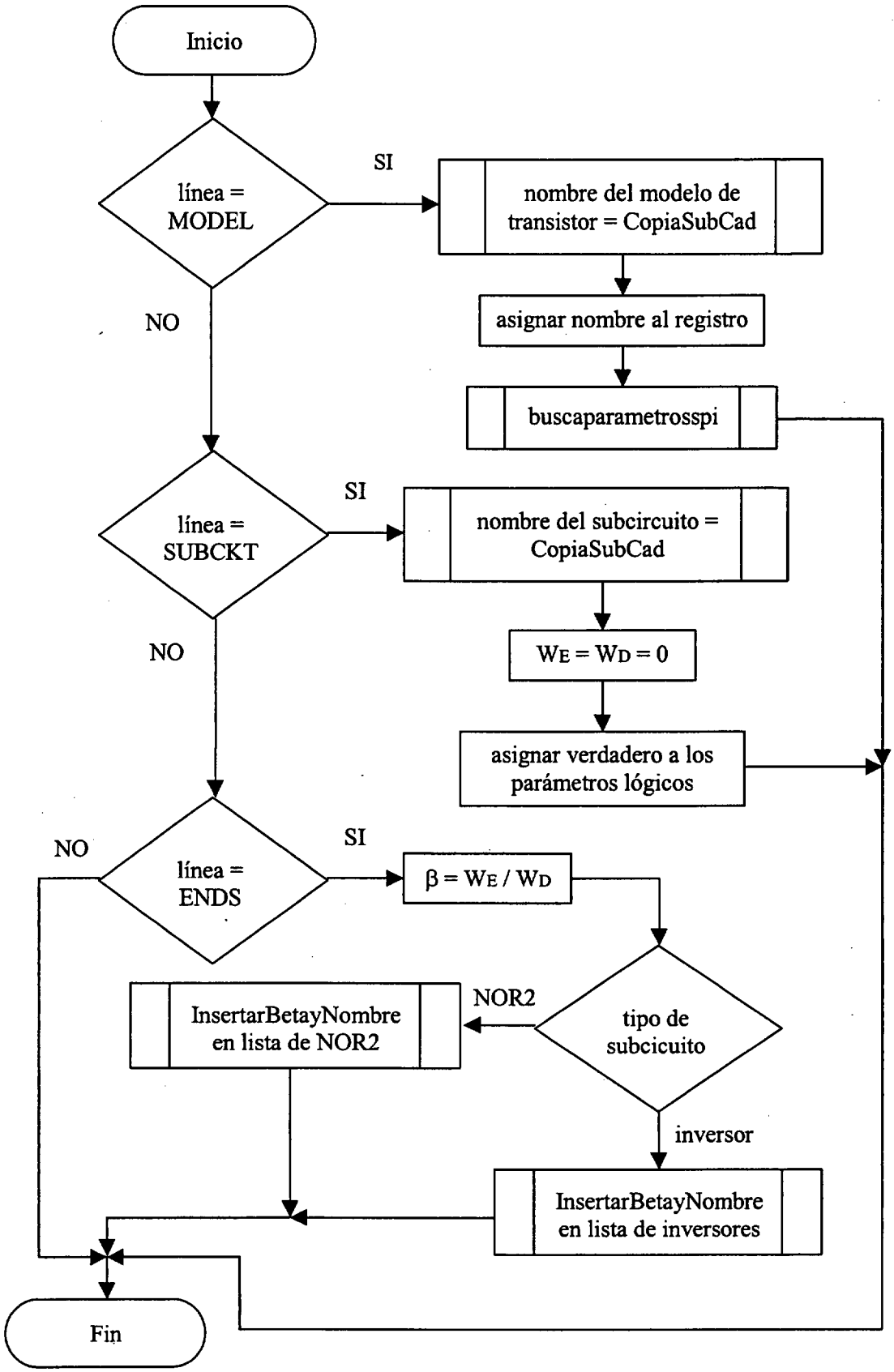


Figura 4. 59. Diagrama de flujos del procedimiento lineapto.

4.2.1.6.40. Procedimiento lineaplus.

El procedimiento lineaplus extrae la información correspondiente a las líneas del fichero SPICE cuyo primer carácter es '+'. Estas líneas contienen la declaración de los parámetros que definen a los transistores empleados por la simulación SPICE. El procedimiento se define anidado al método ConvertirClick. De los parámetros contenidos en el fichero se obtienen aquellos que son considerados como sensibles por el modelo basado en el análisis de la sensibilidad.

Recibe como parámetros la línea del fichero SPICE y dos registros de tipo parametrotx, definido para contener los valores de los parámetros de modelado. Se comprueba con cuál de los dos registros se debe trabajar, y se llama al procedimiento buscaparametrosspi al que se le pasan la cadena de texto SPICE y el registro que recibirá los parámetros que ésta contenga.

4.2.1.6.41. Procedimiento lineaz.

El procedimiento lineaz se define anidado al método ConvertirClick. Se emplea para obtener de una línea del fichero SPICE donde se declara un transistor de un subcircuito el valor de la anchura, y determina el tipo de transistor que define la línea y el nombre que lo identifica. Estas líneas están encabezadas por el carácter 'Z'.

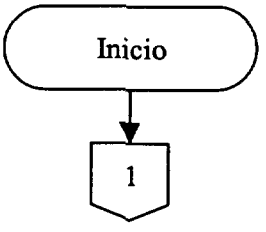
El procedimiento recibe como parámetros dos variables de tipo real para contener los valores de las anchuras según el transistor definido sea de enriquecimiento o de deplexión. Recibe también un parámetro lógico, que se emplea como indicador del tipo básico de puerta, inversor o NOR2, que se define en la declaración actual de subcircuito. Al procedimiento se le pasan, igualmente, dos parámetros de tipo cadena de caracteres para contener los nombres de los modelos de los transistores usados en la simulación SPICE, que se especifican en cada línea de definición de un transistor. En el bloque de definición de los parámetros de modelado se ha obtenido el nombre con que se designa el modelo de cada tipo de transistor. Ahora, en el procedimiento lineaz se determina el tipo de transistor, completando la información de los modelos.

En una línea de este tipo se definen las conexiones del transistor, el nombre del modelo de transistor al que pertenece y el valor de la anchura el mismo, como puede verse en el siguiente ejemplo:

ZEI1 2 1 0 EFET03 10

Si la línea es la primera de esta clase en el fichero SPICE, se desconoce qué tipo de transistor define cada modelo declarado en la sección .MODEL del fichero. De la cadena fuente se extraen todos los datos que contiene utilizando la función CopiaSubCad, declarada en la unidad sim4. Las conexiones del transistor permiten determinar el tipo, enriquecimiento o deplexión, del mismo. El nombre con que se designa, en el caso del anterior ejemplo EFET03, se asigna al parámetro de tipo cadena definido para almacenar el nombre del modelo del transistor, en esta ocasión de enriquecimiento. El valor de la anchura del transistor se convierte a formato real mediante el procedimiento predefinido *Val*, y se asigna al parámetro de tipo real adecuado según el tipo del transistor.

En caso de conocerse el nombre de cada modelo de transistor, almacenado en los parámetros de tipo cadena, éste se usa para determinar qué transistor define la línea del fichero SPICE con que se trabaja. La función de tratamiento de cadenas *Pos* determina si el nombre con que se designa un tipo se encuentra en la cadena de caracteres. Después de identificar el tipo, el valor de la anchura del transistor se extrae de la cadena fuente llamando a la función extraer que devuelve su valor en formato real. Además, si el transistor que se declara en la línea es de enriquecimiento, se comprueba si el parámetro definido para contener el valor de su anchura presenta un valor distinto de cero. Esto supone que la línea pertenece a la definición de un subcircuito que representa a una puerta NOR2 con lo que se asigna al indicador lógico del tipo de puerta el valor falso. Si es cero o el transistor es de deplexión, y el valor que devuelve la función extraer se asigna al correspondiente parámetro de tipo real.



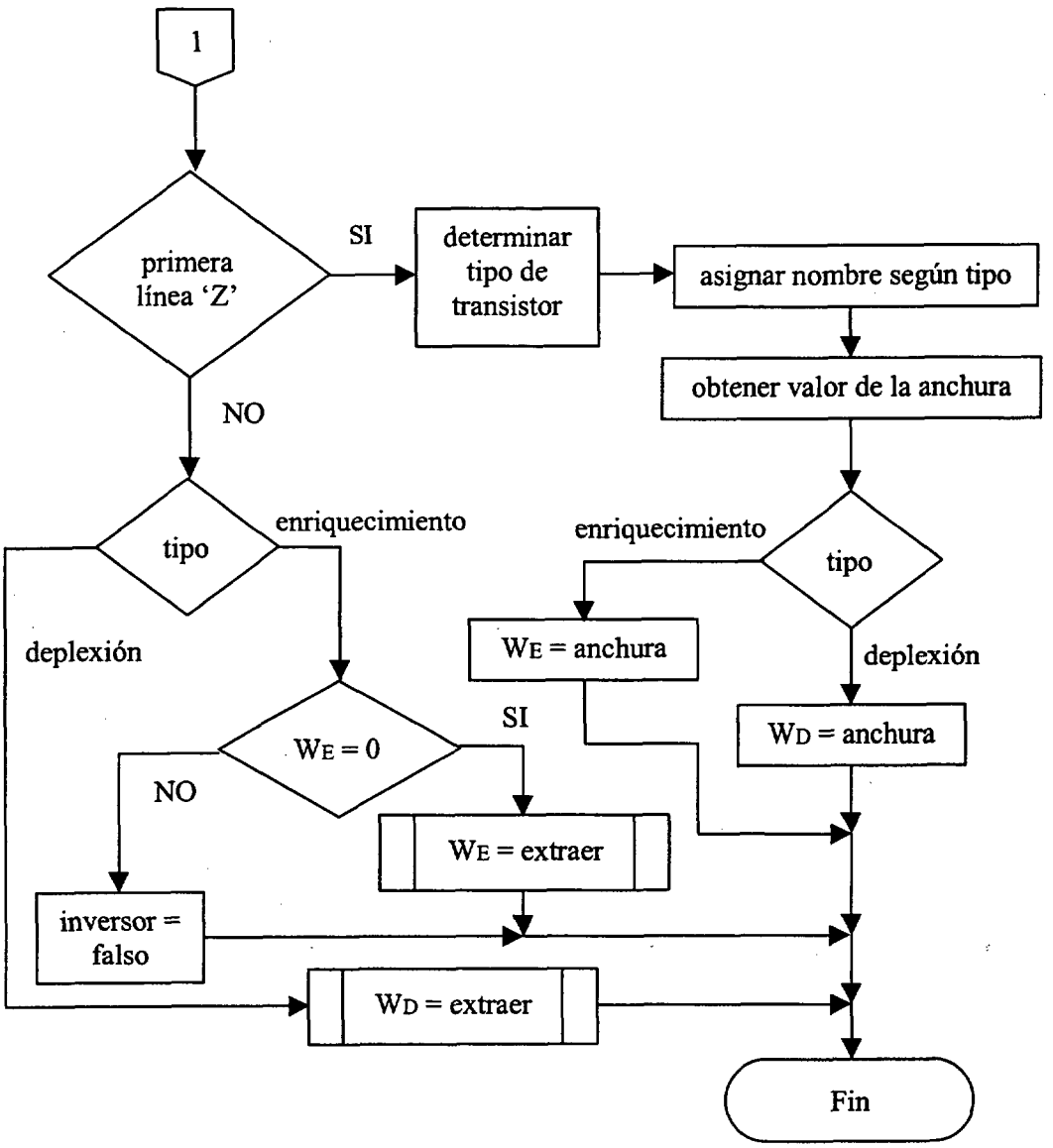


Figura 4. 60. Diagrama de flujos del procedimiento lineaz.

4.2.1.6.42. Función ValorBeta.

La función ValorBeta se define anidada al método ConvertirClick. Busca en una lista enlazada de elementos de tipo InfoS, soportada por un puntero de tipo PInfoS, el valor de la relación de aspecto propio del tipo de subcircuito cuyo nombre recibe como parámetro. Devuelve el valor de β en formato de cadena de caracteres, que contiene el campo bet del elemento de la lista que le corresponde. Mediante un puntero auxiliar se

recorre la lista comparando el nombre recibido con el contenido del campo nombre de cada miembro de la lista hasta encontrar el elemento buscado. Si concluye el recorrido de la lista sin hallarlo la función devuelve la palabra Error.

4.2.1.6.43. Función ValorDC.

La función ValorDC se ha definido localmente al método ConvertirClick. Devuelve una cadena de caracteres que corresponde al valor del nivel constante de tensión especificado en una línea del fichero SPICE. La palabra dc identifica el contenido de la línea. La cadena del fichero SPICE se le pasa como parámetro. El dato a obtener se encuentra al final de la cadena fuente. Se accede al carácter final de la cadena mediante un índice de tipo entero. Dicho índice se decrementa hasta que apunta al carácter inicial del valor de la tensión. Así el valor se puede extraer de la cadena fuente utilizando la función de tratamiento de cadenas *Copy*.

4.2.1.6.44. Procedimiento InEntrada.

El procedimiento InEntrada se define anidado al método ConvertirClick. Se emplea para escribir en el fichero de entradas que se crea a partir de un fichero SPICE la definición de una entrada de nivel constante. El procedimiento recibe como parámetros la línea del fichero SPICE que contiene la declaración de la entrada y el fichero que se está generando. La función CopiaSubCad, definida en la unidad sim4, se utiliza para obtener de la cadena del fichero SPICE el identificador del nodo sobre el que se aplica la entrada. El valor de la tensión especificado se obtiene llamando a la función ValorDC. Según el valor de tensión a la entrada se le asigna 'L', que denota un nivel bajo, o 'H', nivel alto. Con estos datos se construye una entrada conforme al formato establecido para el modelo basado en la sensibilidad, que se escribe en el fichero utilizando los procedimientos predefinidos de tratamiento de archivos de texto *Append* y *writeln*.

4.2.1.6.45. Procedimiento lineav.

El procedimiento lineav, definido anidado al método ConvertirClick, se emplea para el tratamiento de las líneas del fichero SPICE cuyo primer carácter es 'V'. Este tipo de líneas define una entrada al circuito de tipo Pwl o un nivel fijo de tensión. En este caso puede tratarse de una entrada o de la tensión de alimentación. Determinadas palabras incluidas en el texto identifican el tipo de señal que especifica. La palabra 'DC' se incluye en las líneas que definen una tensión fija. Una línea que declara una entrada con una transición debe contener la palabra 'Pwl'.

Si la cadena de caracteres declara una tensión constante hay que distinguir entre una entrada y la alimentación. Si la cadena se incluye dentro del bloque en el que se define un subcircuito se trata de la tensión de alimentación. Se declara en cada subcircuito y debe ser el mismo valor para todos por equivalencia con el modelo basado en la sensibilidad. El valor se extrae de la cadena fuente llamando a la función ValorDC y se incorpora al fichero de la tecnología que se abre mediante el procedimiento de tratamiento de ficheros *Append*. Se devuelve un parámetro de tipo lógico, al que se asigna el valor verdadero, para indicar que el valor de la tensión de alimentación se ha añadido al fichero de la tecnología. De esta forma si la tensión de alimentación se especifica para cada subcircuito declarado en el fichero SPICE consultando el valor de la variable lógica se impide repetir el proceso de adquisición de su valor y escritura en el fichero de la tecnología.

Si no se incluye en la sección de definición de un subcircuito se utilizan otros medios para determinar el tipo de señal que representa la cadena. Si el parámetro de tipo lógico que indica si se ha insertado el valor de la tensión de alimentación contiene el valor verdadero la línea del archivo SPICE especifica una entrada. En este caso se llama al procedimiento InEntrada para insertarla en el fichero de entradas que se está creando. Si el valor es falso se trata de una entrada si el valor especificado en la cadena corresponde al nivel bajo. Se llama a la función ValorDC que devuelve el valor definido para comprobarlo. En caso afirmativo se llama al procedimiento InEntrada.

Si no se trata de ninguno de los casos anteriores no se sabrá el tipo de señal que define la línea del fichero SPICE hasta concluir la lectura del archivo. La línea se almacena en una lista enlazada definida para ello. El tipo declarado contiene dos campos. Uno de

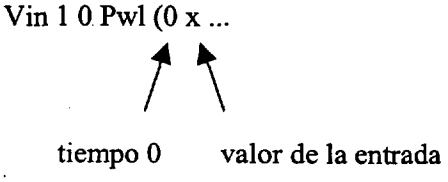
tipo string para contener la cadena y un campo puntero de enlace entre los elementos de la lista. Este tipo de datos se define de forma local al método ConvertirClick y se muestra a continuación:

```
PListCad=^ListCad;
ListCad= record
        cad:string;
        sig:PListCad;
end;
```

Cuando la línea del fichero SPICE contiene la declaración de una entrada de tipo Pwl se incorpora al fichero de entradas (*.ent).

Se utiliza la función CopiaSubCad, declarada en la unidad sim4, para obtener de la cadena fuente el identificador del nodo del circuito sobre el que se aplica la entrada.

Para obtener el sentido de la transición que presenta se comprueba el valor que muestra la entrada SPICE en la primera referencia de tiempo, tiempo 0, que se adquiere borrando los caracteres precedentes y utilizando la función CopiaSubCad.



Si el valor es '0' la pendiente es ascendente, con lo que la entrada a aplicar en el fichero de entradas se constituye asignando el carácter '1' como valor de la pendiente. En caso de valer '1' la pendiente es descendente y a la entrada se le asigna '-1' como valor de su pendiente.

La segunda referencia de tiempo presente en la entrada SPICE se utiliza para obtener el retardo que debe presentar la entrada con respecto al origen de tiempos. Debe ser una expresión en picosegundos. La cadena correspondiente a dicha referencia temporal se extrae de la cadena fuente por medio de la función CopiaSubCad.

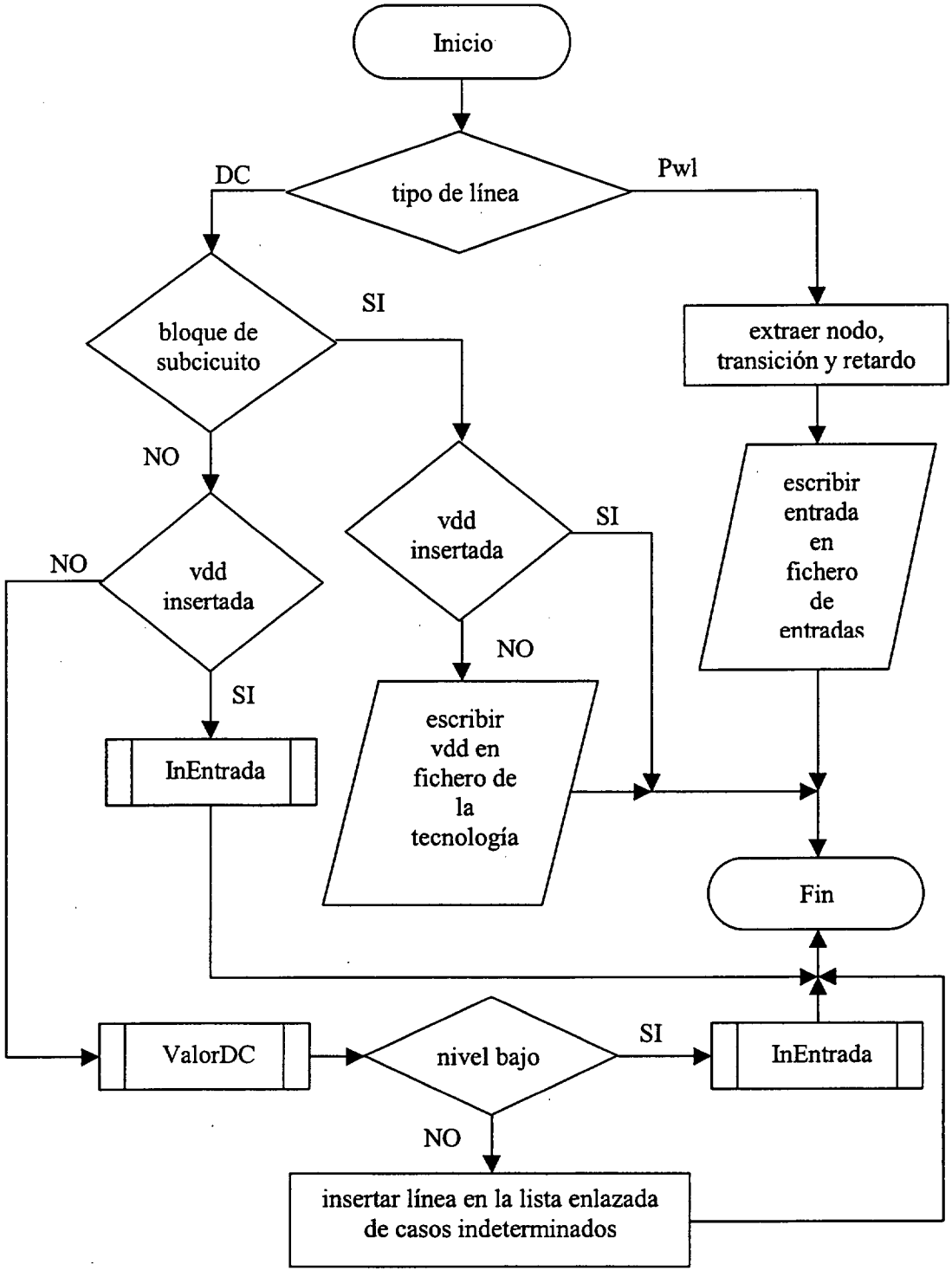


Figura 4. 61. Diagrama de flujos del procedimiento lineav.

4.2.1.6.46. Procedimiento parametrosenriquecimiento.

El procedimiento se ha definido anidado al método ConvertirClick. Inserta en el fichero de la tecnología que se crea los valores de los parámetros de modelado correspondientes al transistor de enriquecimiento. Recibe un parámetro de tipo parametrotx cuyo campo partros contiene los valores de los parámetros.

El fichero de la tecnología se abre mediante el procedimiento predefinido *Append* para añadir la información al final del mismo. Una vez insertados los parámetros se cierra el fichero utilizando el procedimiento predefinido *CloseFile*.

4.2.1.6.47. Procedimiento parametrosdeplexion.

El procedimiento, análogo a parametrosenriquecimiento, se define anidado al método ConvertirClick. Inserta en el fichero de la tecnología que se crea a partir del fichero SPICE los valores de los parámetros de modelado del transistor de deplexión. Recibe un parámetro de tipo parametrotx, cuyo campo partros contiene los valores de los parámetros y el fichero de la tecnología.

El fichero se abre utilizando el procedimiento predefinido *Append*, que permite insertar la información al final del mismo. Después de añadir los parámetros se cierra el fichero utilizando el procedimiento predefinido *CloseFile*.

4.2.1.6.48. Procedimiento construirfichtec.

El procedimiento construirfichtec se define anidado al método ConvertirClick. Añade al fichero de la tecnología en creación los parámetros de modelado definidos en el fichero SPICE.

La información para construir el fichero se encuentra en dos registros de tipo parametrotx que el procedimiento recibe como parámetros. Estos registros contienen los valores de los parámetros de modelado de los transistores y los nombres con que se

designan los modelos en las declaraciones MODEL del archivo SPICE. También se le pasan dos parámetros de tipo cadena de caracteres con los nombres utilizados en el fichero SPICE para identificar los modelos de los transistores, una vez determinado en el procedimiento lineaz el tipo de transistor que define cada modelo. Comparando el contenido del campo nombre de cada registro con los dos parámetros recibidos se determina si los valores que contiene corresponden al transistor de enriquecimiento o al de deplexión. Se llama a los procedimientos parametrosenriquecimiento y parametrosdeplexion, a los cuales se les pasa como parámetros el fichero en proceso de creación y el registro adecuado.

4.2.1.6.49. Procedimiento InEntyTec.

El procedimiento se define anidado al método ConvertirClick. Se emplea para escribir en los ficheros de entradas y de la tecnología aquellas líneas del fichero SPICE que definen una tensión, cuya función, entrada al circuito fijada al nivel alto o alimentación, no se pudo determinar en el proceso de lectura de los datos. Esta indeterminación se produce cuando en el archivo SPICE la tensión de alimentación se especifica fuera de la declaración de subcicuito. Las líneas del fichero se han copiado en el procedimiento lineav a una lista enlazada que se recibe como parámetro a través del puntero que la soporta.

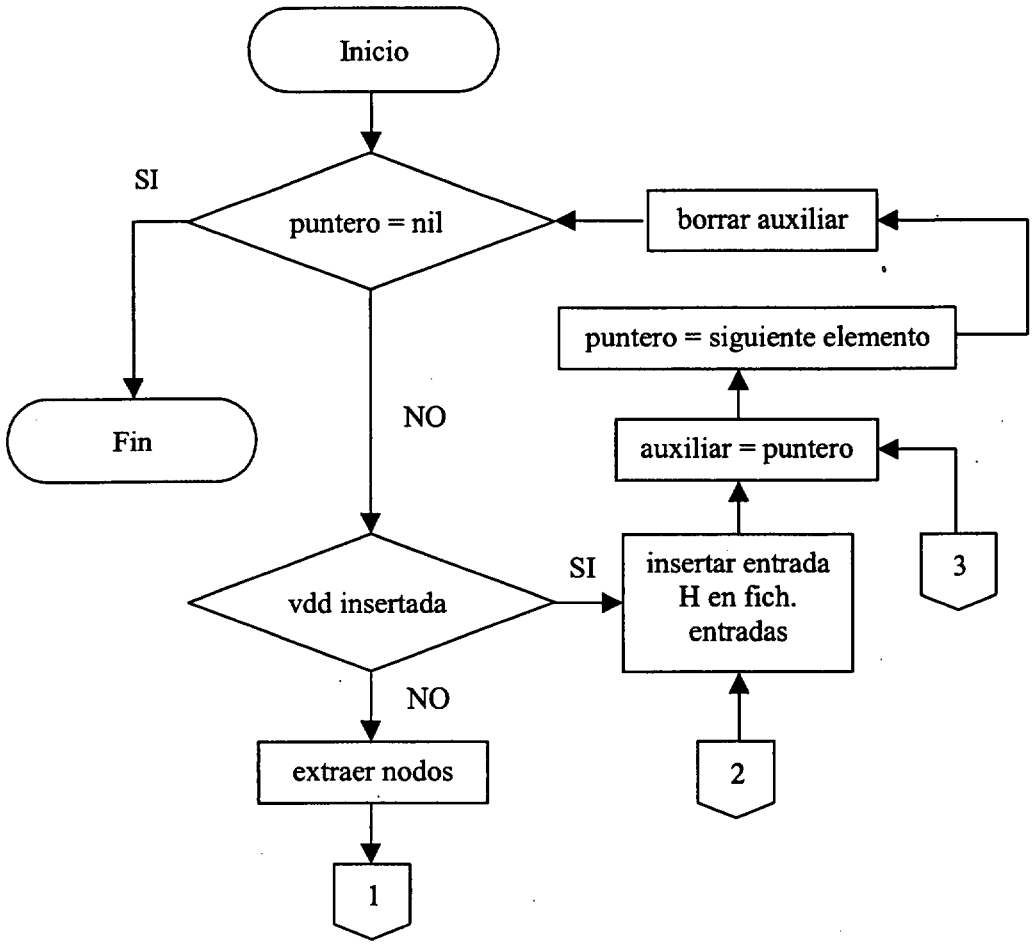
Para determinar el tipo de información que contiene cada línea se deben comparar los nodos especificados en la misma y los definidos en las puertas del circuito. Cuando la tensión de alimentación se especifica fuera de los subcircuitos cada puerta definida en el circuito contiene los identificadores de los nodos de alimentación y tierra. Así para un inversor de ejemplo conectado entre los nodos 1 y 2, perteneciente al tipo de subcircuito inversor1 sería:

X1 1 2 20 0 inversor1

donde 20 y 0 corresponden a los nodos de alimentación y tierra. A la línea contenida en la lista enlazada se le extraen los identificadores de los nodos sobre los que se aplica la señal. Se comprueba mediante la función de tratamiento de cadenas *Pos* si forma parte de la cadena de definición de la puerta, en cuyo caso la línea especifica la tensión de alimentación. En caso contrario se trata de una entrada de nivel alto que se inserta en el fichero de entradas.

El procedimiento recibe como parámetros los dos ficheros que se están creando y un índice de tipo entero que corresponde a la línea donde comienza en el fichero SPICE el bloque en que se define el circuito. Este índice permite acceder al fichero SPICE empleando la propiedad *Lines*, de la clase *TStrings*, del componente de edición de la ficha principal. El procedimiento devuelve un parámetro de tipo *string*, cadena de caracteres, que contendrá los nodos de alimentación y tierra en el caso en que la tensión de alimentación se especifique fuera de los subcircuitos. Se emplea para borrar dichos nodos, que SiDSen no admite en el formato establecido, al crear el fichero *netlist*.

Como se ha dicho las líneas del fichero se encuentran copiadas en una lista enlazada. El proceso de comparación y escritura se realiza recorriendo la lista. Cada vez que se inserta la información de un elemento en su correspondiente fichero se procede a su eliminación. Cuando el puntero que soporta la lista contenga el valor nil, indicativo de lista vacía, concluye la ejecución del procedimiento.



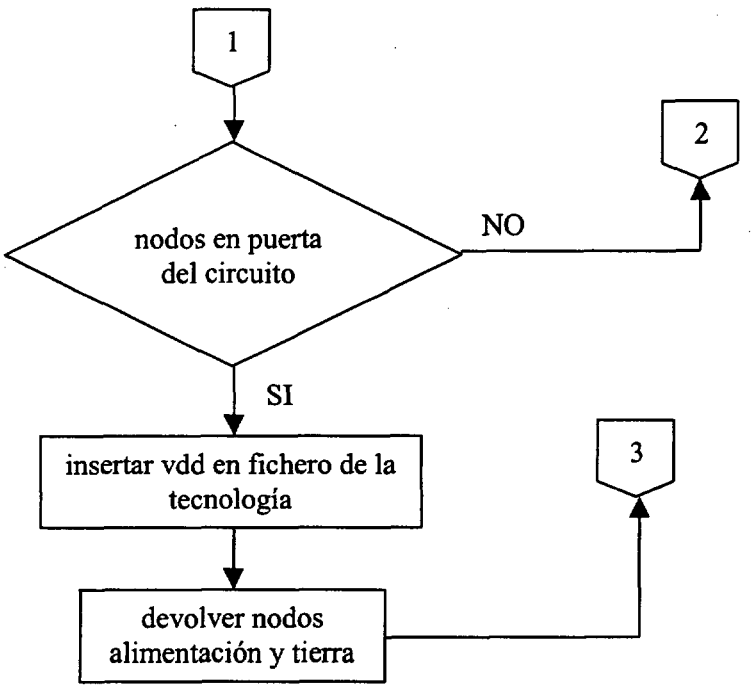


Figura 4. 62. Diagrama de flujos del procedimiento InEntyTec.

4.2.1.6.50. Procedimiento ConsNetlist.

El procedimiento se define de forma local al método ConvertirClick. Se utiliza para escribir en el fichero *netlist* (.net) el circuito definido en el archivo SPICE. Recibe como parámetros el fichero en creación y dos listas enlazadas, soportadas por punteros de tipo PInfoS, que contienen los nombres y los valores de β de los subcircuitos definidos para cada tipo base de puerta. También se le pasa un parámetro de tipo entero que contiene el número de la línea del fichero SPICE en que comienza la declaración del circuito. Mediante un parámetro de tipo *string* recibe los identificadores de los nodos de alimentación y tierra en el caso de que la tensión de alimentación se especifique externamente a la declaración de subcicuitos.

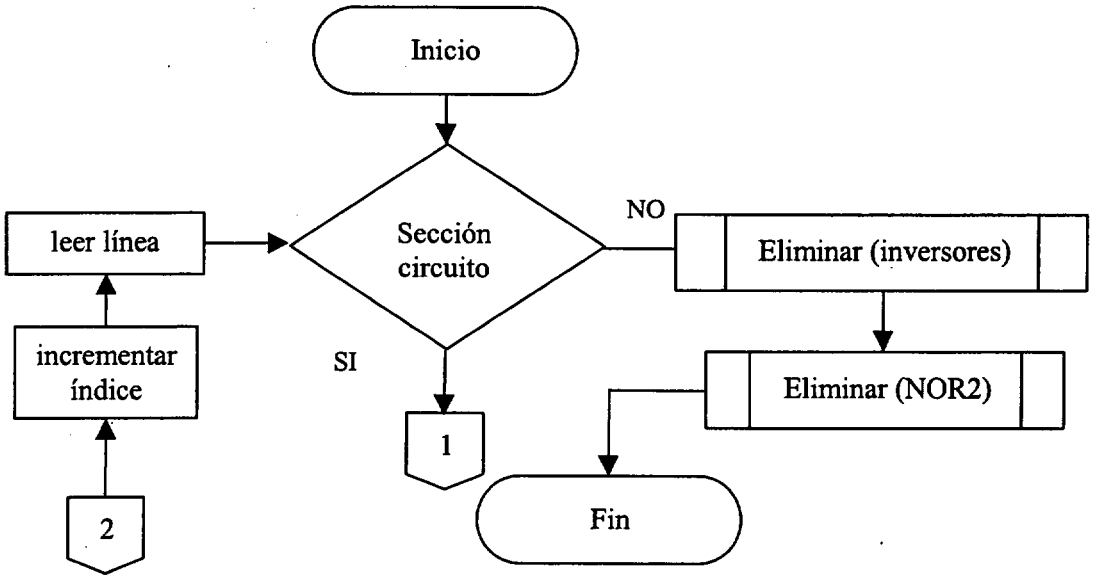
Cada línea del fichero SPICE en que se especifica una puerta del circuito comienza con el carácter 'X'. Se emplea un bucle que comprueba este hecho para la lectura del circuito, incrementando el índice de tipo entero en cada pasada. Este índice permite acceder a las líneas del fichero SPICE utilizando la propiedad *Lines*, de la clase *TStrings*, perteneciente al componente de edición de la ficha principal.

El formato de la línea depende de cómo se haya especificado la tensión de alimentación en el archivo SPICE. Si se define fuera del bloque de subcircuitos la cadena contiene los nodos de alimentación y tierra que se deben borrar. Esto se comprueba comparando el contenido de la línea y el parámetro de tipo *string* recibido.

Los datos que presenta la cadena son los nodos a que se conecta la puerta y el nombre del tipo de subcircuito a que pertenece. Todos estos datos se obtienen de la cadena fuente usando la función CopiaSubCad, declarada en la unidad sim4. El valor de la relación de aspecto propio de la puerta se consigue de la correspondiente lista enlazada según el tipo de puerta, NOR2 o inversor. El valor se determina a partir del identificador del subcircuito de la línea del fichero llamando a la función ValorBeta.

Con los datos correspondientes a los nodos leídos de la cadena SPICE, el valor de β y el identificador del tipo de puerta, inv o nor2, se genera una línea de fichero *netlist* conforme al formato establecido. Esta cadena se inserta al final del archivo que se abre mediante el procedimiento *Append*.

Las dos listas enlazadas se destruyen una vez concluye la escritura del circuito en el fichero *netlist* al detectar el final de la sección SPICE en que se define. Se realizan sendas llamadas al método Eliminar.



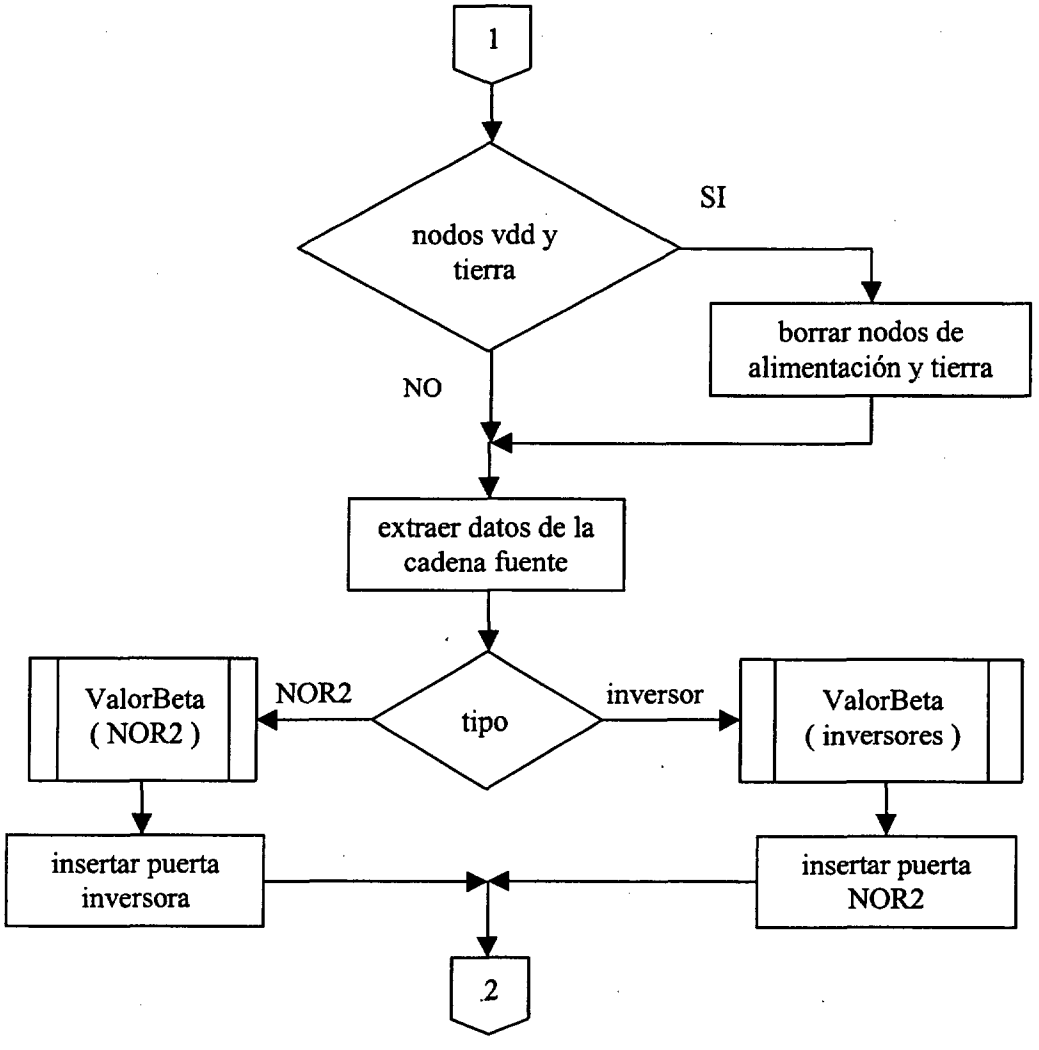


Figura 4. 63. Diagrama de flujos del procedimiento ConsNetlist.

4.2.1.6.51. Método ConvertirClick.

El método ConvertirClick es el gestor del evento OnClick del elemento Convertir, Traducir desde SPICE, del menú Simular. El método crea a partir del fichero SPICE presente en el componente de edición de la ficha principal archivos de tipo *netlist*, de entradas y de la tecnología, conforme al formato establecido para cada uno de ellos. De forma local al método se han definido los procedimientos y funciones que emplea para generar los ficheros. El código implementado se divide en dos partes: una primera de lectura de los datos definidos en el fichero SPICE, y la segunda de construcción de los ficheros a partir de los datos leídos.

Se comprueba la existencia de texto en el elemento editor verificando el contenido de la propiedad *Text* del mismo. Si no hay texto se informa al usuario por medio de un cuadro de mensaje. Se emplea un cuadro de tipo *MessageDlg*, definido como cuadro de mensaje de error al asignar el valor *mtError* al parámetro que define el tipo de cuadro. Se ha situado en el cuadro un botón de tipo *mbOK*, Aceptar.

Se comprueba mediante consulta de la propiedad *Modified* del elemento de edición si el fichero SPICE ha sufrido cambios. Si es así se pregunta al usuario si desea guardar los cambios, omitirlos o poner fin a la operación en curso, para lo que se llama al método SalvarCambios.

Se generan los nombres de los tres ficheros a crear a partir del nombre del fichero SPICE que se encuentra en el campo NombreFich de la ficha. Al nombre del fichero se le añade la correspondiente extensión para cada tipo de archivo.

Las variables declaradas localmente en el método se inicializan. Los registros de tipo parametrotx, utilizados para la lectura de los parámetros que modelan a los transistores de enriquecimiento y deplexión, en la sección .MODEL del fichero SPICE se inicializan mediante sendas llamadas al procedimiento inicregistro.

Se llama al método inicficheros, que crea e inicializa los ficheros *netlist*, de entradas y de la tecnología.

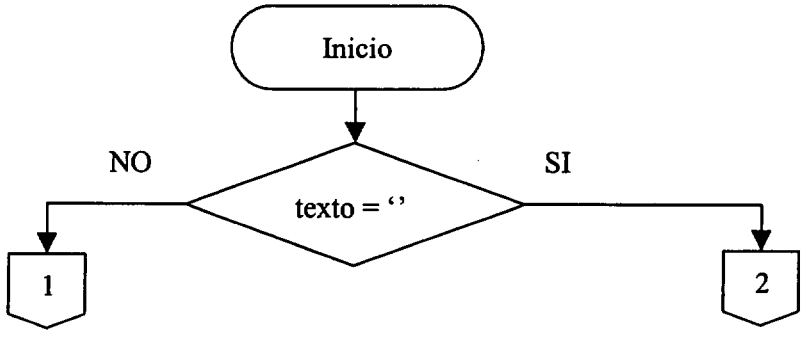
La fase de obtención de los datos del fichero SPICE se realiza mediante la lectura secuencial de todas las líneas que conforman el fichero. Se accede a cada línea a través de la propiedad *Lines* del componente de edición.

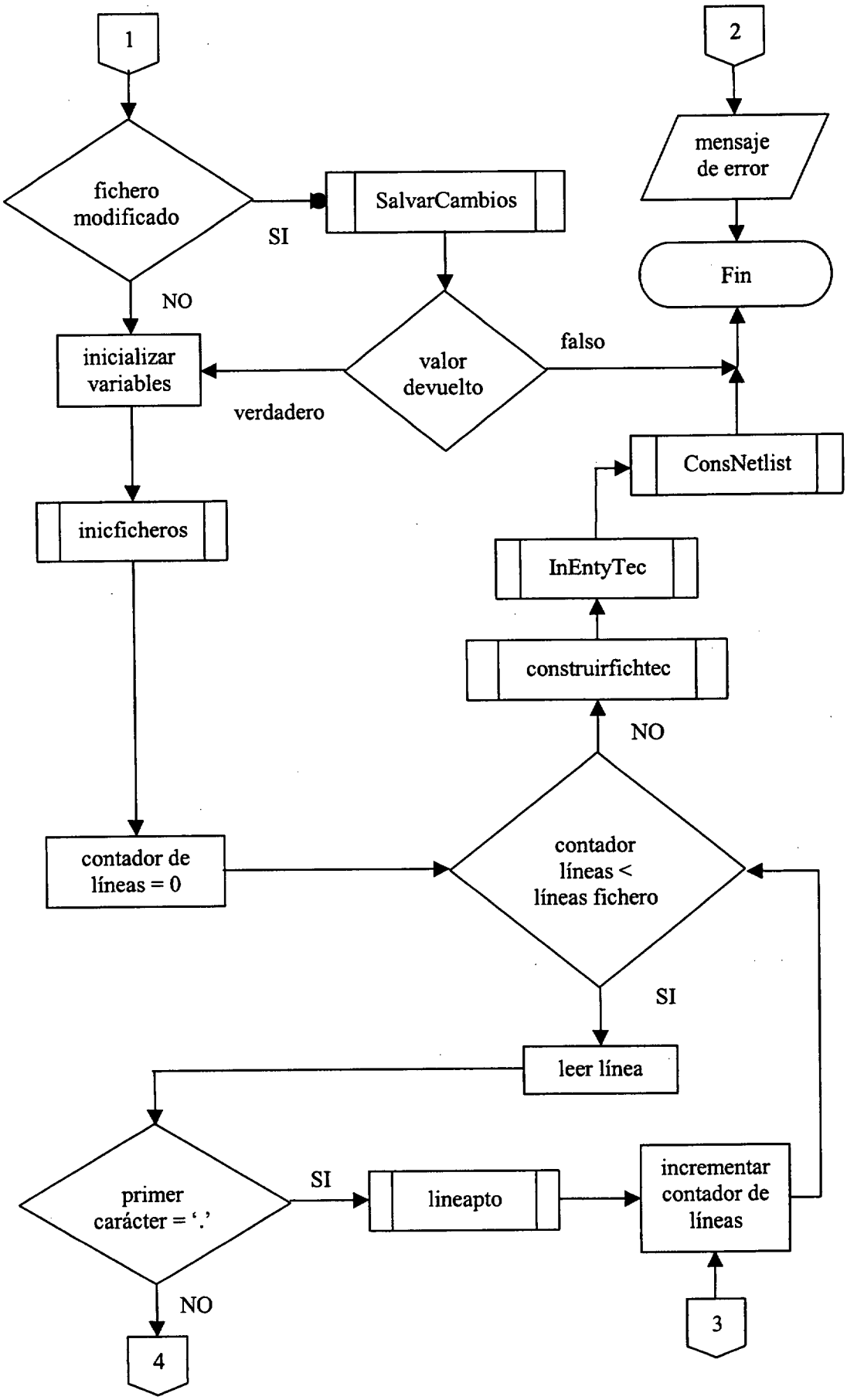
Para cada línea se comprueba cuál es el primer carácter y en función del mismo se llama al procedimiento correspondiente, que obtiene la información que contiene. Se emplean para este fin los procedimientos lineapto, lineaplus, lineaz y lineav. Este último inserta en los ficheros de la tecnología y de entradas aquellas señales definidas en el fichero SPICE cuyo cometido puede determinarse en la ejecución del procedimiento. Si el carácter inicial de la cadena es X la información que contiene es la definición de una puerta del circuito. Cuando se detecta la primera línea en el fichero de este tipo se utiliza una

variable de tipo entero para almacenar el número de línea. Se empleará posteriormente en la etapa de escritura en los ficheros para acceder a la sección del archivo SPICE en que se define el circuito.

Una vez concluida la lectura del fichero SPICE, se llama a tres procedimientos que generan los ficheros *netlist*, de entradas y de la tecnología con los datos obtenidos. Son construirfichtec, InEntyTec y ConsNetlist. El primero añade al fichero de la tecnología los parámetros de modelado sensibles definidos en el archivo SPICE. InEntyTec completa la información del fichero de la tecnología y del fichero de entradas si la tensión de alimentación se ha definido en el fichero SPICE fuera de la declaración de subcircuitos. ConsNetlist inserta en el fichero *netlist* la estructura del circuito definido en el fichero SPICE.

Se informa al usuario de la finalización del proceso de conversión mediante un cuadro de mensaje. Para ello se llama a *MessageBox*, que presenta un cuadro de mensaje que permite escoger el icono que se agrega al mensaje. Se ha utilizado el icono de tipo *MB_ICONINFORMATION* y se ha dispuesto un botón de tipo *MB_OK*, Aceptar. El mensaje informa al usuario de los nombres de los ficheros que han sido creados, así como de su ubicación, que es la misma que el fichero SPICE. El texto del mensaje ha de estar en formato *PChar*, cadena de caracteres terminada en el carácter nulo, y no en el formato *string* en que se tienen los nombres de los ficheros. Se realiza una conversión de tipos cuyo resultado se asigna al parámetro de MessageBox que recibe el mensaje a mostrar. La conversión se realiza anteponiendo al dato a convertir, incluido entre paréntesis, el nombre del tipo de datos al que se desea convertir aquél. En este caso la conversión se obtiene mediante PChar (texto con el nombre de los ficheros).





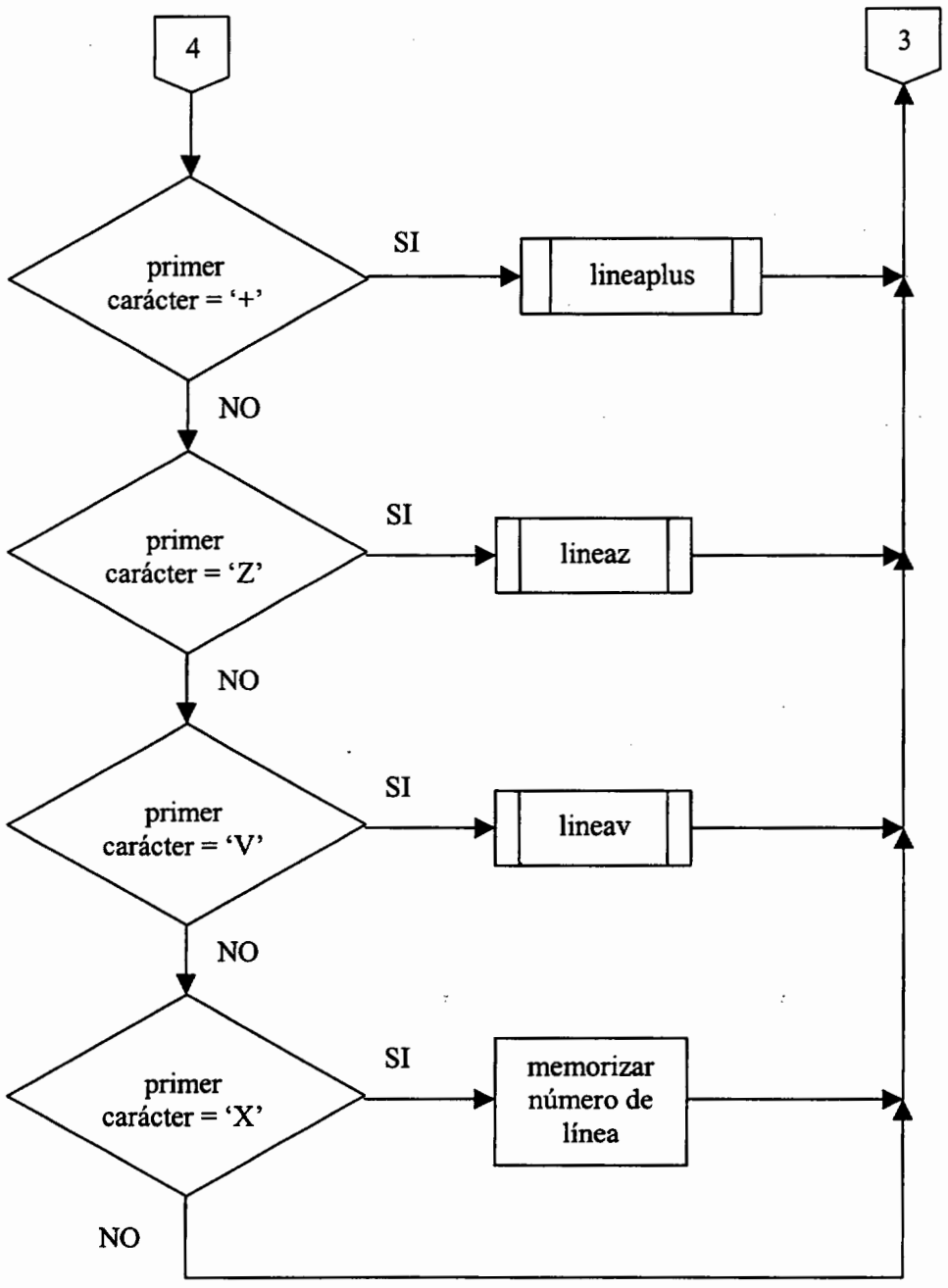


Figura 4. 64. Diagrama de flujos del método ConvertirClick.

4.2.1.6.52. Método Ayuda2Click.

El método Ayuda2Click gestiona el evento OnClick del elemento Ayuda2, Ayuda, del menú desplegable Ayuda de la aplicación.

Muestra la ayuda de la aplicación. Para ello se emplea el método *HelpCommand* del objeto *Application*. Este método da acceso al sistema de ayuda de Windows. A través de un parámetro del mismo se especifica el tipo de acceso a la ayuda. Se le ha asignado el valor HELP_FINDER, que muestra la ayuda presentando el índice con las entradas de que consta.

El fichero de ayuda (*.hlp) que el programa emplea se especifica en la propiedad *HelpFile* de la aplicación en tiempo de diseño a través del cuadro de diálogo que al efecto dispone Delphi en el menú **Project|Options**.

Para la creación del fichero de ayuda se ha empleado la utilidad **Help Workshop, HCW.exe**, con que cuenta Delphi3. Mediante la aplicación se genera un fichero de proyecto (*.hpj) de tipo texto, utilizado para compilar el fichero de ayuda, que contiene toda la información necesaria para ello: fichero de contenidos (*.cnt), ficheros con los temas de la ayuda y especificación de las ventanas empleadas para mostrar cada entrada de la ayuda.

El fichero de contenidos es un fichero de texto que especifica la tabla de contenidos del cuadro de diálogo de la ayuda. También dirige al sitema de ayuda para mostrar el índice y la página Buscar del mismo. En el diseño del fichero, realizado a través de Help Workshop, se especifican cabeceras, denominadas *headings*, representadas por el icono de un libro, y los temas de la ayuda, *topics*, a los que puede accederse, representados por el icono de una página. El fichero de contenidos se asocia al fichero de ayuda. Se han incluido todos los temas de que consta la ayuda salvo los relativos a los comandos de menú de la aplicación, a los que se accede exclusivamente desde el índice.

En la definición de los tipos de ventanas a emplear ha de especificarse obligatoriamente una ventana principal, *main*, y si se desea ventanas secundarias. Se ha definido únicamente la ventana principal, con lo que todas las entradas de la ayuda se muestran en ese tipo de ventana. Se ha definido su tamaño, posición, título, color de fondo, botones y comandos.

El fichero de los temas de la ayuda, denominados *topics*, es un fichero de texto en formato RTF, *Rich Text Format*. Este formato añade al texto un conjunto de parámetros

que especifican, entre otras, propiedades de los caracteres como tamaño, fuente, etc, y de los párrafos como alineación, sangría, etc. El compilador de ficheros de ayuda no reconoce a todos los parámetros, con lo que se ha eliminado del fichero a la mayoría de los mismos. Al contenido del fichero se le añade la información que precisa el compilador para generar el fichero de ayuda. Para cada tema de la ayuda se especifica un identificador y las entradas de índice desde las que acceder al mismo. Se realiza a través del parámetro *footnote* acompañado de los caracteres # y K respectivamente seguidos de su texto. .

Se han asignado varias entradas de índice a determinados temas de la ayuda. El objeto es facilitar al usuario la utilización del sistema de ayuda de forma que use el tipo de entrada de índice con el que se encuentre más familiarizado. Por ejemplo, se invierte el orden de las frases: tipos de puertas y puertas, tipos.

Se agrega a diversos temas puntos de salto, *hotspots*, que permiten acceder a un tema relacionado con el tema actual de forma directa desde el mismo.

4.2.2. SiDSen.DPR

Se trata del archivo fuente del proyecto o archivo .DPR, ***Delphi Project***. Es el fichero que al ser compilado por Delphi produce el fichero ejecutable. Delphi genera y mantiene el fichero de forma automática, y los cambios a efectuar en él se realizan a través del **Project Manager**. Contiene la relación de fichas y unidades que conforman el programa.

Toda aplicación Delphi automáticamente declara una variable *Application* como instancia de la aplicación. Las sentencias que contiene el archivo fuente del proyecto, creadas de forma automática por Delphi, emplean diversos métodos de la clase *TApplication* para inicializar la aplicación, crear las fichas de que dispone e iniciar la aplicación.

El método *Initialize* es el primero en llamarse por toda aplicación Delphi. Por defecto, tal es el caso, la llamada no realiza nada.

El método *CreateForm* crea una nueva ficha. Al llamarse desde el archivo fuente la creación de las fichas se realiza de forma autmática por parte de Delphi. Se inserta en el fichero fuente del proyecto una sentencia de este tipo por cada ficha incluida en él. El orden en que se listan es el orden en que se crean, y la primera en crearse corresponde a la ficha principal de la aplicación.

El método *Run* ejecuta la aplicación, y Delphi lo inserta automáticamente como última sentencia del fichero fuente.

5. Resultados y validación.

5.1. Introducción.

En este capítulo se muestran los resultados obtenidos tras simular diversos circuitos lógicos con el programa desarrollado, SiDSen. Estos se comparan con los que se obtienen al simular los mismos circuitos con SPICE, calculándose el error relativo para cada uno de los casos.

Los datos obtenidos corresponden en todos los circuitos estudiados al retardo estimado para el nodo crítico, que es aquel nodo que presenta el mayor tiempo de propagación entre todos los que componen el circuito.

Para cada uno de los circuitos simulados se muestra su esquemático y se describe la simulación realizada. Los resultados se presentan en tablas y gráficos, como elementos de comparación entre SiDSen y SPICE.

También se presentan dos ejemplos de traducción a SPICE y desde SPICE.

5.2. Traducción a SPICE y desde SPICE.

El primer ejemplo corresponde a la traducción desde el formato de SiDSen a SPICE. Se trata del multiplexor de la Figura 5.1, cuyo correspondiente fichero *netlist* recoge el cuadro 5.1. En el .MODEL se escriben todos los parámetros que modelan los transistores utilizados al definir los subcircuitos .SUBCKT INV1, .SUBCKT INV2 y .SUBCKT INV3, en los que se han definido relaciones de aspecto distintas β=2, β=2.2 y β=1.7 respectivamente. Análogamente se describen dos subcircuitos de puertas NOR de dos entradas. Una vez definidos los subcircuitos se describe el circuito donde se conectan los nodos según el esquemático de la Figura 5.1. El siguiente apartado muestra las tres entradas aplicadas al multiplexor, y finalmente el punto referido al análisis, que en este caso corresponde a un análisis transitorio de 0 a 1.2 ns con 0.5 ps de paso, y las salidas

seleccionadas son los nudos 7, 8, 9 y 10 del circuito. Las entradas y los parámetros que modelan a los transistores corresponden a los contenidos en los cuadros 5.2 y 5.3.

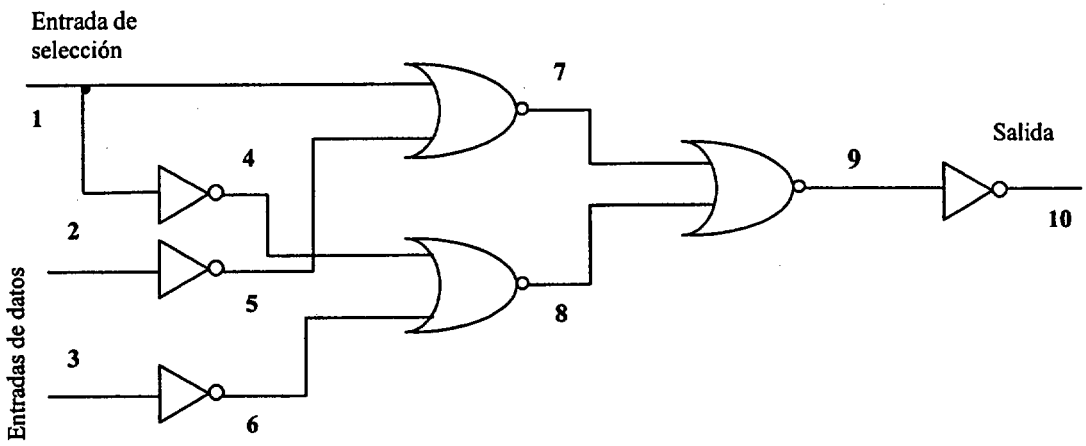


Figura 5. 1. Circuito multiplexor 2-1.

```
*comentarios
* puerta nodos β
inv 1 4 2
inv 2 5 2
inv 3 6 2
nor2 1 5 7 2
nor2 4 6 8 2
nor2 7 8 9 2
inv 9 10 2
```

Cuadro 5. 1. Netlist del multiplexor 2-1.

```
* la entrada 1 (selección)
* se encuentra a nivel bajo, 2
* recibe una transición bajo-alto y
* 3 presenta un nivel alto
* nodo pendiente retardo
1 L
2 1 0
3 h
end
```

Cuadro 5. 2. Fichero de entradas para el multiplexor 2-1.

```
vdd 1.5
tphl0 6.12
tplh0 25.41
ne 1.5
rse 6
ke 2.5
vte 0.5495
ce 2.95
rsd 5
dd 1.9
vtd -0.1
kd 1.7
vsbd 0.78
```

Cuadro 5. 3. Fichero de la tecnología.

El fichero SPICE del multiplexor 2-1 de la Figura 5.1 es:

```
**********
.MODEL EFET03 NMF VC=0.5495 VSB=5 LAMBDA=0.05 RD=600 RS=600 +RG=0.3
MFACT=1  GAMMA=4 IS=1e-13 DXL=10 VS=0.4 CDVC=0.00025
+ CDVSB=0 DELTA=0 NP=1.5 BETA=2.95 ALFA=5 FC=1 VST=1 CGS0=2e-16
+ CGS1=6e-16 VAT=-0.4 VBT=0.15 CDS=2e-16 RGS=800 RGD=800 TD=2e-12
**********
.MODEL DFET03 NMF VC=-0.1 VSB=0.78 LAMBDA=0.15 RD=500 RS=500
+RG=0.3 MFACT=2  GAMMA=2 IS=1e-14 DXL=3 VS=0.4 CDVC=0.00017
+CDVSB=0.00012 DELTA=1.9 NP=1.35 BETA=2.35 ALFA=3.5 FC=0.7 VST=1
+CGS0=1.8e-16 CGS1=4e-16 VAT=-1.1 VBT=-0.4 CDS=2e-16 RGS=900
+RGD=900 TD=2e-12
**********
.options acct opts itls=0 limpts=100000

.SUBCKT INV1 1 2
ZEI1 2 1 0 EFET03 10
ZDI1 3 2 2 DFET03 5
VDD 3 0 DC 1.5
.ENDS
.SUBCKT INV2 1 2
ZEI2 2 1 0 EFET03 11
ZDI2 3 2 2 DFET03 5
VDD 3 0 DC 1.5
.ENDS
.SUBCKT INV3 1 2
ZEI3 2 1 0 EFET03 8.5
ZDI3 3 2 2 DFET03 5
VDD 3 0 DC 1.5
```

```
.ENDS
.SUBCKT NOR21 1 2 3
ZEN11 3 1 0 EFET03 9
ZEN12 3 2 0 EFET03 9
ZDN1 4 3 3 DFET03 5
VDD 3 0 DC 1.5
.ENDS
.SUBCKT NOR22 1 2 3
ZEN21 3 1 0 EFET03 10
ZEN22 3 2 0 EFET03 10
ZDN2 4 3 3 DFET03 5
VDD 3 0 DC 1.5
.ENDS

*multiplexor de 2 entradas
X1 1 4 INV1
X2 2 5 INV2
X3 3 6 INV1
X4 1 5 7 NOR21
X5 4 6 8 NOR22
X6 7 8 9 NOR22
X7 9 10 INV3
*cadena de inversores añadida automáticamente como carga
X8 10 11 INV3
X9 11 12 INV3
X10 12 13 INV3

* ENTRADAS
Vin1 1 0 dc 0
Vin2 2 0 Pwl (0 0 100ps 0 101ps 1 1.2ns 1)
Vin3 3 0 dc 1

* ANALISIS
.tran 0.5ps 1.2ns
.print tran v(7),v(8),v(9),v(10)
.END
```

Se presenta seguidamente un ejemplo de fichero SPICE y los ficheros creados a por traducción del mismo. Se trata de un circuito constituido por una cadena de cuatro inversores.

```
**********
.MODEL EFET03 NMF VC=0.5495 VSB=5 LAMBDA=0.05 RD=600 RS=600 +RG=0.3
MFACT=1  GAMMA=4 IS=1e-13 DXL=10 VS=0.4 CDVC=0.00025
+ CDVSB=0 DELTA=0 NP=1.65 BETA=2.95 ALFA=5 FC=1 VST=1 CGS0=2e-16
+ CGS1=6e-16 VAT=-0.4 VBT=0.15 CDS=2e-16 RGS=800 RGD=800 TD=2e-12
**********
.MODEL DFET03 NMF VC=-0.1 VSB=0.78 LAMBDA=0.15 RD=500 RS=500
```

```
+RG=0.3 MFACT=2  GAMMA=2 IS=1e-14 DXL=3 VS=0.4 CDVC=0.00017
+CDVSB=0.00012 DELTA=1.9 NP=1.35 BETA=2.35 ALFA=3.5 FC=0.7 VST=1
+CGS0=1.8e-16 CGS1=4e-16 VAT=-1.1 VBT=-0.4 CDS=2e-16 RGS=900
+RGD=900 TD=2e-12
**********
.SUBCKT INV1 1 2
ZE 2 1 0 EFET03 10
ZD 3 2 2 DFET03 5
VDD 3 0 DC 1.8
.ENDS
.SUBCKT INV2 1 2
ZE 2 1 0 EFET03 11
ZD 3 2 2 DFET03 5
VDD 3 0 DC 1.8
.ENDS

* CIRCUITO
X1 1 2 INV1
X2 2 3 INV1
X3 3 4 INV1
X4 4 5 INV2
* ENTRADAS
Vin 1 0 Pwl (0 0 115ps 0 116ps 1 1.2ns 1)

* ANALISIS
.tran 0.5ps 1.2ns
.print tran v(2),v(3),v(4)
.END
```

Se muestra a continuación la información generada para cada uno de los ficheros en los cuadros 5.4, 5.5 y 5.6:

```
* nodo pendiente retardo
1 1 115
end
```

Cuadro 5. 4. Fichero de entradas.

```
* netlist cadena inversores
inv 1 2 2
inv 2 3 2
inv 3 4 2
inv 4 5 2.2
```

Cuadro 5. 5. Netlist creado a partir del fichero SPICE.

```
tphl0 6.12
tplh0
25.41
vte 0.5495
rse 6
ke 2.5
ne 1.65
ce 2.95
vtd –0.1
vsbd 0.78
rsd 5
kd 1.7
dd 1.9
vdd 1.8
```

Cuadro 5. 6. Fichero de la tecnología.

5.3. Simulaciones.

Se han realizado simulaciones sobre los siguientes circuitos: cadenas de inversores de diferentes tamaños, un circuito multiplicador 2x2, un multiplexor 2-1, un circuito sumador de acarreo anticipado, un comparador y un circuito de alarma. En este apartado se describen las simulaciones ejecutadas y se recogen sus resultados.

5.2.1. Cadenas de inversores.

Se han realizado simulaciones sobre diversas cadenas de inversores de distinta longitud.

Figura 5. 2. Cadena de n inversores.

El primer conjunto de simulaciones se ha efectuado sobre cadenas de inversores de diverso tamaño aplicando como señal de entrada una transición desde el nivel bajo al alto.

Todos los inversores tienen la misma relación de aspecto $\beta=2$, que para las simulaciones SPICE se convierte en anchuras de 10 para el transistor de enriquecimiento y 5 para el de deplexión. Los parámetros de modelado utilizados mantienen sus valores nominales. La tabla 5.I contiene los valores obtenidos mediante simulaciones SPICE y SiDSen para cada cadena, así como el error relativo cometido en cada caso. De los datos que muestra la tabla se concluye que el error máximo es de un 3.1%.

Tabla 5. I.

Nº de inversores	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
14	221.5	228.18	3
24	379.5	391.17	3.06
34	537.45	554.17	3.1
44	695.45	717.16	3.1
54	853.9	880.15	3.07
64	1011.8	1043.14	3.09
100	1580.6	1629.91	3

La figura 5.3 muestra el retardo (ps) frente al número de inversores. Los datos de SiDSen se representan mediante una línea y los de SPICE a través de puntos.

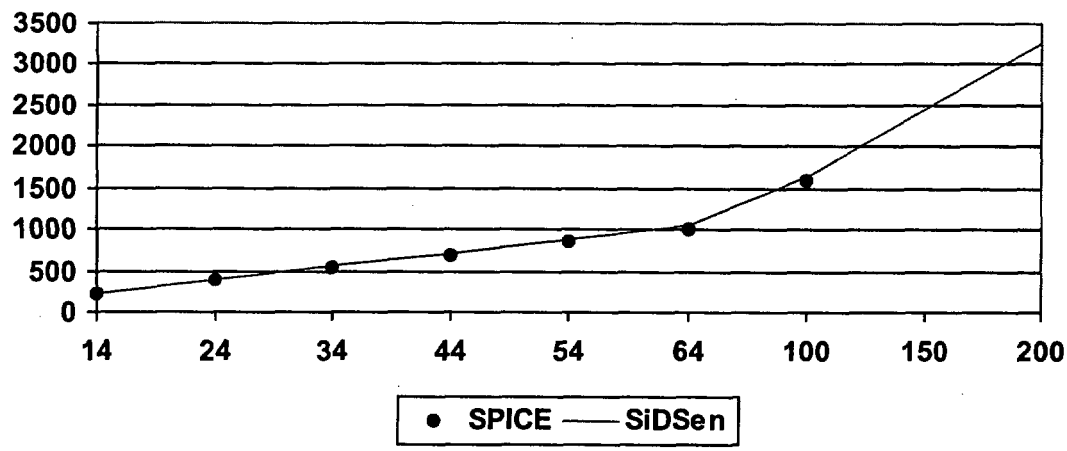


Figura 5. 3.

El segundo grupo de simulaciones se realiza sobre la cadena de 24 inversores. Se aplica la misma señal de entrada que en los casos precedentes, transición de 0 a 1, y se juega con el valor de la tensión de alimentación, manteniendo los restantes parámetros del modelo sus valores nominales. En la tabla 5. II se muestran los resultados de dichas simulaciones en SPICE y con SiDSen, y el error calculado.

Tabla 5. II.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	382.65	380.94	0.45
1.3	382.2	388.76	-1.71
1.4	381	391.75	-2.82
1.5	379.5	391.17	-3.08
1.6	378.05	388.28	-2.7
1.7	376.9	384.31	-1.96
1.8	376.15	380.54	-1.18

En la figura 5.4 se representan gráficamente los datos de la tabla 5. II, quedando los valores de SPICE marcados mediante puntos, mientras que la línea continua representa a los resultados de SiDSen.

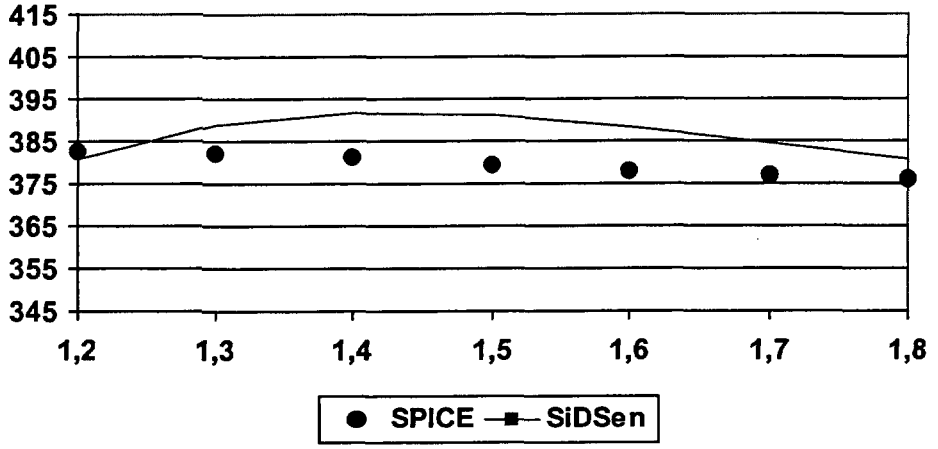


Figura 5. 4.

Como puede verse el error máximo cometido con respecto a SPICE es del 3%.

Las últimas simulaciones se llevan a cabo sobre una cadena de 100 inversores. En este caso el parámetro que se hace variar de una simulación a otra es el valor de la relación de aspecto, β, de cada uno de los inversores que componen el circuito, asignando el mismo valor a todos. Los restantes parámetros presentan valores nominales. Al llevar las simulaciones a SPICE la anchura del transistor de deplexión, W_D, se fija a 5, obteniéndose la anchura correspondiente al transistor de enriquecimiento, W_E, aplicando la ecuación 2.1. La tabla 5.III recoge los resultados obtenidos y el error calculado para cada caso.

Tabla 5. III.

β	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	1393.4	1400.07	-0.47
1.6	1478	1514.99	-2.5
2	1580.6	1629.91	-3.11
2.4	1691.9	1744.83	-3.12
2.8	1807.4	1859.74	-2.89

En la figura 5.5 se representan los datos SPICE en puntos y en línea continua los correspondientes a SiDSen.

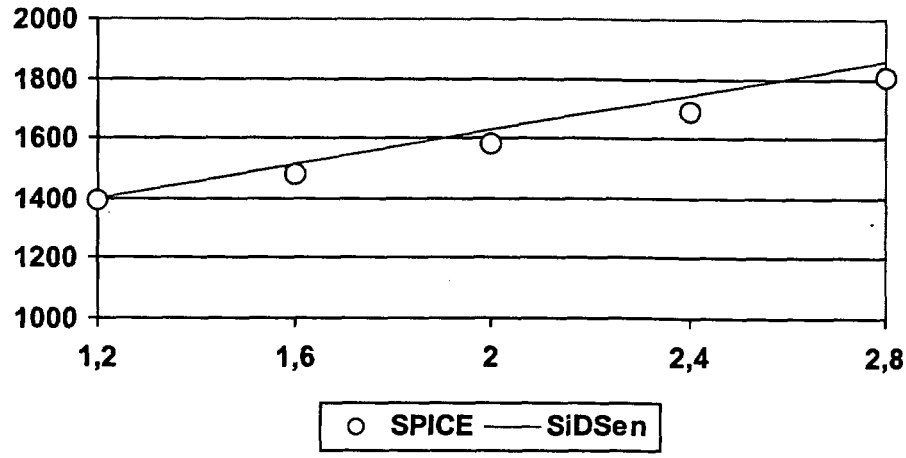


Figura 5. 5.

El error máximo calculado para la cadena de 100 inversores respecto a SPICE es de un 3.1%.

5.2.2. Multiplexor 2-1.

El circuito multiplexor 2-1 consta de dos entradas de datos, A (1) y B (2), una entrada de selección, S (3), y una salida (10). Su esquemático puede verse en la siguiente figura.

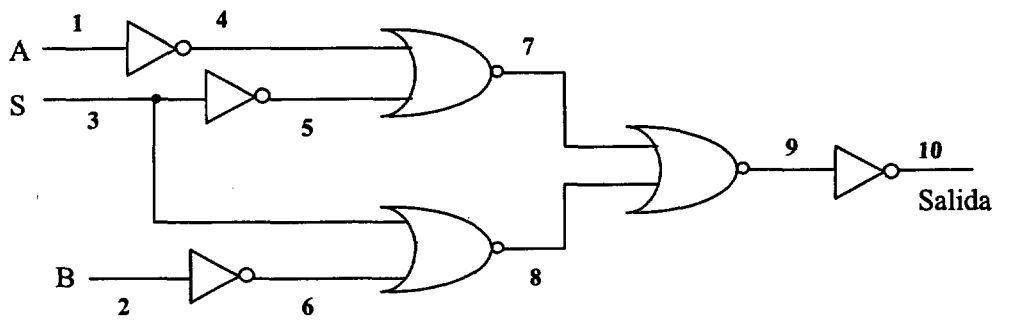


Figura 5. 6. Circuito multiplexor 2-1.

En las simulaciones realizadas la entrada de selección, S, se mantiene a nivel alto, con lo que a la salida se lleva el valor de A. La entrada B se encuentra a nivel alto. Se ejecutan simulaciones con SPICE y SiDSen para diversos valores de la tensión de alimentación, modificando la entrada A. Los restantes parámetros de modelado se fijan a sus valores nominales. A todas las puertas del circuito se les asigna el valor nominal de la relación de aspecto β=2.

Se simula en la entrada A una señal que pasa del nivel bajo al alto. Los resultados obtenidos y el error relativo cometido en cada caso se presentan en la tabla 5.IV.

Tabla 5. IV.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	81.55	83.15	1.96
1.3	82.5	84.06	1.89
1.4	83.15	84.32	1.4

1.5	83.4	84.09	0.82
1.6	83.5	83.55	0.05
1.7	83.6	82.86	-0.88
1.8	83.65	82.2	-1.73

La figura 5.7 representa gráficamente el contenido de la tabla 5.IV. Los valores correspondientes a SPICE se muestran en forma de puntos, mientras que los obtenidos mediante SiDSen se presentan como una línea continua.

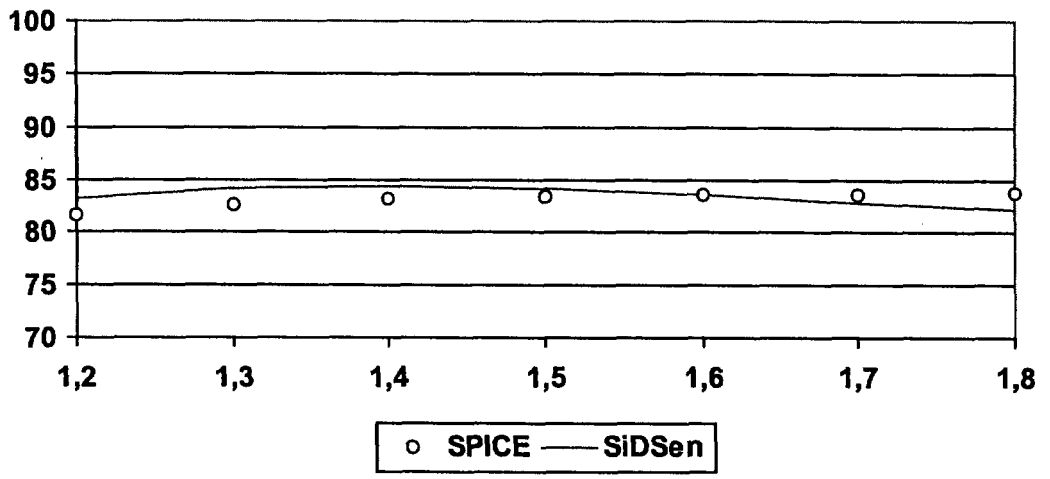


Figura 5. 7.

De los datos que arroja la tabla 5.IV se desprende que el error relativo máximo es del 1.9% para el retardo en el nodo crítico (nodo 10).

5.2.3. Multiplicador 2x2.

El circuito multiplicador 2x2 multiplica dos palabras de dos bits, A y B. Dispone de cuatro salidas, P3 a P1. En las simulaciones realizadas A presenta el valor 2, y B pasa de un valor inicial 3 a 1. De esta forma la entrada 4, B1, presenta una transición desde el nivel lógico alto al bajo. El nodo crítico corresponde a la salida P2, nodo 25, que transita de alto a bajo. Se realizan diversas simulaciones empleando distintos valores para la tensión de alimentación, V_{DD}, y manteniendo el resto de los parámetros de modelado su valor nominal. Todas las puertas que forman el circuito tienen una relación de aspecto nominal,

es decir $\beta = 2$. En las simulaciones SPICE esta relación de aspecto se obtiene fijando los valores de las anchuras de los transistores: W_E=10 y W_D=5. La figura 5.8 muestra el esquemático del circuito multiplicador.

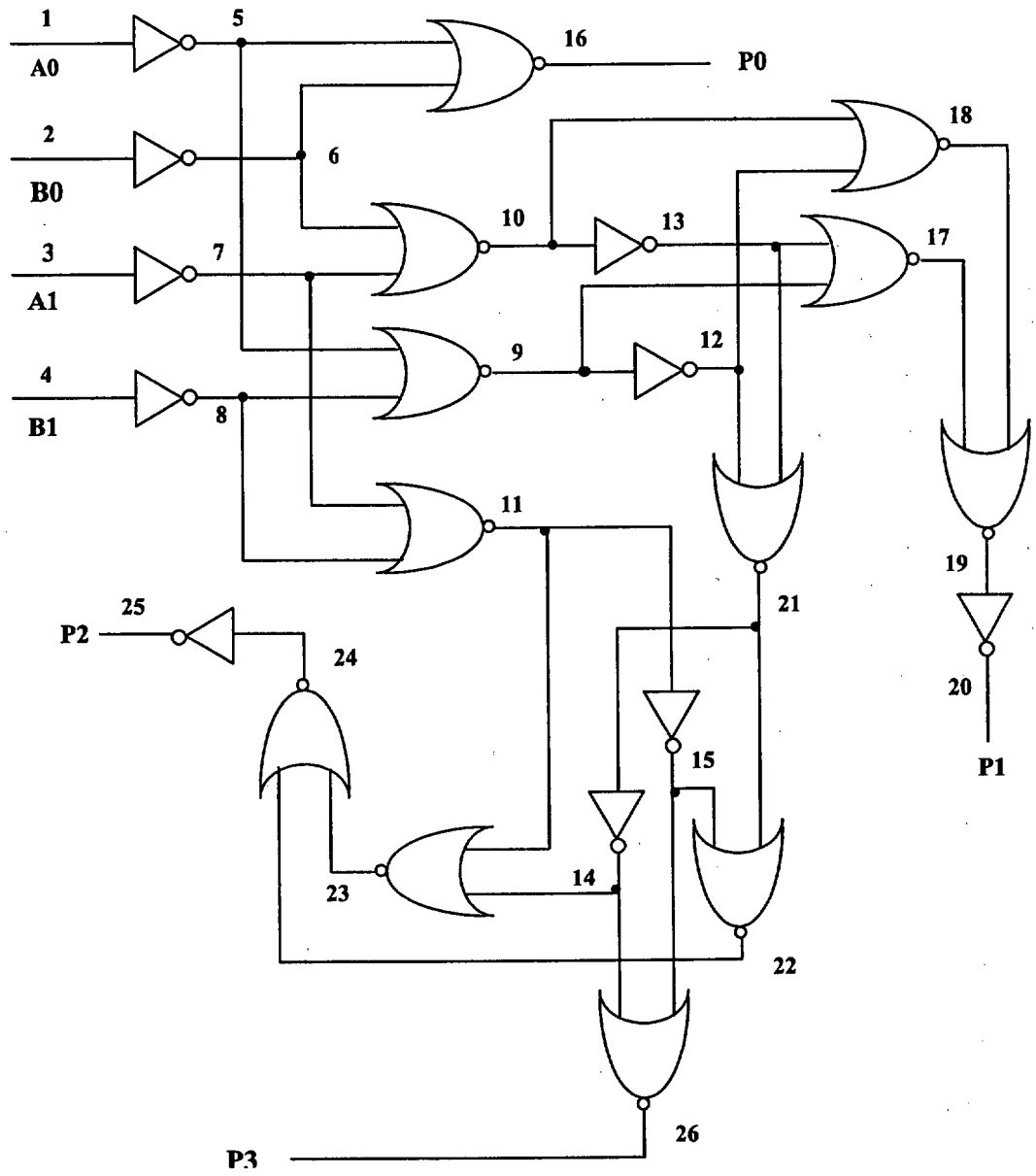


Figura 5. 8. Multiplicador 2x2.

La tabla 5.V ofrece los valores obtenidos de las simulaciones SPICE y los que genera SiDSen, así como el error relativo cometido.

Tabla 5. V.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	122.75	117.58	4.21
1.3	123.25	118.75	3.65
1.4	123.15	119.01	3.36
1.5	122.85	118.61	3.45
1.6	122.5	117.76	3.86
1.7	122.2	116.72	4.48
1.8	121.9	115.71	5.07

En la figura 5.9 están marcados con puntos los valores obtenidos a través de SPICE y con una línea continua la representación gráfica de los valores de SiDSen. El eje vertical contiene el tiempo de propagación en picosegundos, y el eje horizontal la tensión de alimentación en voltios.

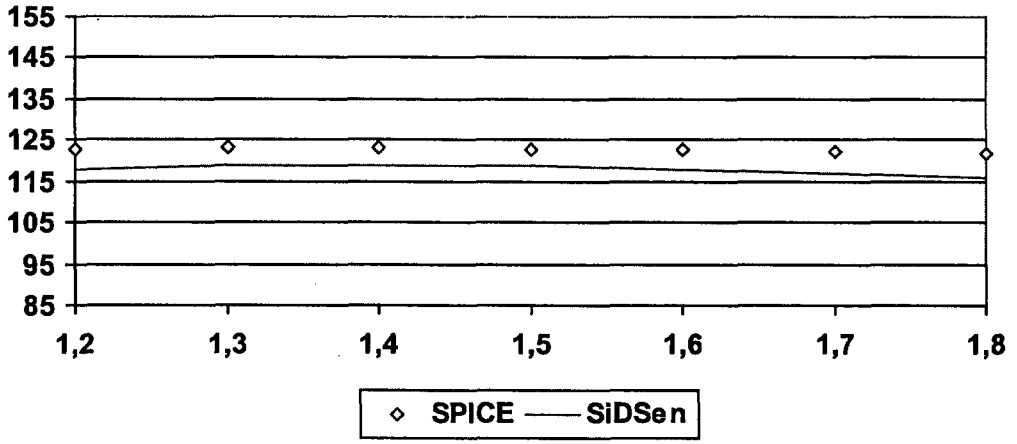


Figura 5. 9.

El error relativo máximo calculado respecto a los resultados de las simulaciones SPICE es de un 5%.

5.2.4. Circuito de alarma.

El circuito de alarma dispone de 6 entradas y una salida en el nodo 22. En la figura 5.10 se muestra el esquemático del circuito.

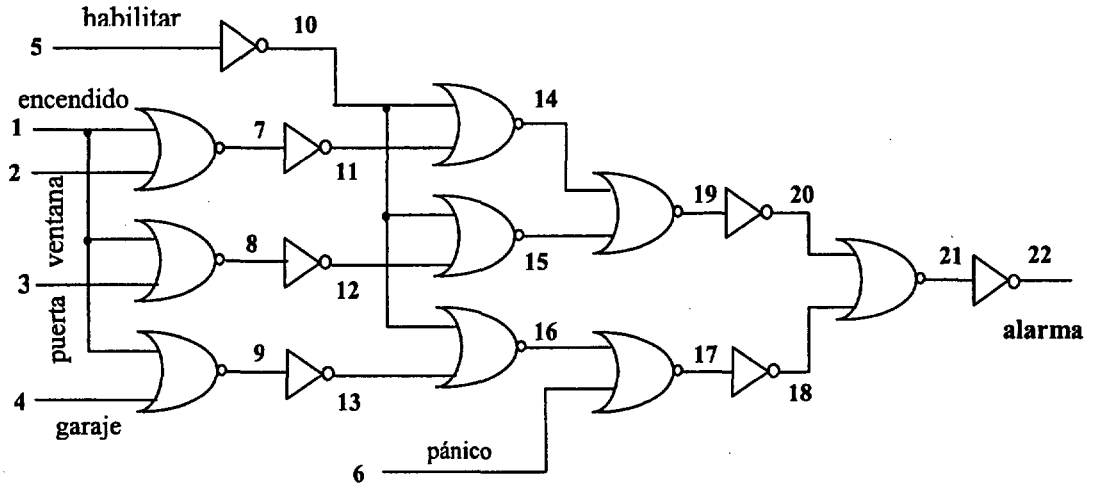


Figura 5. 10. Esquemático del circuito de alarma.

Hay dos casos en los que la salida del circuito es 1. Uno se produce cuando la entrada "pánico", nodo 6, es 1. El segundo caso se da cuando la entrada habilitadora, nodo 5, es 1, la entrada encendido, nodo 1, se mantiene a nivel bajo, y una de las otras tres entradas, "ventana", nodo2, "puerta", nodo 3, o "garaje", nodo 4, también es el 0 lógico. Todas las puertas definidas en el netlist tienen una relación de aspecto nominal β=2. Con este circuito se calcula el retardo con variaciones en el valor de la tensión de alimentación, V_{DD}, y se conserva el valor nominal para los restantes parámetros de modelado.

En el primer grupo de simulaciones la entrada habilitadora está a nivel alto y el encendido a nivel bajo. La entrada correspondiente a la ventana presenta una transición alto-bajo, con lo que se activa la salida del circuito. El camino crítico, por lo tanto, enlaza a los nodos 2 y 22. El contenido de la tabla 5.VI corresponde a los resultados de dichas simulaciones, junto al error cometido en cada caso.

Tabla 5. VI.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	164.7	161.49	1.9
1.3	167.75	162.26	3.3
1.4	169.9	162.12	4.6
1.5	171.25	161.31	5.8
1.6	171.95	160.07	6.9
1.7	172.4	158.63	7.9
1.8	172.65	157.22	8.9

Los datos de la tabla se representan en la figura 5.11, donde la línea continua corresponde a los tiempos estimados por SiDSen y los puntos representan los resultados de SPICE.

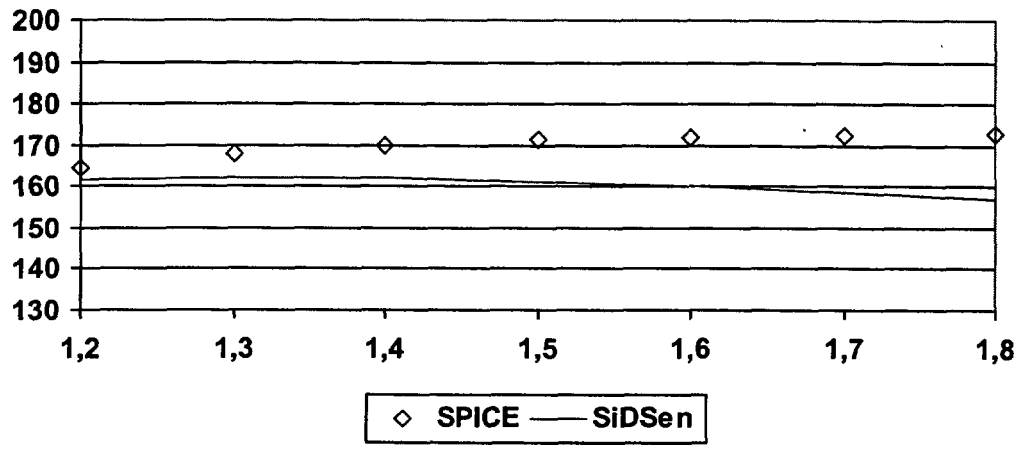


Figura 5. 11.

De los datos que recoge la tabla se desprende que el error relativo máximo es del 8.9% para el retardo en el nodo crítico.

El segundo conjunto de simulaciones se realiza desactivando la entrada de habilitación y provocando la alarma al aplicar una transición desde el nivel bajo al alto al nodo 6, que corresponde a la entrada "pánico". El camino crítico va desde este nodo a la

salida. En la tabla 5.VII se consignan los resultados obtenidos en las simulaciones y el error calculado para cada una de ellas.

Tabla 5. VII.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	53.8	52.9	1.67
1.3	54.1	60.426	-11.68
1.4	54.05	60.43	-11.8
1.5	53.85	60.07	-11.55
1.6	53.7	59.47	-10.74
1.7	53.5	58.76	-9.83
1.8	53.35	58.07	-8.84

En la figura 5.12 se muestran como puntos los valores obtenidos a través de SPICE y con una línea continua se representan los resultados que genera SiDSen.

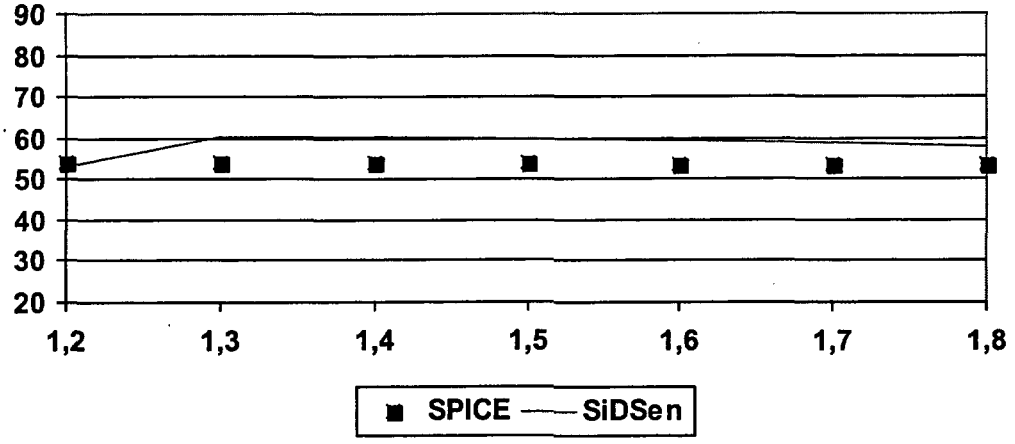


Figura 5. 12.

El error máximo calculado respecto a SPICE para el nodo crítico es en este caso de un 11.8%.

5.2.5. Comparador.

El circuito compara dos palabras de 4 bits. La salida, nodo 79, adopta el 1 lógico si la palabra A, a3-a0 nodos 1 a 4, es mayor que la palabra B, b3-b0 nodos 5 a 8

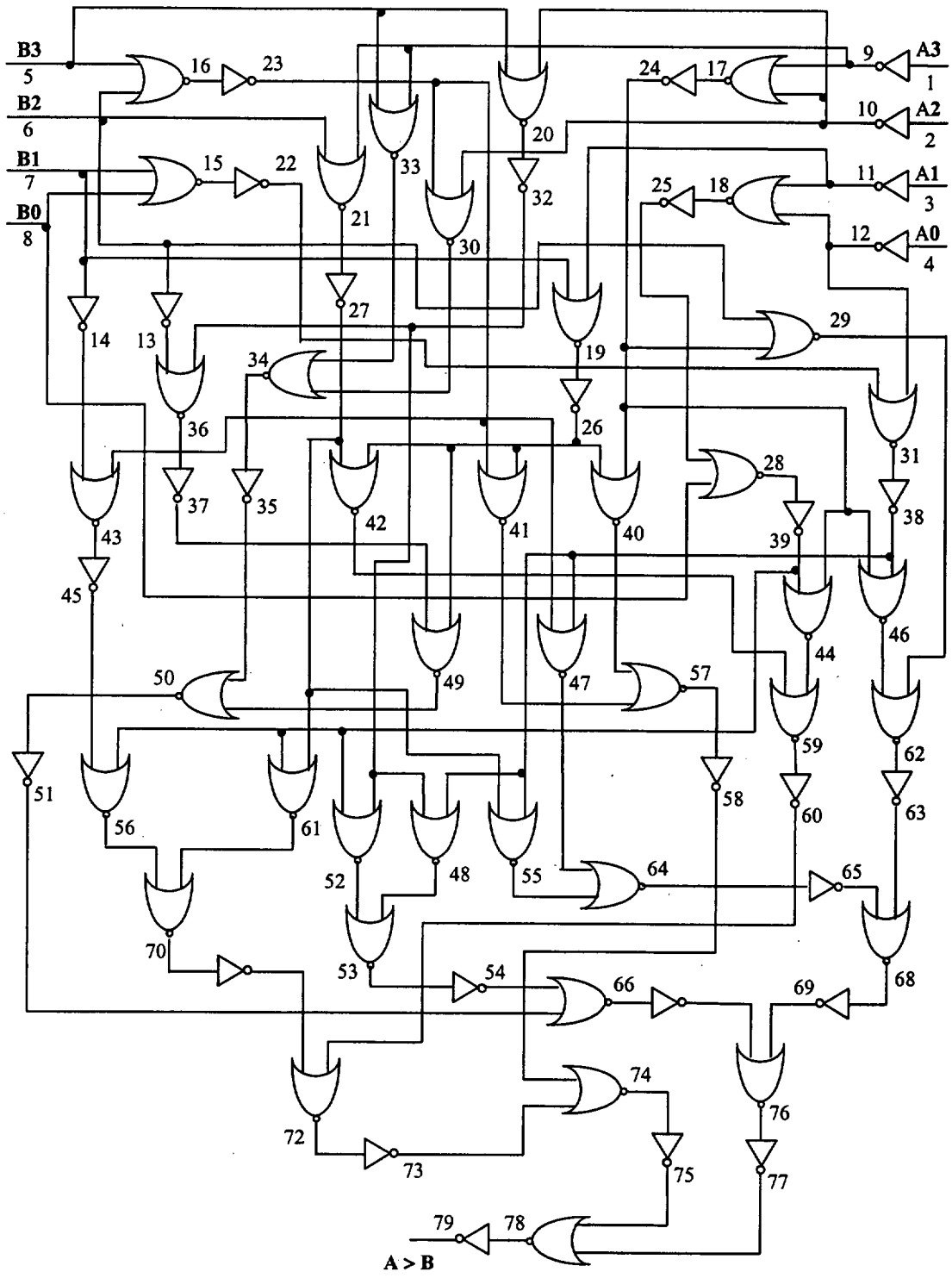


Figura 5. 13. Circuito comparador de 4 bits.

En las simulaciones efectuadas la palabra A será 10. En B el bit de mayor peso, b3, nodo 5, transita desde el nivel bajo al alto, permanecen a nivel bajo los bits b2 y b0, y se mantiene a nivel alto b1. De esta forma B alcanza el valor 10, con lo que la salida transita de 1 a 0 al ser las dos palabras iguales. Se realizan simulaciones modificando el valor de la tensión de alimentación, V_{DD}, manteniéndose el resto de los parámetros de modelado en sus valores nominales. La tabla 5.VIII contiene los resultados obtenidos para el nodo crítico, nodo 79, y el error calculado en cada caso.

Tabla 5. VIII.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	290.4	269.04	7.3
1.3	293.1	272.96	6.8
1.4	296.65	274.61	7.4
1.5	299.25	274.59	8.2
1.6	301.1	273.49	9.1
1.7	301.05	271.88	9.6
1.8	300.95	270.36	10.1

En la figura 5.14 se representan gráficamente los valores de la tabla 5.VIII, marcándose mediante puntos los resultados de SPICE y con una línea continua los de SiDSen.

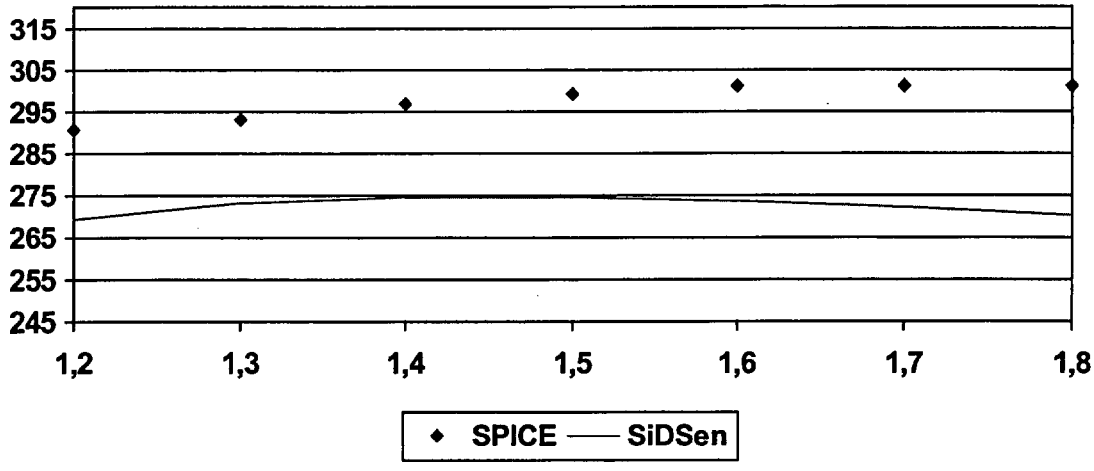


Figura 5. 14.

El error máximo calculado respecto a SPICE para el nodo crítico es de un 10.1%.

5.2.6. Sumador de acarreo anticipado.

En la figura 5.15 puede verse el esquemático del sumador de acarreo anticipado.

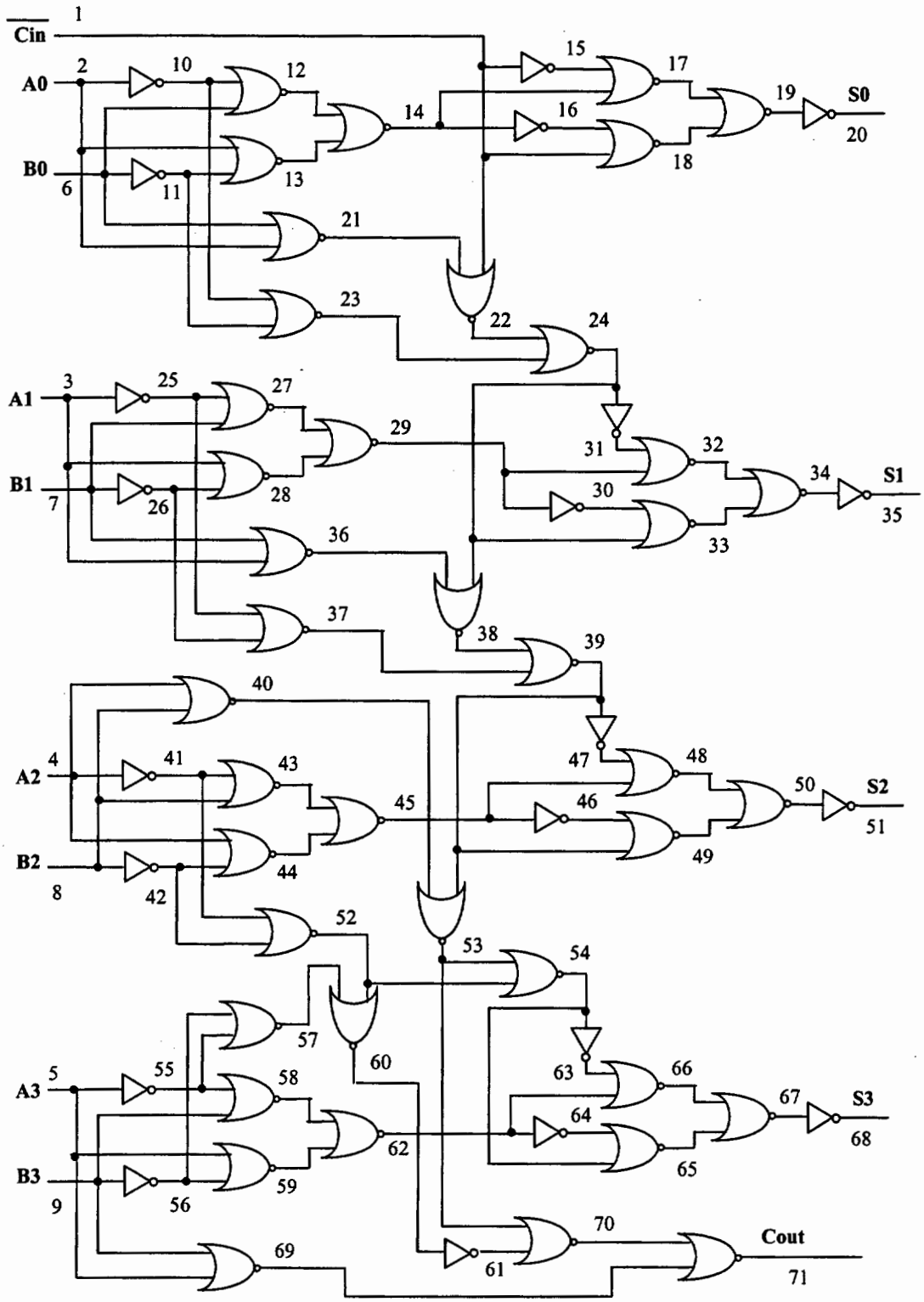


Figura 5. 15. Circuito sumador de acarreo anticipado.

El circuito sumador de acarreo anticipado suma dos palabras de cuatro bits, A, nodos 5 a 2, y B, nodos 9 a 6. Consta de entrada de acarreo, $/C_{in}$, activa a nivel bajo. Dispone de cuatro salidas, S3 (68), S2 (51), S1 (35) y S0 (20), y de una salida de acarreo, C_{out}, nodo 71.

En las simulaciones efectuadas la entrada de acarreo permanece a nivel alto. La palabra A será 12. En B el bit de más peso, B3, permanece a nivel bajo, los dos siguientes, B2 y B1, se mantienen a nivel alto, mientras que B0 presenta transición. Se realizan variaciones sobre el valor de la tensión de alimentación, V_{DD}, manteniéndose el resto de los parámetros de modelado en sus valores nominales. Todas las puertas definidas en el circuito presentan un valor de relación de aspecto 2. Para las simulaciones a realizar en SPICE se logra dicha relación de aspecto fijando la anchura del transistor de enriquecimiento a 10, y 5 a la anchura del transistor de deplexión.

En el primer conjunto de simulaciones realizadas B0 recibe una transición desde el 0 lógico al 1. Los retardos estimados para el nodo crítico (nodo 20, salida S0) se muestran en la tabla 5.IX.

Tabla 5. IX.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	157.55	138.76	11.92
1.3	157.95	139.93	11.4
1.4	158.05	140.19	11.3
1.5	158	139.79	11.52
1.6	157.85	138.94	11.9
1.7	No disponible	137.9	------
1.8	157.6	136.89	13.14

En la figura 5.16 se representan los datos SPICE en puntos y en línea continua los correspondientes a SiDSen.

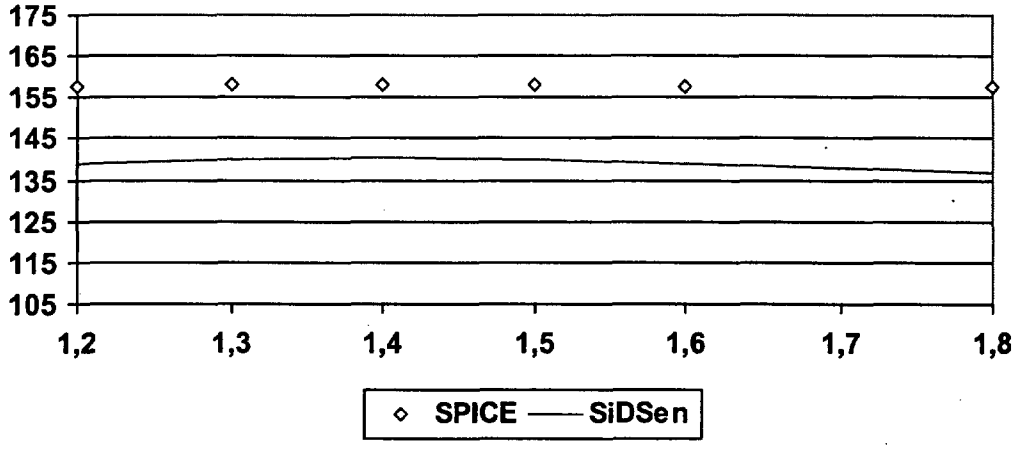


Figura 5. 16.

Del análisis de los tiempos estimados se infiere que se comete un error máxima relativo del 13.1% sobre la medida en el nodo crítico.

Las siguientes simulaciones se realizan asignando a la entrada B0 una transición desde el nivel lógico alto al bajo. Sus resultados se recogen en la tabla 5.X.

Tabla 5. X.

V_{DD} (V)	t_p SPICE (ps)	t_p SiDSen (ps)	Error (%)
1.2	126.65	136.94	8.12
1.3	127.2	138.11	8.57
1.4	127.15	138.37	8.82
1.5	126.8	137.97	8.8
1.6	126.45	137.12	8.43
1.7	126.1	136.08	7.91
1.8	125.75	135.07	7.41

En la figura 5.17 están representados mediante puntos los valores que se obtienen simulando con SPICE, la línea continua representa los resultados retardos calculados a través de las simulaciones con SiDSen.

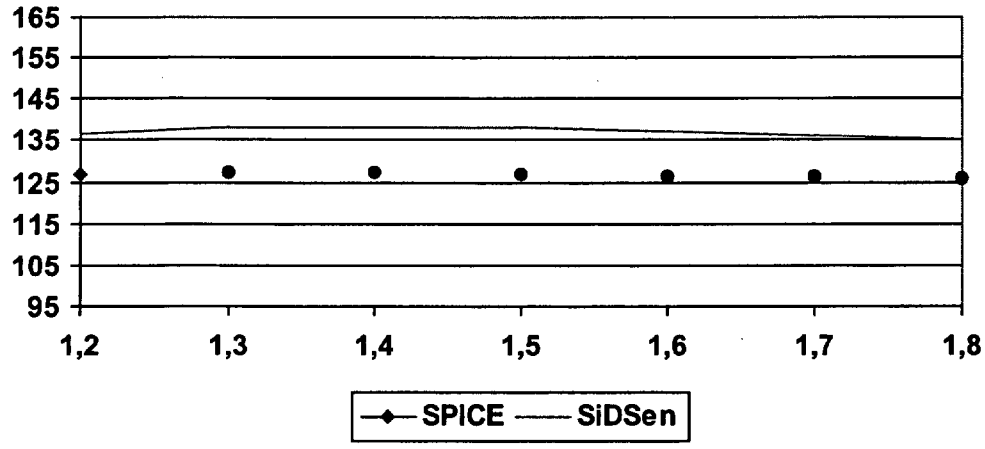


Figura 5. 17.

El error máximo que se comete en este caso es del 8.8%.

A continuación, y a modo de ejemplo, se explica el proceso de simulación llevado a cabo por SiDSen sobre el circuito sumador de acarreo anticipado. Se parte de la señal de entrada B0 que cambia desde el nivel lógico alto al bajo, con V_{DD}=1.5 V. El camino crítico que recorre la señal es el que enlaza a los nodos 6, entrada B0, y 20, salida S0.

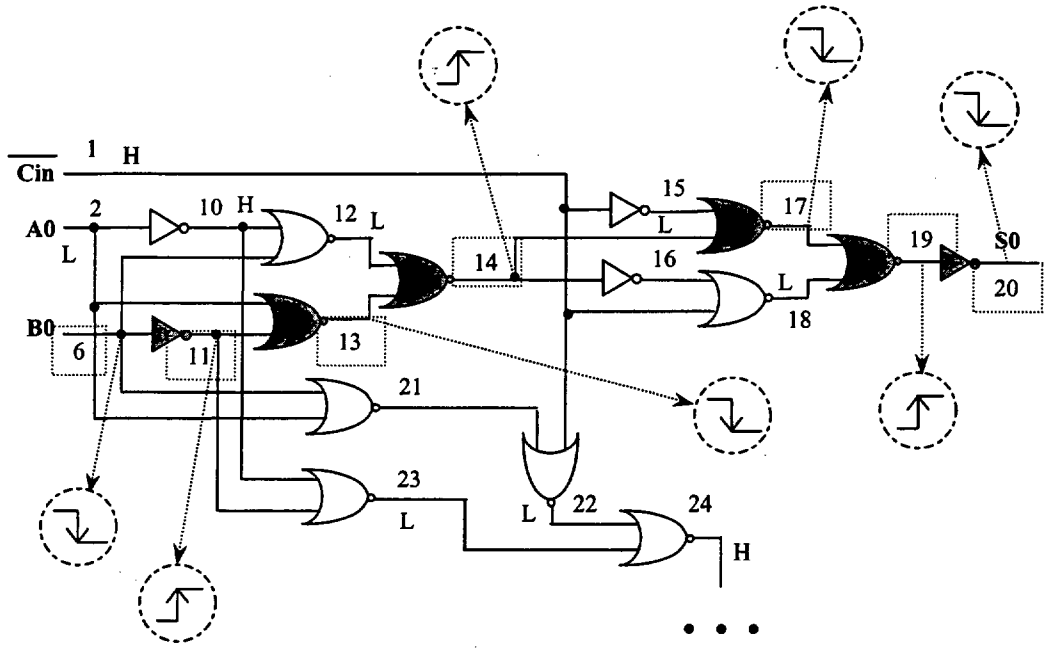


Figura 5. 18. Camino crítico en el sumador de acarreo anticipado.

En el esquemático de la figura 5.18 se ha marcado el camino crítico resaltando los nodos en recuadros punteados y sombreando las puertas que lo componen. También se

señalan las transiciones que se producen y los niveles constantes en los distintos nodos del circuito. El camino crítico consta de 6 puertas. Para cada una de ellas se enumeran las condiciones bajo las que se simula y se realiza el cómputo del tiempo de propagación. Todas las reseñas a ecuaciones que se emplean en este análisis se refieren al capítulo 2 de la memoria.

1ª puerta: inversor entre los nodos 6 y 11, conectado a la entrada B0 y relación de aspecto β=2. Se le aplica una señal de entrada que transita de 1 a 0, luego la salida lo hace de 0 a 1. A su salida se conectan dos puertas por lo que FO=2, con lo que SiDSen emplea la ecuación 2.12 → t_{pLH}=26.91 ps.

2ª puerta: NOR2 con entradas en los nodos 2, que se mantiene a nivel bajo, y 11 que transita de bajo a alto, tiene en el nodo 13 su salida que sólo se conecta a otra puerta, con lo que FO=1. El tiempo de propagación se calcula mediante un inversor equivalente de β_{eq}=2.3, y empleando la ecuación 2.8 → t_{pHL}=4.06 ps.

3ª puerta: NOR2 con entradas en 12, a nivel bajo, y 13, transición de 1 a 0 y la salida, al nodo 14, se conecta a dos puertas, FO=2. Se calcula el retardo mediante la ecuación 2.12, como un inversor equivalente de β_{eq}=3.5 → t_{pLH}=48.52 ps.

4ª puerta: NOR2, entradas en 14, transición de bajo a alto, y 15, constante a nivel bajo. Su salida es el nodo 17, conectado a una sola puerta, FO=1. Se recurre a un inversor equivalente de β_{eq}=2.3 para calcular el retardo, y se utiliza la ecuación 2.8 → t_{pHL}=4.06 ps.

5ª puerta: NOR2 con entradas en 17, que transita desde el 1 lógico al 0, y 18, que presenta un nivel bajo constante. La salida, nodo 19, se conecta a una sola puerta, dando lugar a FO=1. Se utiliza la ecuación 2.9 para calcular el tiempo de propagación de la puerta mediante un inversor equivalente de β_{eq}=2.3 → t_{pLH}=47.33 ps.

6ª puerta: inversor entre los nodos 19 y 20, salida S0 y β=2. La entrada presenta una transición desde el nivel bajo al alto, con lo que la salida va de alto a bajo. Al tratarse

de una salida del circuito se considera que FO=1. Se calcula el retardo usando la ecuación 2.8 → t_{pHL}= 7.08 ps.

El tiempo de propagación correspondiente al camino crítico se obtiene sumando los tiempos individuales de las puertas que forman el camino → $\mathbf{t_{pB0 \rightarrow S0}}$**=137.97 ps**.

6. Conclusiones.

Se ha desarrollado un programa, SiDSen, para la simulación, bajo el entorno Windows, de la respuesta temporal de la familia lógica DCFL implementada mediante HFETs en Arseniuro de Galio. Los tiempos de propagación en los distintos nodos de los circuitos se estiman a través del modelo temporal basado en el análisis de la sensibilidad, presentado en el Capítulo 2. Se ha logrado de esta forma una aplicación con un bajo coste en cuanto al tiempo de computación que requiere, que se sitúa alrededor de un segundo en todos los circuitos analizados. Del estudio presentado en el Capítulo 5 se desprende que la precisión de los tiempos estimados respecto a SPICE es comparable a otros modelos existentes y el error máximo cometido es inferior al 13.2% en todos los casos estudiados.

El programa se ha diseñado para que contemple la posibilidad de modificar externamente las ecuaciones de modelado lo que permite incluir nuevos parámetros y coeficientes, constituyendo así una herramienta de apoyo en el desarrollo del modelo. De la misma forma se pueden incorporar nuevos modelos temporales de otras tecnologías.

SiDSen ofrece la opción de generar un fichero SPICE. Esto permite no sólo la validación rápida del circuito estimado con una herramienta ampliamente utilizada, sino prediseñar complejos circuitos de una forma simple.

Se ha enviado como aportación al DCIS2001, XVI Conference on Design of Circuits and Integrated Systems, Porto, Portugal, November 2001.

A continuación se destacan una serie de posibles trabajos a desarrollar en el futuro prorrogando la línea seguida.

Si se introducen nuevos parámetros de modelado, como la temperatura, no modelada hasta ahora por SPICE, con objeto de disponer de unas expresiones más completas se debería actualizar el programa incorporando estas nuevas ecuaciones. También puede conllevar una actualización de las expresiones la obtención de ecuaciones válidas para todos los valores de fan-out. Se ha logrado una expresión única para t_{pLH}, pero

por simetría no se aplicó en el programa. La utilización de ecuaciones externas permite usar estas expresiones actualmente y progresar en su desarrollo.

Se está trabajando en la implementación del modelo eléctrico del HFET en un simulador que opere bajo Windows, PSPICE o APLAC, con lo que puede resultar interesante modificar los traductores de SiDSen para que funcionen con estos programas de simulación.

Otro trabajo futuro sería desarrollar un programa que calculase el modelo temporal basado en el análisis de la sensibilidad sobre otros transistores.

7. Presupuesto del proyecto.

Para el cálculo del presupuesto del proyecto, se ha seguido la **Propuesta de baremos orientativos para el cálculo de honorarios** del Colegio Oficial de Ingenieros Técnicos de Telecomunicación.

Esta propuesta establece que para Trabajos tarifados por tiempo empleado se aplique la siguiente ecuación:

$$H = Hn \cdot 9700 + He \cdot 10500$$

Donde:

H: Honorarios

Hn: Horas en jornada normal

He: Horas fuera de la jornada normal

Estos Honorarios tendrán una reducción en función del número de horas aplicando los coeficientes de reducción de la siguiente tabla.

Los honorarios que se obtengan por aplicación de la clave H se reducirán a medida que aumenta el número de horas, a cuyo efecto serán multiplicados por los coeficientes reductores con arreglo a la siguiente tabla.

Tabla 8. I. Coeficientes reductores.

		COEFICIENTE
Hasta 36 horas		C = 1
Exceso de 36 horas	Hasta 72	C = 0,9
Exceso de 72 horas	Hasta 108	C = 0,8
Exceso de 108 horas	Hasta 144	C = 0,7
Exceso de 144 horas	Hasta 180	C = 0,65
Exceso de 180 horas	Hasta 360	C = 0,6

Exceso de 360 horas	Hasta 540	C = 0,55
Exceso de 540 horas	Hasta 720	C = 0,5
Exceso de 720 horas	Hasta 1080	C = 0,45
Exceso de 1080 horas		C = 0,4

Ahora, particularizando para el proyecto que aquí se dispone, se establece una tabla indicativa acerca del tiempo empleado para el desarrollo del mismo.

Tabla 8. II. Tiempos parciales empleados en el desarrollo del proyecto.

UNIDADES	DESCRIPCIÓN	PARCIAL
Horas	Estudio de la documentación sobre el modelo	30
Horas	Aprendizaje del lenguaje Delphi	150
Horas	Diseño, implementación y depuración del programa	850

En definitiva, se necesitaron un total de 1030 horas, que se consideran en su totalidad del tipo de jornada normal, con lo que el cálculo de la clave H resulta:

$$H = 1030 \cdot 9700 = 9.991.000 \text{ ptas.} \quad 60.047,12 \text{ euros}$$

Aplicando el coeficiente de corrección según la tabla anterior se tiene:

$$H = 9.991.000 \cdot 0,45 = 4.495.950 \text{ ptas.} \quad 27.021,2 \text{ euros}$$

El coste global del proyecto se obtiene sumando a dicha cantidad el precio del material empleado.

Tabla 8. III. Precio del material empleado.

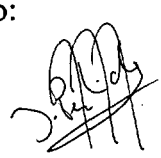
UNIDADES	DESCRIPCIÓN	PESETAS	EUROS
1	Soporte software: Delphi3	146.900	882,88
1	Equipo PC	212.000	1247,15

$$C.G. = H + 146.900 + 212.000 = 4.854.850 \text{ ptas.} \quad 29.178,23 \text{ euros}$$

El precio de coste global del proyecto asciende a CUATRO MILLONES OCHOCIENTAS CINCUENTA Y CUATRO MIL OCHOCIENTAS CINCUENTA

PESETAS (4.854.850 pesetas), VEINTE Y NUEVE MIL CIENTO SETENTA Y OCHO EUROS CON VEINTITRES CÉNTIMOS (29.178,23 euros).

Firmado:



Nombre: Juan Marcos Pulido González.

8. Bibliografía

[1] J.García, A. Hernández, B. González, J. del Pino and A. Nuñez. Analysis Model based on Sensitivity for DCFL family with HFETs. XII Design of Circuits and Integrated Systems Conference DCIS'97 pp 453-457 Sevilla, November 1997.

[2] J. García, J. del Pino, A. Hernández, B. González, and A. Nuñez. Application of sensitivity analysis in modelling power and delay for HFETs DCFL circuits. VII Power and timing Modelling, Optimization and Simulation PATMOS'98. Lyngby, Denmark. October 1998. pp 51-60.

[3] M. S. Shur, GaAs Devices and Circuits. Plenum Press, 1990.

[4] Design Manual v.3.0. Fraunhofer Institute for Applied Solid State Physics. March94.

[5] Design Manual v.4.0. Fraunhofer Institute for Applied Solid State Physics. January97.

[6] Javier García, Juan Pulido, Antonio Hernández, Javier del Pino, Benito González, José Sendra and Antonio Núñez. SiDSen: A Program to simulate delays based on sensitivity analysis model. XVI Conference on Design of Circuits and Integrated Systems DCIS2001. Porto, Portugal. November 20-23, 2001 (aceptado para revisión).

[7] Francisco Charte Ojeda . Programación con Delphi 3.0. Madrid, Anaya, 1997.

[8] Ian Marteens. Manual fundamental de Borland Delphi 2 para Windows 3.1 y Windows95/NT. Madrid, Anaya Multimedia, 1996.

[9] Marco Cantú. La biblia de Delphi 3. Madrid, Anaya Multimedia, 1998.

[10] Luis Joyanes Aguilar. Programación en Turbo Pascal versiones 5.5, 6.0 y 7.0. Mc Graw-Hill, 1993.

Anexo

```
unit cuadro_t;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls;

type
  TTAS = class(TForm)
    AceptBtn: TBitBtn;
    canclBtn: TBitBtn;
    Edittran: TEdit;
    Editprint: TEdit;
    tranlabel: TLabel;
    printlabel: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Edit2: TEdit;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    Bevel5: TBevel;
    Bevel6: TBevel;
```

```
    Bevel7: TBevel;
    Bevel8: TBevel;
    dialogante: TOpenDialog;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  TAS: TTAS;

implementation

{$R *.DFM}

procedure TTAS.Button1Click(Sender: TObject);
begin
  dialogante.FilterIndex:=1;
  if dialogante.Execute then
    Edit1.Text:=dialogante.FileName;
end;

procedure TTAS.Button2Click(Sender: TObject);
begin
  dialogante.FilterIndex:=2;
  if dialogante.Execute then
    Edit2.Text:=dialogante.FileName;
end;



end.
```

```
unit cuadro_sim;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 ExtCtrls, StdCtrls, Buttons;

type
 TDatSim = class(TForm)
  etiq: TLabel;
  thlEdit: TEdit;
  tlhEdit: TEdit;
  Bevel1: TBevel;
  acepBtn: TBitBtn;
  cancBtn: TBitBtn;
  thlLabel: TLabel;
  tlhLabel: TLabel;
  Label1: TLabel;
  Edit1: TEdit;
  Button1: TButton;
  Bevel2: TBevel;
  Label2: TLabel;
  Edit2: TEdit;
  Button2: TButton;
  Bevel3: TBevel;
  Dialogo: TOpenDialog;
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);

 private
  { Private declarations }
 public
  { Public declarations }
```

```
 end;

var
 DatSim: TDatSim;

implementation

{$R *.DFM}

procedure TDatSim.Button1Click(Sender: TObject);
begin
 Dialogo.FilterIndex:=1;
 if Dialogo.Execute then
  Edit1.Text:=Dialogo.FileName;
end;

procedure TDatSim.Button2Click(Sender: TObject);
begin
 Dialogo.FilterIndex:=2;
 if Dialogo.Execute then
  Edit2.Text:=Dialogo.FileName;
end;


end.


unit sec4;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 StdCtrls, Buttons;
```

```
type
 TSelSalv = class(TForm)
  SiBtn: TBitBtn;
  NoBtn: TBitBtn;
  CancBtn: TBitBtn;
  Label1: TLabel;
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 SelSalv: TSelSalv;

implementation

{$R *.DFM}

end.



unit sim4;

interface
const
 LOW = 888;
 HIGH = 999;
type
 PunteroPuerta = ^Puerta;
 Puerta = record                {tipo definido para representar a una puerta del circuito}
        nodoe1,nodoe2,nodo_s,respcambio,pendiente:integer;
        beta,tp,fanout:real;
```

```
        ant,prox:PunteroPuerta;
        end;

procedure Buscar (var prim,nodo,ult,nodobusc:PunteroPuerta;valorbusc:integer);
function CopiaSubCad (var s:string) : string ;
procedure ObDatNet( s:string;var info:boolean;var NCorrecto,n_ent1,n_ent2,
        n_sal:integer;var bta:real);
procedure LeerEntradas(var f :TextFile; var lisN,flisN:PunteroPuerta;
        var code:integer);
function rcad(x:real):string;
procedure CalcularAtributos(var listNodos,finNodos:PunteroPuerta;
        const cadlh,cadhl,NombreTecnolog:string;ModeloAplicable:boolean);
procedure ListNetlist(var prim,ult,aux:PunteroPuerta;n_e1,n_e2,n_s,t0:integer;
        bet,tp1:real);
procedure BorrarLista(var plist,flist:PunteroPuerta);
procedure InicRetardos (var punt_list:PunteroPuerta);
procedure BorrarEntradas(var plista,flista:PunteroPuerta);
procedure CalcFanout(var lista:PunteroPuerta);
function valorfich(cad:string;var fich:TextFile):string;
function ConexCorrecta (var punteroI,punteroF:PunteroPuerta;var nodo_error:integer):
boolean;


implementation
uses
 SysUtils;

function CopiaSubCad (var s:string) : string ;      {devuelve la subcadena situada al}
var                                                 {comienzo de la cadena fuente}
 blanco : integer;
begin
 blanco := Pos (' ',s);
 if blanco=0 then
  Result := Copy(s,1,length(s))
```

```
 else begin
  Result := Copy(s,1,blanco-1);
  repeat begin
   delete (s,1,blanco);
   blanco := pos (' ',s);
  end;
  until (blanco=0) or (blanco>1);
 end;
end;

procedure ObDatNet( s:string;var info:boolean;var NCorrecto,n_ent1,n_ent2,
        n_sal:integer;var bta:real);
var                                  {extrae de una línea de fichero netlist los datos}
 Ent2NoCero:boolean;                 {que contiene y verifica que se ajustan al formato}
 codigo,TipPuerta :integer;
 m,d1,d2,d3,beta_char:string;
 function TPuerta (const m:string):integer;          {anidada a ObDatNet}
 begin                          {determina el tipo de puerta}
  Result:=0;
  if (m='inv') or (m='Inv') or (m='INV') then
   Result:=1
  else begin
   if (m='nor2') or (m='Nor2') or (m='NOR2') then
    Result:=2;
  end;
 end;
begin
 info:=false;
 if (s<>'') and (s[1]<>' ') and (s[1]<>'*') and (s[1]<>'C') and (s[1]<>'c') then
 begin
  info:=true;
  m:=CopiaSubCad(s);
  TipPuerta:=TPuerta(m);             {comprueba el tipo de puerta que define la línea}
  if TipPuerta<>0 then begin         {puerta soportada por SiDSen}
```

```
d1:=CopiaSubCad(s);                 {primer nodo de entrada}
val(d1,n_ent1,codigo);
if codigo=0 then begin
 if n_ent1<>0 then begin
  if TipPuerta=1 then begin         {puerta de tipo inversor}
   n_ent2:=0;
   codigo:=0;
   Ent2NoCero:=true;
  end else begin                    {puerta NOR2}
   d2:=CopiaSubCad(s);              {segundo nodo de entrada}
   val(d2,n_ent2,codigo);
   if codigo=0 then
    Ent2NoCero:=n_ent2<>0;
  end;
  if codigo=0 then begin
   if Ent2NoCero then begin
    d3:=CopiaSubCad(s);
    val(d3,n_sal,codigo);           {nodo de salida}
    if codigo=0 then begin
     if n_sal<>0 then begin
      beta_char:=CopiaSubCad(s);    {relación de aspecto}
      val(beta_char,bta,codigo);
      if codigo<>0 then
       NCorrecto:=2;
     end else
      NCorrecto:=3;
    end else
     NCorrecto:=1;
   end else
    NCorrecto:=3;
  end else
   NCorrecto:=1;
 end else
  NCorrecto:=3;
```

```
    end else
      NCorrecto:= 1;
   end else
    NCorrecto:=4;
 end;
end;

procedure Nuevo(var punt:PunteroPuerta;ent1,ent2,sal,nto:integer;nbeta,ntp:real);
begin                    {crea un nuevo elemento de tipo Puerta y asigna}
 new(punt);              {valores a sus campos}
 punt^.nodo_s:=sal;
 punt^.nodoe1:=ent1;
 punt^.beta:=nbeta;
 punt^.nodoe2:=ent2;
 punt^.fanout:=0;
 punt^.respcambio:=0;
 punt^.pendiente:=nto;
 punt^.tp:=ntp;
 punt^.ant:=nil;
 punt^.prox:=nil;
end;


procedure ListNetlist(var prim,ult,aux:PunteroPuerta;n_e1,n_e2,n_s,t0:integer;
        bet,tp1:real);
var                      {inserta en la lista enlazada un elemento que contiene}
 nuev:PunteroPuerta;     {los datos que recibe como parámetro ordenado en función}
                         {del valor ascendente del identificador del nodo de salida}
begin
 Nuevo(nuev,n_e1,n_e2,n_s,t0,bet,tp1);
 if prim=nil then begin  {inserta en lista vacía}
  prim:=nuev;
  ult:=nuev;
  aux:=nuev;
```

```
end else begin
 if n_s<prim^.nodo_s then begin   {inserta al comienzo de la lista}
  prim^.ant:=nuev;
  nuev^.prox:=prim;
  prim:=nuev;
  aux:=nuev;
 end else begin
  if n_s>ult^.nodo_s then begin    {inserta al final de la lista}
   ult^.prox:=nuev;
   nuev^.ant:=ult;
   ult:=nuev;
   aux:=nuev;
  end else begin              {inserta en el cuerpo de la lista, buscar el pto. de inserción }
   if ((aux^.nodo_s+ult^.nodo_s) div 2)< n_s then        {se determina el intervalo de}
    aux:=ult                                  {búsqueda fijando el elemento desde el que }
   else                                       { inicia en aquél que lleve a ejecutar }
    if ((aux^.nodo_s+prim^.nodo_s) div 2)> n_s then     { menos iteraciones}
    aux:=prim;
   if (aux=prim) or (n_s>aux^.nodo_s)then begin
    while aux^.nodo_s<n_s do
     aux:=aux^.prox;
   end else begin
    if (aux=ult) or (n_s<aux^.nodo_s)then begin
     while aux^.nodo_s>n_s do
      aux:=aux^.ant;
    end;
   end;
   if n_s>aux^.nodo_s then begin
    nuev^.prox:=aux^.prox;
    nuev^.ant:=aux;
    nuev^.prox^.ant:=nuev;
    aux^.prox:=nuev;
   end else begin
    nuev^.ant:=aux^.ant;
```

```
      nuev^.prox:=aux;
      nuev^.ant^.prox:=nuev;
      aux^.ant:=nuev;
     end;
     aux:=nuev;
    end;
   end;
  end;
end;

procedure LeerEntradas (var f :TextFile; var lisN,flisN:PunteroPuerta;
        var code : integer);
var                              {obtiene las entradas definidas en el fichero de entradas}
 to2,Ndo:integer;                {las inserta en la lista enlazada que representa al netlist}
 s1,retardo,nodo,pendte:string;  {y verifica el formato de las entradas}
 tp2:real;
 pt:PunteroPuerta;
begin
 code:=0;
 {$I-}
 Reset(f);
 {$I+}
 code:=IOResult;
 if code=0 then begin
  pt:=lisN;
  ReadLn(f,s1);
  while not Eof(f) and not (s1[1] in ['E','e']) and (code=0)do begin      {el proceso se}
   if s1[1]<>'*' then begin        { repite hasta alcanzar el final del fichero o encontrar }
    nodo:=CopiaSubCad(s1);         { un error de formato}
    val(nodo,Ndo,code);
    if code=0 then begin
     pendte:=CopiaSubCad(s1);
     if (pendte='h') or (pendte='H') then
      to2:=HIGH
```

```
      else begin
       if (pendte='l') or (pendte='L') then
        to2:=LOW
       else
        val(pendte,to2,code);
      end;
      if code=0 then begin
       tp2:=0;
       if (to2=1) or (to2=-1) then begin
        retardo:=CopiaSubCad(s1);
        val(retardo,tp2,code);
       end;
       if code=0 then
        ListNetlist(lisN,flisN,pt,0,0,Ndo,to2,0,tp2);        {la inserta en la lista enlazada}
      end;
     end;
    end;
    ReadLn(f,s1);
   end;
  end;
  CloseFile(f);
end;


procedure InicRetardos (var punt_list:PunteroPuerta);
var
 i:PunteroPuerta;      {inicializa diversos campos de los elementos de la lista para}
begin                  {comenzar una nueva simulación sobre el netlist}
 i:=punt_list;
 while i<>nil do begin
  i^.pendiente:=0;
  i^.respcambio:=0;
  i^.tp:=0;
  i:=i^.prox;
```

```
 end;
end;

procedure BorrarEntradas(var plista,flista:PunteroPuerta);        {borra los elementos de}
var                                {la lista que representan a entradas que se reconocen porque}
 borrar,aux:PunteroPuerta;         {los campos nodoe1=nodoe2=0}
begin
 borrar:=plista;
 while (borrar^.nodoe1=0) and (borrar^.nodoe2=0) do begin      {se borra un elemento}
  plista:=plista^.prox;                                        {que ocupa la cabeza dela lista}
  dispose(borrar);
  plista^.ant:=nil;
  borrar:=plista;
 end;
 borrar:=borrar^.prox;
 while borrar<>nil do begin
  if (borrar^.nodoe1=0) and (borrar^.nodoe2=0) then begin
   if borrar=flista then begin              {se borra el último elemento de la lista}
    flista:=flista^.ant;
    dispose(borrar);
    flista^.prox:=nil;
    borrar:=nil;
   end else begin
    borrar^.ant^.prox:=borrar^.prox;        {borrado de un elemento en el cuerpo de la }
    borrar^.prox^.ant:=borrar^.ant;         {lista}
    aux:=borrar^.prox;
    dispose(borrar);
    borrar:=aux;
   end;
  end else
   borrar:=borrar^.prox;
 end;
end;
```

```
procedure Buscar (var prim,nodo,ult,nodobusc:PunteroPuerta;valorbusc:integer);
var
 inicio,fin:PunteroPuerta;          {busca el elemento cuyo nodo de salida recibe}
 i:integer;                         {como parámetro}
 encontrado:boolean;
begin
 encontrado:=false;
 if valorbusc>nodo^.nodo_s then begin     {se compara el valor buscado para }
  inicio:=nodo;                           {determinar el intervalo de búsqueda}
  fin:=ult;
 end else begin
  inicio:=prim;
  fin:=nodo;
 end;
 i:=(inicio^.nodo_s+fin^.nodo_s)div 2;
 if valorbusc>i then begin
  nodobusc:=fin;              {busca hacia valores descendentes de nodo de salida}
  while not encontrado and (nodobusc^.nodo_s>=valorbusc) do begin
   if nodobusc^.nodo_s=valorbusc then
    encontrado:=true
   else
    nodobusc:=nodobusc^.ant;
  end;
 end else begin
  nodobusc:=inicio;           {busca hacia valores ascendentes}
  while not encontrado and (nodobusc^.nodo_s<=valorbusc) do begin
   if nodobusc^.nodo_s=valorbusc then
    encontrado:=true
   else
    nodobusc:=nodobusc^.prox;
  end;
 end;
 if not encontrado then
  nodobusc:=nil;
```

```
end;

procedure BorrarLista(var plist,flist:PunteroPuerta);        {elimina la lista enlazada}
var
 aux:PunteroPuerta;
begin
 aux:=plist;
 while plist<>nil do begin
  plist:=plist^.prox;
  dispose(aux);
  aux:=plist;
 end;
 flist:=nil;
end;

function ConexCorrecta (var punteroI,punteroF:PunteroPuerta;var nodo_error:integer):
        boolean;                {comprueba que se conecten correctamente los nodos del}
var                             {circuito y las entradas a aplicar sobre el mismo}
 conex:boolean;
 puntero1,puntero_conec:PunteroPuerta;
begin
 nodo_error:=0;                 {cada nodo de entrada especificado en un elemento de la }
 conex:=true;                   {lista se busca para comprobar su conexión}
 puntero1:=punteroI;
 while conex and (puntero1<>nil) do begin
  if puntero1^.nodoe1<>0 then begin
   Buscar(punteroI,puntero1,punteroF,puntero_conec,puntero1^.nodoe1);
   if puntero_conec=nil then
    conex:=false
   else begin
    if puntero1^.nodoe2<>0 then begin
     Buscar(punteroI,puntero1,punteroF,puntero_conec,puntero1^.nodoe2);
     if puntero_conec=nil then
      conex:=false
```

```
      else
       puntero1:=puntero1^.prox;
     end else
      puntero1:=puntero1^.prox;
    end;
   end else
    puntero1:=puntero1^.prox;
 end;
 if (not conex) and (puntero1<>nil) then
  nodo_error:=puntero1^.nodo_s;
 Result:=conex;
end;

function valorfich(cad:string;var fich:TextFile):string;
var                          {lee del fichero de la tecnología el valor del }
 linea,cadaux:string;        { parámetro de modelado especificado}
 obtenido:boolean;
 i:integer;
begin
 obtenido:=false;
 {$I-}
 reset(fich);
 {$I+}
 i:=IOResult;
 if i=0 then begin
  repeat
   readln(fich,linea);
   cadaux:=CopiaSubCad(linea);
  if cadaux=cad then
   obtenido:=true;
  until Eof(fich) or obtenido;
 end;
 CloseFile(fich);
 if obtenido then
```

```
   Result:=linea
  else
   Result:='';
end;

function valor_en_fich(cadena:string;var fichero:TextFile):real;
var                    {devuelve en formato real el valor del parámetro de modelado}
 i:integer;            {que se especifica obtenido del fichero de la tecnología}
 datoreal:real;
 datocad:string;
begin
 Result:=0;
 datocad:=valorfich(cadena,fichero);
 if datocad<>'' then begin
    val(datocad,datoreal,i);
  if i=0 then
    Result:=datoreal;
 end;
end;

function rcad(x:real):string;          {convierte a cadena de caracteres el real recibido}
var
 exp,i:string;
 pE,expo,punto,cod,cont:integer;
begin
 str(x,i);             {primero se obtiene en formato de mantisa y exponente, para}
 delete(i,1,1);        {transformarlo a notación de coma fija eliminando el exponente}
 pE:=pos('E',i);       {y trasladando el punto decimal en función su valor y signo}
 exp:=copy(i,pE+1,length(i)-pE);
 delete(i,pE,length(i)-pE+1);
 delete(i,length(i)-2,3);
 val(exp,expo,cod);
 if expo<>0 then begin
  punto:=pos('.',i);
```

```
  delete(i,punto,1);
  if expo>=1 then begin
   if (punto+expo)>length(i)then begin
    for pE:=1 to (punto+expo-length(i)-1) do
     i:=i+'0';
   end;
   insert('.',i,punto+expo)
  end else begin
   if expo<=(-2) then
    for cod:=-1 downto (expo+1) do
     i:='0'+i;
   i:='0.'+i;
  end;
 end;
 if pos('.',i)<>length(i) then begin
  cod:=length(i);
  if i[cod-1]='0' then
   delete(i,cod,1)
  else begin
   cont:=0;
   while i[cod]='9' do begin
    dec(cod);
    inc(cont);
   end;
   if (cont>=3) and (i[cod]<>'.') then begin
    delete(i,cod+1,length(i)-cod);
    exp:=copy(i,cod,1);
    delete(i,cod,1);
    val(exp,pE,cod);
    pE:=pE+1;
    i:=i+IntToStr(pE);
   end;
  end;
  while i[length(i)]='0' do
```

```
    delete(i,length(i),1);
 end;
 if i[length(i)]='.' then
  delete(i,length(i),1);
 if x<0 then
  i:='-'+i;
 Result:=i;
end;

procedure CalcFanout (var lista:PunteroPuerta);   {calcula el fan-out de cada puerta}
var
 fout,aux:PunteroPuerta;
begin
 fout:=lista;
 while fout<>nil do begin
  aux:=lista;
  while aux<>nil do begin
   if aux^.nodoe1=fout^.nodo_s then
    fout^.fanout:=fout^.fanout+aux^.beta;
   if aux^.nodoe2=fout^.nodo_s then
    fout^.fanout:=fout^.fanout+aux^.beta;
   aux:=aux^.prox;
  end;
  fout^.fanout:=fout^.fanout/fout^.beta;
  fout:=fout^.prox;
 end;
end;

procedure CalcularAtributos(var listNodos,finNodos:PunteroPuerta;
        const cadlh,cadhl,NombreTecnolog:string;ModeloAplicable:boolean);
var
 NodoDesconocido,NodoConocido:PunteroPuerta;
 vdd,tphl0,tplh0,ne,rse,ke,vte,ce,rsd,dd,vtd,kd,vsbd:real;
 ecthl,ectlh:string;
```

```
 ficherotecnolog:TextFile;

procedure ParametroModelo(var par_valor:real;par_cad:string;const FO,beta:real;
        var tecnologfich:TextFile);          {anidada a CalcularAtributos}
var
 neg:boolean;                          {obtiene el valor de un parámetro cuando se usan}
begin                                  {ecuaciones externas}
 neg:=false;
 if par_cad[1]='-' then begin
  delete(par_cad,1,1);
  neg:=true;
 end;
 if pos('FO',par_cad)<>0 then
  par_valor:=FO
 else begin
  if pos('beta',par_cad)<>0 then
   par_valor:=beta
  else
   par_valor:=valor_en_fich(par_cad,tecnologfich);
 end;
 if neg then
  par_valor:=(-1)*par_valor;
end;

function operar_mat(operador:char;operando1,operando2:string;fo,bt:real;
        var tecnfich:TextFile):real;                    {anidada a CalcularAtributos}
var
 oper1,oper2:real;
 function ValorOperando(opdo:string;foVO,btVO:real;var tecnfichVO:TextFile):real;
 var                                                    {anidada a operar_mat}
  cod,i:integer;              {devuelve el valor de un operando de una subexpresión}
  x:real;                                               {de una ecuación de teclado}
 begin
  x:=0;
```

```
  i:=1;
  if opdo[i]='-' then
   i:=2;
  if opdo[i] in ['A'..'Z','a'..'z'] then          {operando=parámetro de modelado}
   ParametroModelo(x,opdo,foVO,btVO,tecnfichVO)
  else                                            {operando=valor numérico}
   val(opdo,x,cod);
  Result:=x;
 end;
begin                                  {calcula una subexpresión de ecuación de teclado}
 oper1:=ValorOperando(operando1,fo,bt,tecnfich);
 oper2:=ValorOperando(operando2,fo,bt,tecnfich);
 case operador of
  '+':Result:=oper1+oper2;
  '-':Result:=oper1-oper2;
  '·':Result:=oper1*oper2;
  '/':Result:=oper1/oper2;
 end;
end;

function evaluar(ope1,ope2:string;var cadmemoFO,cadmemobta:string;
        Simpl:boolean;fan,rasp:real;var t_fich:TextFile;operad:char):string;
begin
 if Simpl then begin                                    {fase de simplificación}
  if (pos('FO',ope1)<>0) or (pos('FO',ope2)<>0) then begin   {dependiente del fan-out}
   cadmemoFO:='('+ope1+operad+ope2+')';                 {se almacena y sustituye}
   Result:='Idfo';
  end else begin
   if (pos('beta',ope1)<>0) or (pos('beta',ope2)<>0) then begin  {dependiente de relación}
     cadmemobta:='('+ope1+operad+ope2+')';          {de aspecto se almacena y sustituye}
     Result:='Idbta';
    end else begin
   if pos('Idbta',ope1)<>0 then begin         {dependiente de relación de aspecto}
      cadmemobta:='('+cadmemobta+operad+ope2+')';
```

```
        Result:='Idbta';
      end else begin
       if pos('Idbta',ope2)<>0 then begin          {dependiente de relación de aspecto}
        cadmemobta:='('+ope1+operad+cadmemobta+')';
        Result:='Idbta';
       end else begin
        if pos('Idfo',ope1)<>0 then begin          {dependiente del fan-out}
         cadmemoFO:='('+cadmemoFO+operad+ope2+')';
         Result:='Idfo';
        end else begin
         if pos('Idfo',ope2)<>0 then begin         {dependiente del fan-out}
          cadmemoFO:='('+ope1+operad+cadmemoFO+')';
          Result:='Idfo';
         end else
          Result:=rcad(operar_mat(operad,ope1,ope2,fan,rasp,t_fich)); {no dependiente}
       end;                                                          {se calcula}
      end;
     end;
    end;
   end;
  end else
   Result:=rcad(operar_mat(operad,ope1,ope2,fan,rasp,t_fich));    {fase de cómputo}
end;                                                             {se calcula la subexpresión}

function TratarEcuacion(const ecuacion:string; valorbta,valorFO :real;
        var tecnofich :TextFile;simplificar:boolean):string;
var
 op1,op2,op3,j,pos1,pos2:integer;                                {devuelve el resultado de una}
 opndo1,opndo2,cadena,subcad,cadbta,cadFO:string;                {ecuación de teclado}
 cadenaeval,subcadeval:boolean;
 opdor:char;
begin
 cadena:=ecuacion;
 cadbta:='';
```

```
cadFO:='';
cadenaeval:=false;
while not cadenaeval do begin                {ecuación no evaluada}
 pos1:=pos(')',cadena);
 if pos1<>0 then begin                       {incluye términos entre paréntesis}
  pos2:=pos1-1;
  while cadena[pos2]<>'(' do
   dec(pos2);
  subcad:=copy(cadena,pos2+1,pos1-pos2-1);   {copia subexpresión entre paréntesis}
  delete(cadena,pos2+1,pos1-pos2-1);
  subcadeval:=false
 end else begin                              {no hay términos entre paréntesis}
  j:=1;
  if cadena[j]='-' then
   inc(j);
  while not(cadena[j] in ['+','-','/','·']) and (j<length(cadena)) do      {busca operador}
   inc(j);
  if j<length(cadena) then begin              {hay operador}
   subcad:=cadena;
   subcadeval:=false;
  end else begin                              {no hay operador, expresión evaluada}
   cadenaeval:=true;
   subcadeval:=true;
  end;
 end;
 while not subcadeval do begin               {mientras subexpresión no evaluada}
  op1:=1;
  if subcad[op1]='-' then
   inc(op1);
  while not (subcad[op1] in ['+','-','/','·']) and (op1<length(subcad)) do
   inc(op1);                                  {busca operador1}
  if op1<length(subcad) then begin                          {hay 1° operador}
   op2:=op1+1;
   if subcad[op2]='-' then
```

```
 inc(op2);
while not (subcad[op2] in ['+','-','/','·']) and (op2<length(subcad)) do
 inc(op2);
if op2<length(subcad) then begin                          {hay 2° operador}
 if (subcad[op1] in ['+','-']) and (subcad[op2] in ['/','·']) then begin
          {2° operador mayor precedencia de evaluación}
  op3:=op2+1;
  if subcad[op3]='-' then
   inc(op3);
  while not (subcad[op3] in ['+','-','/','·']) and (op3<length(subcad)) do
   inc(op3);
  if op3<length(subcad) then begin     {hay 3 operador}
   opdor:=subcad[op2];
   opndo1:=copy(subcad,op1+1,op2-op1-1);          {obtiene operandos}
   opndo2:=copy(subcad,op2+1,op3-1-op2);
   delete(subcad,op1+1,op3-(op1+1));
            {calcula la subexpresión y la sustituye por su valor}
   insert(evaluar(opndo1,opndo2,cadFO,cadbta,simplificar,valorFO,valorbta,
          tecnofich,opdor),subcad,op1+1);
  end else begin                {no hay 3 operador}
   opdor:=subcad[op2];
   opndo1:=copy(subcad,op1+1,op2-op1-1);          {obtiene operandos}
   opndo2:=copy(subcad,op2+1,length(subcad)-op2);
   delete(subcad,op1+1,length(subcad)-op1);
            {calcula la subexpresión y la sustituye por su valor}
   subcad:=subcad+evaluar(opndo1,opndo2,cadFO,cadbta,simplificar,valorFO,
          valorbta,tecnofich,opdor);
  end;
end else begin          {igual orden de evaluacion}
 opndo1:=copy(subcad,1,op1-1);              {obtiene operandos}
 opndo2:=copy(subcad,op1+1,op2-op1-1);
 opdor:=subcad[op1];
 delete(subcad,1,op2-1);
            {calcula la subexpresión y la sustituye por su valor}
```

```
      subcad:=evaluar(opndo1,opndo2,cadFO,cadbta,simplificar,valorFO,valorbta,
              tecnofich,opdor)+subcad;
     end;
    end else begin          {no hay 2º operador}
     opndo1:=copy(subcad,1,op1-1);          {obtiene operandos}
     opndo2:=copy(subcad,op1+1,length(subcad)-op1);
              {calcula la subexpresión y la sustituye por su valor}
     subcad:=evaluar(opndo1,opndo2,cadFO,cadbta,simplificar,valorFO,valorbta,
          tecnofich,subcad[op1]);
    end;
   end else          {subcadena evaluada, no contiene operadores}
    subcadeval:=true;
  end;              {fin del bucle subcad}
  if pos1<>0 then begin
   delete(cadena,pos2,2);
   insert(subcad,cadena,pos2);
  end else
   cadena:=subcad;
end;
if simplificar then begin          {en la fase de simplificación se termina insertando}
 repeat                        {los términos dependientes del fan-out y la relación}
  j:=pos('Idbta',cadena);          {de aspecto}
  if j<>0 then begin
   delete(cadena,j,5);
   insert(cadbta,cadena,j);
  end;
  j:=pos('Idfo',cadena);
  if j<>0 then begin
   delete(cadena,j,4);
   insert(cadFO,cadena,j);
  end;
 until j=0;
end;
Result:=cadena;
```

```
end;


function evaluar_hl1(const beta:real;const fan_out:real;var pend:integer;
        ecuac:string;var fich_tecnolog:TextFile;const modelo:boolean):real;
var
 fanout,thl:real;
 i:integer;
begin
 fanout:=fan_out;
 if fanout=0 then
  fanout:=1;
 if modelo and (ecuac<>'') then
  val(TratarEcuacion(ecuac,beta,fanout,fich_tecnolog,false),thl,i)
 else begin
  if fanout>1 then
   thl:=(tphl0*(1.0+0.18*(rse-6.0)-0.9342*(rsd-5.0)-0.2*(vsbd-1.5)-0.2*
   (dd-1.9)-4.81*(vtd+0.1)-1.08*(ke-2.5)+2.3305*(ne-1.5)-1.42*(beta-2.0))*
   (2.56*vdd*vdd*vdd-12.94*vdd*vdd+21.89*vdd-11.35)*
   (-0.9097*kd*kd+4.2936*kd-3.5384)*(-46.3739*vte*vte+64.2627*vte-20.3512)*
   (-0.894*ce*ce+5.8456*ce-8.3542))+(((2.3207*fanout)+8.2016)/tphl0)
  else
   thl:=tphl0*(1.0+0.18*(rse-6.0)-0.9342*(rsd-5.0)-0.2*(vsbd-1.5)-0.2*
   (dd-1.9)-4.81*(vtd+0.1)-1.08*(ke-2.5)+2.3305*(ne-1.5)-1.42*(beta-2.0)+
   0.86*(fanout-1.0))*(2.56*vdd*vdd*vdd-12.94*vdd*vdd+21.89*vdd-11.35)*
   (-0.9097*kd*kd+4.2936*kd-3.5384)*(-46.3739*vte*vte+64.2627*vte-20.3512)*
   (-0.894*ce*ce+5.8456*ce-8.3542);
 end;
 pend:=-1;
 Result:=thl;
end;

function evaluar_lh1(const bta:real;const ft:real;var pte:integer;ec:string;
        var fich_tec:TextFile;const modelolh:boolean):real;
```

```
var
 fanout,tlh:real;
 i:integer;
begin
 fanout:=ft;
 if fanout=0 then
  fanout:=1;
 if modelolh and (ec<>'') then
  val(TratarEcuacion(ec,bta,fanout,fich_tec,false),tlh,i)
 else begin
  if fanout>1 then
   tlh:=(tplh0*(1.0-0.1557*(vdd-1.5)+0.1824*(ke-2.5)-0.2966*(ne-1.5)+
   0.555*(bta-2.0))*(1.6/kd)*(2.8/ce)*((1.2/(vte*vte))-(3.0/vte)+2.5))+
   (((14.88*fanout)+0.46)/tplh0)
  else
   tlh:=tplh0*(1.0-0.1557*(vdd-1.5)+0.1824*(ke-2.5)-0.2966*(ne-1.5)+
   0.555*(bta-2.0)+0.31*(fanout-1.0))*(1.6/kd)*(2.8/ce)*((1.2/(vte*vte))-
   (3.0/vte)+2.5);
 end;
 pte:=1;
 Result:=tlh;
end;

procedure inversor(var ndoent,ndosal:PunteroPuerta;const eclh,echl:string;
        var tecfich_i:TextFile;modelo_i:boolean);
begin
 if ndoent^.pendiente=LOW then
  ndosal^.pendiente:=HIGH
 else begin
  if ndoent^.pendiente=HIGH then
   ndosal^.pendiente:=LOW
  else begin
   if ndoent^.pendiente=-1 then
    ndosal^.tp:=evaluar_lh1(ndosal^.beta,ndosal^.fanout,ndosal^.pendiente,
```

```
        eclh,tecfich_i,modelo_i)+ndoent^.tp;
    if ndoent^.pendiente=1 then
     ndosal^.tp:= evaluar_hl1(ndosal^.beta,ndosal^.fanout,ndosal^.pendiente,
        echl,tecfich_i,modelo_i)+ndoent^.tp;
    ndosal^.respcambio:=ndoent^.nodo_s;
   end;
 end;
end;

function beta_equivalente_LH(const dif_ret1,beta_m1:real):real;
begin
 Result:=(0.0002*dif_ret1*dif_ret1-0.0089*dif_ret1+3.6030)*beta_m1/2;
end;

function beta_equivalente_HL(const dif_ret2,beta_m2:real):real;
begin
 Result:=(0.0008*dif_ret2*dif_ret2-0.0525*dif_ret2+2.9443)*beta_m2/2;
end;

procedure pos_col_sub (var nodo_1,nodo_2,ns:PunteroPuerta;
        const eclh1:string;var tecfichpcs:TextFile;modelopcs:boolean);
const
 phicrit=30;
var
 phi,beq:real;
begin
 phi:=abs(nodo_1^.tp-nodo_2^.tp);
 if nodo_1^.tp>nodo_2^.tp then begin
  if phi>phicrit then
   ns^.tp:=64.55+nodo_1^.tp
  else begin
   beq:=beta_equivalente_LH(phi,ns^.beta);
   ns^.tp:=(12.1041*beq*beq-39.7883*beq+55.542)+nodo_1^.tp;
  end;
```

```
  ns^.respcambio:=nodo_1^.nodo_s;
 end else begin
  if phi>phicrit then
   ns^.tp:=64.55+nodo_2^.tp
  else begin
   beq:=beta_equivalente_LH(phi,ns^.beta);
   ns^.tp:=(12.1041*beq*beq-39.7883*beq+55.542)+nodo_2^.tp;
  end;
  ns^.respcambio:=nodo_2^.nodo_s;
 end;
 ns^.pendiente:=1;
end;

procedure pos_col_baj(var ndo1,ndo2,ndos:PunteroPuerta;const echl2:string;
        var tecfichpcb:TextFile;modelopcb:boolean);
const
 phicrit=30;
var
 phi,b_eq:real;
begin
 phi:=abs(ndo1^.tp-ndo2^.tp);
 if ndo1^.tp>ndo2^.tp then begin
  if phi>phicrit then
   ndos^.tp:=2.338+ndo2^.tp
  else begin
   b_eq:=beta_equivalente_HL(phi,ndos^.beta);
   ndos^.tp:= (-1.9696*b_eq*b_eq-3.9622*b_eq+21.8703)+ndo2^.tp;
  end;
  ndos^.respcambio:=ndo2^.nodo_s;
 end else begin
  if phi>phicrit then
   ndos^.tp:=2.338+ndo1^.tp
  else begin
   b_eq:=beta_equivalente_HL(phi,ndos^.beta);
```

```
    ndos^.tp:= (-1.9696*b_eq*b_eq-3.9622*b_eq+21.8703)+ndo1^.tp;
   end;
   ndos^.respcambio:=ndo1^.nodo_s;
  end;
  ndos^.pendiente:=-1;
end;

procedure nor2(var listndos,finndos,nodo1,nods:PunteroPuerta;
        const eclhn2,echln2:string;var tecfichn2:TextFile;modelon2:boolean);
var
 nodo2:PunteroPuerta;
 nodo_search:integer;
begin
 if nodo1^.nodo_s=nods^.nodoe1 then
  nodo_search:=nods^.nodoe2
 else
  if nodo1^.nodo_s=nods^.nodoe2 then
   nodo_search:=nods^.nodoe1;
 Buscar(listndos,nods,finndos,nodo2,nodo_search);
 if nodo2<>nil then begin
  if nodo2^.pendiente<>0 then begin
   if (nodo1^.pendiente=LOW) and (nodo2^.pendiente=LOW) then
    nods^.pendiente:=HIGH
   else begin
    if (nodo1^.pendiente=HIGH) or (nodo2^.pendiente=HIGH) then
     nods^.pendiente:=LOW
    else begin
     if nodo1^.pendiente=LOW then begin
      if nodo2^.pendiente=-1 then
       nods^.tp:=evaluar_lh1(3.503,nods^.fanout,nods^.pendiente,eclhn2,tecfichn2,
            modelon2)+nodo2^.tp
      else
       nods^.tp:=evaluar_hl1(2.3,nods^.fanout,nods^.pendiente,echln2,tecfichn2,
            modelon2)+nodo2^.tp;
```

```
      nods^.respcambio:=nodo2^.nodo_s;
     end else begin
      if nodo2^.pendiente=LOW then begin
       if nodo1^.pendiente=-1 then
        nods^.tp:=evaluar_lh1(3.503,nods^.fanout,nods^.pendiente,eclhn2,tecfichn2,
              modelon2)+nodo1^.tp
       else
        nods^.tp:=evaluar_hl1(2.3,nods^.fanout,nods^.pendiente,echln2,tecfichn2,
              modelon2)+nodo1^.tp;
       nods^.respcambio:=nodo1^.nodo_s;
      end else begin
       if (nodo1^.pendiente=-1) and (nodo2^.pendiente=-1) then
        pos_col_sub(nodo1,nodo2,nods,eclhn2,tecfichn2,modelon2)
       else begin
        if (nodo1^.pendiente=1) and (nodo2^.pendiente=1) then
         pos_col_baj(nodo1,nodo2,nods,echln2,tecfichn2,modelon2);
       end;
      end;
     end;
    end;
    if nods^.pendiente=0 then
     nods^.pendiente:=LOW;
  end;
 end;
end;


function check_nodos(var nodos:PunteroPuerta):boolean;          {verifica si todas las}
var                                                             {puertas han sido evaluadas}
 flag:boolean;
 i:PunteroPuerta;
begin
 i:=nodos;
```

```
 flag:=false;
 while not flag and (i<>nil) do begin
  if i^.pendiente=0 then
   flag:=true;
  i:=i^.prox;
 end;
 Result:=flag;
end;

procedure tipo_puerta (var listtp,fintp,nconoc,ndesc:PunteroPuerta;
        const eclhtp,echltp:string;var tecfichtp:TextFile;modelotp:boolean);
begin
 if (ndesc^.nodoe1<>0) and (ndesc^.nodoe2=0) then
  inversor(nconoc,ndesc,eclhtp,echltp,tecfichtp,modelotp)
 else
  if (ndesc^.nodoe1<>0) and (ndesc^.nodoe2<>0) then
   nor2(listtp,fintp,nconoc,ndesc,eclhtp,echltp,tecfichtp,modelotp);
end;

procedure LeerParametrosModelo(s:string);
var
 fich:TextFile;
begin
 AssignFile(fich,s);
 vdd:=valor_en_fich('vdd',fich);
 tphl0:=valor_en_fich('tphl0',fich);
 tplh0:=valor_en_fich('tplh0',fich);
 ne:=valor_en_fich('ne',fich);
 rse:=valor_en_fich('rse',fich);
 ke:=valor_en_fich('ke',fich);
 vte:=valor_en_fich('vte',fich);
 ce:=valor_en_fich('ce',fich);
 rsd:=valor_en_fich('rsd',fich);
 dd:=valor_en_fich('dd',fich);
```

```
 vtd:=valor_en_fich('vtd',fich);
 kd:=valor_en_fich('kd',fich);
 vsbd:=valor_en_fich('vsbd',fich);
end;
                    {CalcularAtributos}
begin
 ecthl:='';
 ectlh:='';
 AssignFile(ficherotecnolog,NombreTecnolog);
 if ModeloAplicable then begin          {determina aplicación del modelo vs. Ec. }
  if cadhl<>'' then                     {de teclado, que se simplifican}
   ecthl:=TratarEcuacion(cadhl,0,0,ficherotecnolog,true);
  if cadlh<>'' then
   ectlh:=TratarEcuacion(cadlh,0,0,ficherotecnolog,true);
 end;
 if not ModeloAplicable or (cadhl='') or (cadlh='') then
  LeerParametrosModelo(NombreTecnolog);
 while check_nodos(listNodos) do begin          {circuito no evaluado}
  NodoConocido:=listNodos;
  while NodoConocido<>nil do begin              {busca elementos ya evaluados}
   if NodoConocido^.pendiente<>0 then begin
    NodoDesconocido:=listNodos;
    while NodoDesconocido<>nil do begin         {elementos no evaluados conectados a}
     if (NodoDesconocido^.nodoe1=NodoConocido^.nodo_s) or       {uno evaluado}
           (NodoDesconocido^.nodoe2=NodoConocido^.nodo_s) then
     begin
      if NodoDesconocido^.pendiente=0 then
       tipo_puerta(listNodos,finNodos,NodoConocido,NodoDesconocido,ectlh,ecthl,
           ficherotecnolog,ModeloAplicable);
     end;
     NodoDesconocido:=NodoDesconocido^.prox;
    end;
   end;
   NodoConocido:=NodoConocido^.prox;
```

```
  end;
 end;
end;

end.          {Fin de la unidad sim4}


unit res4;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 StdCtrls, ExtCtrls , sim4, TeeProcs, TeEngine, Chart, Series;

type
 pbuffer=^bfer;
 bfer=record
        nod:integer;
        retardo:real;
        enlace:pbuffer;
   end;
 TResultado = class(TForm)
  Chart1: TChart;
  Series1: TLineSeries;
  Panel1: TPanel;
  Seleccion: TRadioGroup;
  MosBtn: TButton;
  ListaNodos: TListBox;
  NodEtiq: TLabel;
  EfG2: TBevel;
  CerBtn: TButton;
  CaminoM: TMemo;
  Label1: TLabel;
```

```
  Etiqueta: TLabel;
  Bevel1: TBevel;
  OrigenTemp: TCheckBox;
  procedure SeleccionClick(Sender: TObject);
  procedure CerBtnClick(Sender: TObject);
  procedure MosBtnClick(Sender: TObject);
  procedure FormCreate(Sender: TObject);

  private
  buffer : pbuffer;
  NodoEscogido:PunteroPuerta;
  { Private declarations }
  public
  FinlistaN,PrinclistaN:PunteroPuerta;
  procedure Camino;
  procedure EncontrarNodoCritico;
  function RealACad(valor_real:real) : string;
  procedure Inbuffer(const x:integer;const y:real);
  procedure Borrarbuffer;
  procedure Mostrar (var cadena : string);
  procedure inicchart;
  procedure construirserie;
  procedure MostrarHint(var HintStr:string;var CanShow:Boolean;var
  HintInfo:THintInfo);
        { Public declarations }
 end;

var
 Resultado: TResultado;

implementation

{$R *.DFM}
```

```
procedure TResultado.SeleccionClick(Sender: TObject);
begin
 if Seleccion.ItemIndex =0 then
  ListaNodos.Enabled :=false
 else begin
  ListaNodos.Enabled :=true;
  ListaNodos.ItemIndex :=0;
 end;
end;

procedure TResultado.CerBtnClick(Sender: TObject);
begin
 Close;
end;

procedure TResultado.Inbuffer(const x:integer;const y:real);      {inserta un nuevo }
var                                  { elemento en la lista enlazada}
 q:pbuffer;
begin
 new(q);
 q^.nod:=x;
 q^.retardo:=y;
 if buffer=nil then
  q^.enlace:=nil
 else
  q^.enlace:=buffer;
 buffer:=q;
end;

procedure TResultado.Borrarbuffer;          {borra la lista enlazada}
var
 p:pbuffer;
begin
 while buffer<>nil do begin
```

```
  p:=buffer;
  buffer:=buffer^.enlace;
  dispose(p);
 end;
end;

procedure TResultado.Camino;          {construye una lista enlazada con los nodos}
var                                   {que forman el camino que recorre la señal}
 Naux,rcambio:PunteroPuerta;
begin
 Naux:=NodoEscogido;
 rcambio:=nil;
 Inbuffer(Naux^.nodo_s,Naux^.tp);
 while (Naux<>nil) and (Naux^.respcambio<>0) do begin
  Buscar(PrinclistaN,Naux,FinlistaN,rcambio,Naux^.respcambio);
  Naux:=rcambio;
  Inbuffer(Naux^.nodo_s,Naux^.tp);
 end;
end;

procedure TResultado.EncontrarNodoCritico;    {busca el nodo que presenta mayor}
var                                           {retardo en el circuito simulado}
 p:PunteroPuerta;
begin
 NodoEscogido:=PrinclistaN;
 p:=PrinclistaN;
 while p<>nil do begin
  if p^.tp > NodoEscogido^.tp then
   NodoEscogido:=p;
  p:=p^.prox;
 end;
 if NodoEscogido^.tp=0 then
  NodoEscogido:=nil;
end;
```

```
procedure TResultado.Mostrar(var cadena:string);
begin
 CaminoM.Lines.Add(cadena);
 cadena:='';
end;

procedure TResultado.inicchart;          {inicializa el componente de gráficos de datos}
begin
 if Chart1.Title.Text.Count<>0 then
  Chart1.Title.Text.Delete(0);
 if Series1.Count<>0 then
  Series1.Clear;
end;

procedure TResultado.construirserie;     {inserta los puntos de la serie a representar}
var                  {en el componente de gráficos que corresponden al camino de la señal}
 paux:pbuffer;
begin
 Chart1.Title.Text.Add('Camino correspondiente al nodo '+
                IntToStr(NodoEscogido^.nodo_s));
 paux:=buffer;
 while paux<>nil do begin
  if OrigenTemp.Checked then
   Series1.AddXY(paux^.nod,paux^.retardo-buffer^.retardo,IntToStr(paux^.nod),
       Series1.SeriesColor)
  else
   Series1.AddXY(paux^.nod,paux^.retardo,IntToStr(paux^.nod),Series1.SeriesColor);
  paux:=paux^.enlace;
 end;
end;

procedure TResultado.MosBtnClick(Sender: TObject);
var
 ndoselec,code:integer;
```

```
 retardo,Nresp,caden :string;
 j:PunteroPuerta;
 auxbuf:pbuffer;
begin
 buffer:=nil;
 NodoEscogido:=nil;
 CaminoM.Text :='';
 inicchart;
 if Seleccion.ItemIndex =0 then
  EncontrarNodoCritico
 else begin
  val(ListaNodos.Items[ListaNodos.ItemIndex],ndoselec,code);
  j:=PrinclistaN;
  Buscar(PrinclistaN,j,FinlistaN,NodoEscogido,ndoselec);
 end;
 if NodoEscogido=nil then
  Etiqueta.Caption:= 'El circuito no presenta transiciones en sus nodos'
 else begin
  if NodoEscogido^.tp=0 then begin
   if NodoEscogido^.pendiente=LOW then
    Etiqueta.Caption:='El nodo '+ ListaNodos.Items[ListaNodos.ItemIndex]+
       ' permanece a nivel bajo'
   else begin
    if NodoEscogido^.pendiente=HIGH then
     Etiqueta.Caption:='El nodo '+ ListaNodos.Items[ListaNodos.ItemIndex]+
          ' permanece a nivel alto';
   end;
  end else begin
   Camino;
   if OrigenTemp.Checked then
    retardo:=RealACad(NodoEscogido^.tp-buffer^.retardo)
   else
    retardo:=RealACad(NodoEscogido^.tp);
   Etiqueta.Caption:='El retardo que presenta el nodo '+
```

```
        IntToStr(NodoEscogido^.nodo_s)+' es:'+#13+retardo+' ps.';
    caden:='';
    auxbuf:=buffer;
    while auxbuf<>nil do begin
     Nresp:=IntToStr(auxbuf^.nod);
     if (length(caden)+length(Nresp))>80 then
      Mostrar(caden);
     caden:=caden+Nresp;
     if auxbuf^.enlace<>nil then begin
      if length(caden)>77 then
       Mostrar(caden);
      caden:=caden+'-->';
     end;
     auxbuf:=auxbuf^.enlace;
    end;
    CaminoM.Lines.Add(caden);
    CaminoM.SelStart :=0;
    CaminoM.SelLength :=1;
    construirserie;
    Borrarbuffer;
   end;
  end;
end;

function TResultado.RealACad(valor_real:real) : string;
var
 cad:string;
 pospto:integer;
begin
 cad:=rcad(valor_real);
 pospto:=pos('.',cad);
 if (length(cad)-pospto)>2 then
  delete(cad,pospto+3,length(cad)-pospto-2);
 Result:=cad;
```

```
end;
procedure TResultado.MostrarHint(var HintStr:string;var CanShow:Boolean;
        var HintInfo:THintInfo);          {muestra el valor del punto de la serie sobre el}
var                                        {que el usuario sitúa el cursor del ratón}
 indice:LongInt;
 texto:string;
 posc:TPoint;
begin
 if HintInfo.HintControl=Chart1 then begin
  CanShow:=false;
  indice:=Series1.GetCursorValueIndex;
  if indice>-1 then begin
   posc:=Chart1.GetCursorPos;
   texto:='Nodo: '+Series1.XValueToText(Series1.XValues[indice]);
   texto:=texto+' Retardo: '+Series1.YValueToText(Series1.YValues[indice])+' ps';
   HintStr:=texto;
   HintInfo.HintPos.x:=posc.x+Chart1.ClientOrigin.x;
   HintInfo.HintPos.y:=posc.y+Chart1.ClientOrigin.y;
   CanShow:=true;
  end;
 end;
end;

procedure TResultado.FormCreate(Sender: TObject);
begin
 Application.OnShowHint:=MostrarHint;
end;

end.


unit princ5;

interface
```

```
uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 StdCtrls, ComCtrls, Menus, ExtCtrls, Buttons,sim4;

type
 PInfoS = ^InfoS;
 InfoS = record
     nombre,bet: string;
     next: PInfoS;
  end;
 TFormPFC = class(TForm)
  MenuPrinc1: TMainMenu;
  BarrHerr1: TPanel;
  Arch1: TMenuItem;
  Nuevo1: TMenuItem;
  Abr1: TMenuItem;
  Guard1: TMenuItem;
  GComo1: TMenuItem;
  sep1: TMenuItem;
  Sal1: TMenuItem;
  BarrEST1: TStatusBar;
  Editor: TRichEdit;
  Edit1: TMenuItem;
  Cop1: TMenuItem;
  Cort1: TMenuItem;
  Peg1: TMenuItem;
  Bor1: TMenuItem;
  n1: TMenuItem;
  Sel1: TMenuItem;
  Ver1: TMenuItem;
  BH1: TMenuItem;
  BE1: TMenuItem;
  CopiarBtn: TSpeedButton;
```

```
CortarBtn: TSpeedButton;
PegarBtn: TSpeedButton;
BorrarBtn: TSpeedButton;
NuevoBtn: TSpeedButton;
AbrirBtn: TSpeedButton;
GuardarBtn: TSpeedButton;
OpenDialog1: TOpenDialog;
SaveDialog1: TSaveDialog;
PopupM1: TPopupMenu;
CortPup1: TMenuItem;
CopPup1: TMenuItem;
PegPup1: TMenuItem;
BorPup1: TMenuItem;
A1: TMenuItem;
SelPup1: TMenuItem;
Simular1: TMenuItem;
SimRet1: TMenuItem;
ReSim1: TMenuItem;
Trad1: TMenuItem;
NuevNet: TMenuItem;
NuevEnt: TMenuItem;
NuevEntBtn: TSpeedButton;
Ayuda1: TMenuItem;
Ayuda2: TMenuItem;
Convertir: TMenuItem;
procedure BH1Click(Sender: TObject);
procedure BE1Click(Sender: TObject);
procedure Edit1Click(Sender: TObject);
procedure Cop1Click(Sender: TObject);
procedure Cort1Click(Sender: TObject);
procedure Bor1Click(Sender: TObject);
procedure Peg1Click(Sender: TObject);
procedure Sel1Click(Sender: TObject);
procedure PopupM1Popup(Sender: TObject);
```

```
  procedure EditorSelectionChange(Sender: TObject);
  procedure Sal1Click(Sender: TObject);
  procedure Guard1Click(Sender: TObject);
  procedure GComo1Click(Sender: TObject);
  procedure NuevNetClick(Sender: TObject);
  procedure Abr1Click(Sender: TObject);
  procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  procedure FormCreate(Sender: TObject);
  procedure SimRet1Click(Sender: TObject);
  procedure ReSim1Click(Sender: TObject);
  procedure Trad1Click(Sender: TObject);
  procedure NuevEntClick(Sender: TObject);
  procedure Ayuda2Click(Sender: TObject);
  procedure ConvertirClick(Sender: TObject);


private
  NombreFich,FichEntradas : string;
  valido : boolean;
  PListaNodos,FListaNodos:PunteroPuerta;
        { Private declarations }
public
  function SalvarCambios : boolean ;
  function Salvar : boolean ;
  function SalvarComo : boolean;
  procedure PrepararLista;
  procedure Liberar;
  procedure ShowHint(Sender:TObject);
  function ParSPICE(const num:string;desplazamiento:integer):string;
  procedure InModelo(var archspi1,archmodel1:TextFile);
  procedure Entradas(const input1: string;const InfoA: boolean);
  procedure inicficheros (var f_ent,f_dat,f_net:TextFile);
  procedure InsertarBetayNombre (const betaI: real; const nombreI: string;
       var puntero: PInfoS);
```

```
  procedure Eliminar (var lisElim: PInfoS);
        { Public declarations }
 end;

var
 FormPFC: TFormPFC;

implementation

{$R *.DFM}
uses
 Clipbrd , sec4, res4,analisis_unit,ecuacion_unit;

procedure TFormPFC.BH1Click(Sender: TObject);
begin
 BarrHerr1.visible := not BarrHerr1.visible;
 if BarrHerr1.visible then
  BH1.Caption := 'Ocultar barra de &herramientas'
 else
  BH1.Caption := 'Mostrar barra de &herramientas';
end;

procedure TFormPFC.BE1Click(Sender: TObject);
begin
 BarrEST1.Visible := not BarrEST1.Visible;
 if BarrEST1.Visible then
  BE1.Caption := 'Ocultar barra de &estado'
 else
  BE1.Caption := 'Mostrar barra de &estado';
end;

procedure TFormPFC.Edit1Click(Sender: TObject);
var
 textosel : Boolean;
```

```
begin
  textosel := Editor.SelLength > 0;
  Cop1.Enabled := textosel;
  Cort1.Enabled := textosel;
  Bor1.Enabled := textosel;
  Peg1.Enabled := Clipboard.HasFormat (CF_TEXT);
end;

procedure TFormPFC.Cop1Click(Sender: TObject);
begin
  Editor.CopyToClipboard;
  PegarBtn.Enabled := true;
end;

procedure TFormPFC.Cort1Click(Sender: TObject);
begin
  Editor.CutToClipboard;
  PegarBtn.Enabled := true;
end;

procedure TFormPFC.Bor1Click(Sender: TObject);
begin
  Editor.ClearSelection;
end;

procedure TFormPFC.Peg1Click(Sender: TObject);
begin
  Editor.PasteFromClipboard;
end;


procedure TFormPFC.Sel1Click(Sender: TObject);
begin
  Editor.SelectAll;
```

```
end;

procedure TFormPFC.PopupM1Popup(Sender: TObject);
var
 tsel : Boolean;
begin
 tsel := Editor.SelLength > 0 ;
 CortPup1.Enabled := tsel;
 CopPup1.Enabled := tsel;
 BorPup1.Enabled := tsel;
 PegPup1.Enabled := Clipboard.HasFormat(CF_TEXT);
end;

procedure TFormPFC.EditorSelectionChange(Sender: TObject);
var
 textosl : Boolean;
begin
 textosl := Editor.SelLength > 0 ;
 CopiarBtn.Enabled := textosl;
 CortarBtn.Enabled := textosl;
 BorrarBtn.Enabled := textosl;
end;

procedure TFormPFC.Liberar;
begin
 if PListaNodos<>nil then
   BorrarLista(PListaNodos,FListaNodos);
end;

procedure TFormPFC.Sal1Click(Sender: TObject);
begin
 Close;
end;
```

```
procedure TFormPFC.Guard1Click(Sender: TObject);
begin
 if Editor.Modified then begin
  Salvar;
  if pos('.net',NombreFich)<>0 then begin
   valido:=false;
   Liberar;
   ReSim1.Enabled :=false;
  end;
 end;
end;

procedure TFormPFC.GComo1Click(Sender: TObject);
begin
 SalvarComo;
 if pos('.net',NombreFich)<>0 then begin
  valido:=false;
  Liberar;
  ReSim1.Enabled :=false;
 end;
end;

procedure TFormPFC.NuevNetClick(Sender: TObject);
begin
 if not Editor.Modified or SalvarCambios then begin
  if pos('.net',NombreFich)<>0 then begin
   valido:=false;
   ReSim1.Enabled :=false;
   Liberar;
  end;
  Editor.Text := '';
  Editor.Modified := False;
  NombreFich:= '';
  Caption := 'SiDSen-[ Sin título ]';
```

```
 end;
end;

procedure TFormPFC.Abr1Click(Sender: TObject);
begin
 if not Editor.Modified or SalvarCambios then
  if OpenDialog1.Execute then begin
   if pos('.net',NombreFich)<>0 then begin
    valido :=false;
    Liberar;
    ReSim1.Enabled :=false;
   end;
   NombreFich := OpenDialog1.FileName;
   Editor.Lines.LoadFromFile ( NombreFich);
   Editor.Modified := false;
   Caption := 'SiDSen-'+ NombreFich;
  end;
end;

procedure TFormPFC.ShowHint(Sender:TObject);
begin
 BarrEST1.SimpleText:=Application.Hint;
end;

function TFormPFC.Salvar : boolean;
begin
 if NombreFich = '' then
  Result := SalvarComo
 else begin
  Editor.Lines.SaveToFile ( NombreFich);
  Editor.Modified := false;
  Result := true;
 end;
end;
```

```
function TFormPFC.SalvarComo : boolean ;
begin
 SaveDialog1.FileName := NombreFich;
 if SaveDialog1.Execute then begin
  NombreFich := SaveDialog1.FileName ;
  Salvar;
  Caption := 'SiDSen-' + NombreFich;
  Result := true;
 end else
  Result := false;
end;

function TFormPFC.SalvarCambios : boolean ;
begin
 SelSalv.Label1.Caption := 'Ha modificado el documento' + NombreFich + #13 +
       #13 + '¿ Desea guardar los cambios ?';
 case SelSalv.ShowModal of
  mrYes :
   Result := Salvar;
  mrNo :
   Result := true;
  mrCancel :
   Result := false;
 end;
end;

procedure TFormPFC.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
 CanClose := not Editor.Modified or SalvarCambios ;
 if CanClose then
  Liberar;
end;

procedure TFormPFC.FormCreate(Sender: TObject);
```

```
begin
 NombreFich :='';
 FichEntradas :='';
 valido :=false;
 PListaNodos:=nil;
 FListaNodos:=nil;
 Application.OnHint:=ShowHint;
end;

procedure TFormPFC.SimRet1Click(Sender: TObject);
var
 Nentrada1,Nentrada2,Nsalida,nn,NetlistCorrecto,ErrorFichEnt,nodoE:integer;
 ModelUsuario,informacion:boolean;
 ArchTecnolog,thlcad,tlhcad,Mensaje:string;
 fichent,artec : TextFile;
 Beta1:real;
 punt:PunteroPuerta;
begin
 punt:=nil;
 if Editor.Text <> '' then begin
  if Editor.Modified then begin      {si se ha modificado el netlist se elimina la lista }
   valido:=false;                    {que lo representa}
   Liberar;
  end;
  if not valido then begin           {se lee el fichero netlist, creando la lista enlazada}
   ReSim1.Enabled:=false;            {y comprobando el formato de los datos}
   nn:=0;
   NetlistCorrecto:=0;
   while (nn < Editor.Lines.Count) and (NetlistCorrecto = 0) do begin
    ObDatNet(Editor.Lines[nn],informacion,NetlistCorrecto,Nentrada1,Nentrada2,
             Nsalida,Beta1);
    inc(nn);
    if informacion and (NetlistCorrecto=0) then
     ListNetlist(PListaNodos,FListaNodos,punt,Nentrada1,Nentrada2,Nsalida,0,
```

```
            Beta1,0);
  end;
  if NetlistCorrecto<>0 then begin
   Liberar;
   case NetlistCorrecto of
    1:Mensaje:='Identificador de nodo incorrecto( no numérico).';
    2:Mensaje:='Valor de relación de aspecto incorrecto( no numérico).';
    3:Mensaje:='Puerta conectada al nodo 0.';
    4:Mensaje:='Puerta no presente en la librería de puertas.';
   end;
   MessageDlg('Error en línea '+IntToStr(nn)+#13+Mensaje, mtError,[mbOK],0)
  end else begin
   valido:=true;
   CalcFanout(PListaNodos);
  end;
 end;
 if valido then begin
  ModelUsuario:=false;
  thlcad:='';
  tlhcad:='';
  if DatSim.ShowModal=mrOK then begin        {el usuario selecciona ficheros de la}
   thlcad:=DatSim.thlEdit.Text;              {tecnología y entradas, e introduce ec.}
   tlhcad:=DatSim.tlhEdit.Text;              {externas de forma opcional}
   if (thlcad<>'') or (tlhcad<>'')then
    ModelUsuario:=true;
   if (DatSim.Edit1.Text<>'') and (DatSim.Edit2.Text<>'') then begin
    if PListaNodos^.pendiente<>0 then begin          {se borran resultados y entradas}
     BorrarEntradas(PListaNodos,FListaNodos);         {de la simulación previa}
     InicRetardos(PListaNodos);
    end;
    ReSim1.Enabled:=false;
    FichEntradas:=DatSim.Edit1.Text;
    AssignFile(fichent,FichEntradas);
    {$I-}
```

```
Reset(fichent);
{$I+}
if IOResult=0 then begin
 CloseFile(fichent);
 LeerEntradas(fichent,PListaNodos,FListaNodos,ErrorFichEnt);
 if ErrorFichEnt = 0 then begin
  if ConexCorrecta(PlistaNodos,FlistaNodos,nodoE) then begin      {conexiones}
   ArchTecnolog:=DatSim.Edit2.Text;                               {correctas}
   AssignFile(arctec,ArchTecnolog);
   {$I-}
   Reset(arctec);
   {$I+}
   if IOResult=0 then begin                                {simulación}
    CloseFile(arctec);
    CalcularAtributos(PListaNodos,FListaNodos,tlhcad,thlcad,ArchTecnolog,
        ModelUsuario);
    MessageBox(0,'La simulación ha finalizado.','SiDSen',
        MB_OK+MB_ICONEXCLAMATION);
    ReSim1.Enabled :=true
   end else begin
    CloseFile(arctec);
    BorrarEntradas(PlistaNodos,FlistaNodos);
    MessageDlg(' Imposible realizar la simulación. El fichero '+ArchTecnolog+
      ' no se encuentra.',mtError,[mbOK],0);
   end;
  end else begin
   valido:=false;
   Liberar;
   Mensaje:= ' Imposible realizar la simulación. La puerta cuyo nodo de salida es '+
   IntToStr(nodoE)+ ' presenta un error en su conexión, o no recibe una entrada de '
        +FichEntradas;
   MessageDlg(Mensaje,mtError,[mbOK],0);
  end;
 end else begin
```

```
        BorrarEntradas(PListaNodos,FListaNodos);
        MessageDlg('Error en fichero de entradas.Imposible realizar la simulación.',
              mtError,[mbOK],0);
      end;
     end else begin
      CloseFile(fichent);
      MessageDlg('Imposible realizar la simulación. El fichero '+fichent+' no se '+
        'encuentra.',mtError,[mbOK],0);
     end;
    end else begin
     if DatSim.Edit1.Text='' then
      MessageDlg('Imposible realizar la simulación. Debe especificar un fichero de '+
       'entradas.',mtError,[mbOK],0);
     if DatSim.Edit2.Text='' then
      MessageDlg('Imposible realizar la simulación. Debe especificar un fichero de '+
       'la tecnología con los parámetros de modelado.',mtError,[mbOK],0);
    end;
   end;
  end;
 end else
  MessageDlg('No hay Netlist abierto.',mtError,[mbOK],0);
end;

procedure TFormPFC.ReSim1Click(Sender: TObject);          {inicializa y muestra el}
begin                                                     {cuadro de diálogo para consultar los}
 Resultado.PrinclistaN:=PListaNodos;                      {resultados de la simulación}
 Resultado.FinlistaN:=FListaNodos;
 PrepararLista;
 Resultado.Etiqueta.Caption:='';
 Resultado.CaminoM.Text:='';
 Resultado.Seleccion.ItemIndex:=0;
 Resultado.inicchart;
 Resultado.ShowModal;
end;
```

```
procedure TFormPFC.PrepararLista;
var
 aux:PunteroPuerta;
begin
 while Resultado.ListaNodos.Items.Count <> 0 do
  Resultado.ListaNodos.Items.Delete(0);
 aux:=PListaNodos;
 while aux<>nil do begin
  if aux^.nodoe1<>0 then
   Resultado.ListaNodos.Items.Add(IntToStr(aux^.nodo_s));
  aux:=aux^.prox;
 end;
end;

function TFormPFC.ParSPICE(const num:string;desplazamiento:integer):string;
var                          {multiplica por potencias de 10 para cambiar unidad de medida}
 dif,longitud,posicion,i:integer;
 numero:string;
begin
 Result:='';
 if num<>'' then begin
  numero:=num;
  posicion:=pos('.',numero);
  if posicion=0 then begin          {número entero}
   if desplazamiento > 0 then begin       {exponente positivo, añadir ceros derecha}
    for i:=1 to desplazamiento do
     numero:=numero+'0';
   end else begin                   {exponente negativo}
    longitud:=length(numero);
    dif:=longitud+desplazamiento;
    if dif>0 then begin             {insertar pto. decimal}
     insert('.',numero,dif+1);
     while numero[length(numero)]='0' do
      delete(numero,length(numero),1);
```

```
    if numero[length(numero)]='.' then
      delete(numero,length(numero),1);
    end else begin
     if dif<0 then begin              {insertar ceros izquierda y luego pto. decimal}
      for i:=-1 downto dif do
       numero:='0'+numero;
     end;
     numero:='0.'+numero;
    end;
   end;
  end else begin                      {número real}
   if desplazamiento>0 then begin          {exponente positivo}
    longitud:=length(numero);
    dif:=longitud-posicion;
    dif:=desplazamiento-dif;
    delete(numero,posicion,1);
    if dif>0 then begin
     for i:=1 to dif do          {se añaden ceros a la derecha}
      numero:=numero+'0';
    end else begin
     if dif<0 then               {se desplaza el pto. decimal a la derecha}
      insert('.',numero,desplazamiento+posicion);
    end;
    while numero[1]='0' do
     delete (numero,1,1);
   end else begin                      {exponente negativo}
    dif:=posicion+desplazamiento;
    delete(numero,posicion,1);
    if dif>1 then                      {desplazar pto. decimal a la izquierda}
     insert('.',numero,dif)
    else begin
     if dif<1 then begin               {insertar ceros izquierda y luego pto. decimal}
      for i:=0 downto dif do
       numero:='0'+numero;
```

```
      end;
      numero:='0.'+numero;
     end;
    end;
   end;
   Result:=numero;
 end;
end;


procedure TFormPFC.InModelo(var archspi1,archmodel1:TextFile);
begin                          {inserta en el fichero SPICE los parámetros de los transistores}
 Append(archspi1);
 writeln(archspi1,'***************');
 writeln(archspi1,'.MODEL EFET03 NMF VC='+valorfich('vte',archmodel1)+
 ' VSB=5'+' LAMBDA=0.05 RD=600 RS='+ParSPICE(valorfich('rse',archmodel1),2)+
                ' RG=0.3');
 writeln(archspi1,'+MFACT=1 GAMMA=4 IS=1e-13 DXL=10 VS=0.4 CDVC='+
       ParSPICE(valorfich('ke',archmodel1),-4)+' CDVSB=0');
 writeln(archspi1,'+DELTA=0 NP='+valorfich('ne',archmodel1)+' BETA='+
 valorfich('ce',archmodel1)+' ALFA=5 FC=1 VST=1 CGS0=2e-16 CGS1=6e-16');
 writeln(archspi1,'+VAT=-0.4 VBT=0.15 CDS=2e-16 RGS=800 RGD=800'+
                ' TD=2e-12');
 writeln(archspi1,'***************');
 writeln(archspi1,'.MODEL DFET03 NMF VC='+valorfich('vtd',archmodel1)+
       ' VSB='+valorfich('vsbd',archmodel1)+' LAMBDA=0.15 RD=500 RS='+
                ParSPICE(valorfich('rsd',archmodel1),2));
 writeln(archspi1,'+RG=0.3 MFACT=2 GAMMA=2 IS=1e-14 DXL=3 VS=0.4
       CDVC='+ParSPICE(valorfich('kd',archmodel1),-4));
 writeln(archspi1,'+CDVSB=0.00012 DELTA='+valorfich('dd',archmodel1)+
       ' NP=1.35 BETA=2.35 ALFA=3.5 FC=0.7 VST=1');
 writeln(archspi1,'+CGS0=1.8e-16 CGS1=4E-16 VAT=-1.1 VBT=-0.4 CDS=2e-16'+
       ' RGS=900');
 writeln(archspi1,'+RGD=900 TD=2e-12');
```

```
 writeln(archspi1,'**************');
 writeln(archspi1,' ');
 CloseFile(archspi1);
end;

procedure TFormPFC.Entradas(const input1:string; const InfoA: boolean);
var                                   {inserta en el fichero SPICE las entradas a simular}
 cadena,nodo,pendiente,retardo,tiempof:string;
 i,contador,linea,h:integer;
 tr:real;
 archent:TextFile;
begin
 AssignFile(archent,input1);
 {$I-}
 reset(archent);
 {$I+}
 i:=IOResult;
 if i=0 then begin
  linea:=Editor.Lines.indexOf('* ANALISIS');
  if linea=-1 then
   linea:=Editor.Lines.Count-2;
  if (TAS.Edittran.Text<>'') and InfoA then begin
   tiempof:=TAS.Edittran.Text;          {obtiene el tiempo definido para el análisis}
   i:=pos('s',tiempof);        {transitorio que se usa como última referencia de tiempo de}
   if i<>0 then begin                   {las entradas Pwl}
    h:=length(tiempof);
    while (i<>h) and not (tiempof[i] in ['0', '1'.. '9'] do
     inc(i);
    if i<>h then
     delete(tiempof,1,i-1)
    else
     tiempof:= '';
   end else
    tiempof:= '';
```

```
end else
 tiempof:= '';
Editor.Lines.Insert(linea, '*ENTRADAS');
Editor.Lines.Insert(linea, ' ');
Inc(linea);
readln(archent,cadena);
contador:=0;
while not Eof(archent) and not (cadena[1] in ['e','E']) do begin          {lee cada línea}
 if cadena[1]<>'*' then begin                                  {del fichero de entradas, creando}
  inc(contador);                                               {una entrada SPICE equivalente}
  nodo:=CopiaSubCad(cadena);
  pendiente:=CopiaSubCad(cadena);
  if (pendiente='h') or (pendiente='H') then
   Editor.Lines.Insert(linea, 'Vin'+IntToStr(contador)+ ' '+nodo+' 0 DC 1')
  else begin
   if (pendiente='l') or (pendiente='L') then
    Editor.Lines.Insert(linea, 'Vin'+IntToStr(contador)+ ' '+nodo+' 0 DC')
   else begin
    retardo:=CopiaSubCad(cadena);
    val(retardo,tr,i);
    if i<>0 then
     Editor.Lines.Insert(linea, 'Error en fichero '+input1)
    else begin
     tr:=tr+100;
     if pendiente='1' then begin
      if tiempof<>'' then
       Editor.Lines.Insert(linea, 'Vin'+IntToStr(contador)+ ' '+nodo+' 0 Pwl (0 0 '+
       rcad(tr)+ 'ps 0 '+rcad(tr+1)+ 'ps 1 '+tiempof+' 1)')
      else
       Editor.Lines.Insert (linea,'Vin'+IntToStr(contador)+' '+nodo+
       ' 0 Pwl (0 0 '+rcad(tr)+'ps 0 '+rcad(tr+1)+'ps 1 '+rcad(tr+200)+'ps 1)');
     end else begin
      if pendiente='-1' then begin
       if tiempof<>'' then
```

```
            Editor.Lines.Insert(linea, 'Vin'+IntToStr(contador)+ ' '+nodo+' 0 Pwl (0 1 '+
            rcad(tr)+ 'ps 1 '+rcad(tr+1)+ 'ps 0 '+tiempof+' 0)')
          else
            Editor. Lines.Insert(linea,'Vin'+IntToStr(contador)+' '+nodo+
            ' 0 Pwl (0 1 '+rcad(tr)+'ps 1 '+rcad(tr+1)+'ps 0 '+rcad(tr+200)+'ps 0)');
        end else
          Editor.Lines.Insert(linea, 'Error en fichero '+input1);
       end;
      end;
     end;
    end;
    inc(linea);
   end;
   readln(archent,cadena);
  end;
 end;
 CloseFile(archent);
end;

procedure TFormPFC.InsertarBetayNombre(const betaI:real;const nombreI:string;
        var puntero:PInfoS);
var
 aux:PInfoS;
begin
 new(aux);
 aux^.nombre:=nombreI;
 aux^.bet:=rcad(betaI);
 if puntero=nil then begin
  puntero:=aux;
  puntero^.next:=nil;
 end else begin
  aux^.next:=puntero;
  puntero:=aux;
 end;
```

```
end;

procedure TFormPFC.Eliminar(var lisElim:PInfoS);
var
 auxE:PInfoS;
begin
 while lisElim<>nil do begin
  auxE:=lisElim;
  lisElim:=lisElim^.next;
  dispose(auxE);
 end;
end;

procedure TFormPFC.Trad1Click(Sender: TObject);        {traduce a SPICE}
var
 archspi,archmodel:TextFile;
 spicefich,cad,input,archtecno,alimentacion:string;
 InfoAnalisis:boolean;
 invlista,nor2lista,p_liar:PInfoS;
 l:integer;

 function SubNombre(const wesn:real;var lisSN:PInfoS):string;
 var
  wesncad:string;
  psn:PInfoS;
 begin
  if lisSN<>nil then begin
   wesncad:=rcad(wesn);
   psn:=lisSN;
   while (psn<>nil) and (psn^.bet<>wesncad) do
    psn:=psn^.next;
   if psn<>nil then
    Result:=psn^.nombre
   else
```

```
    Result:= '';
  end else
    Result:= '';
end;

procedure Circuito(var archspi2:TextFile;var linv,lnor2:PInfoS;const InfoAn:boolean);
var
  contx,i,j,k,nodocarga,contn2,conti:integer;          {añade al fichero SPICE el}
  we,wecarga:real;                                     {circuito a simular}
  cargas:boolean;
  cadena,nodoent1,nodoent2,nodosal,puerta,beta,nodoprint:string;
begin
  i:=0;
  contx:=1;
  conti:=0;
  contn2:=0;
  Append(archspi2);
  if InfoAn and (TAS.Editprint.Text<>'') then begin    {nodo solicitado por el usuario}
    wecarga:=0;                                        {para cargarlo}
    nodocarga:=0;
    nodoprint:=TAS.Editprint.Text;
    j:=length(nodoprint);
    while not (nodoprint[j] in ['0', '1'.. '9']) do
      dec(j),
    k:=j;
    while not nodoprint[k]= '(' do
      dec(k);
    nodoprint:=copy(nodoprint,k+1,j-k);
    cargas:=true;
  end else
    cargas:=false;
  while i<Editor.Lines.Count do begin
    cadena:=Editor.Lines[i];
    if cadena<>'' then begin
```

```
if cadena[1]='*' then
 writeln(archspi2,cadena)
else begin
 nodoent2:= '';
 if not (cadena[1] in [' ','c','C']) then begin        {obtiene tipo, nodos y relación de}
  puerta:=CopiaSubCad(cadena);                         {aspecto de la puerta a insertar}
  nodoent1:=CopiaSubCad(cadena);
  if (puerta='nor2') or (puerta='Nor2') or (puerta='NOR2') then
   nodoent2:=CopiaSubCad(cadena);
  nodosal:=CopiaSubCad(cadena);
  beta:=CopiaSubCad(cadena);
  val(beta,we,j);
  we:=5*we;
  if (puerta='inv') or (puerta='Inv') or (puerta='INV') then begin
   cadena:=SubNombre(we,linv);
   if cadena='' then begin
    inc(conti);
    cadena:=IntToStr(conti);
    InsertarBetayNombre(we,cadena,linv);
   end;
   writeln(archspi2,'X'+IntToStr(contx)+' '+nodoent1+' '+nodosal+' INV'+cadena);
   inc(contx);
  end else begin
   cadena:=SubNombre(we,lnor2);
   if cadena='' then begin
    inc(contn2);
    cadena:=IntToStr(contn2);
    InsertarBetayNombre(we,cadena,lnor2);
   end;
   writeln(archspi2,'X'+IntToStr(contx)+' '+nodoent1+' '+nodoent2+' '+nodosal+
                    ' NOR2'+ cadena);
   inc(contx);
  end;
 if cargas then begin                {comprueba si el nodo solicitado corresponde a una}
```

```
      if (nodoprint=nodoent1) or (nodoprint=nodoent2) then      {salida del circuito}
       cargas:=false
      else begin
       if nodosal=nodoprint then
        wecarga:=we;
       val(nodosal,k,j);
       if k>nodocarga then
        nodocarga:=k;
      end;
     end;
     end;
    end;
   end;
   inc(i);
  end;
  if cargas then begin                    {si el nodo solicitado es una salida añade una}
   cadena:=SubNombre(wecarga,linv);       {cadena de tres inversores como carga}
   if cadena='' then begin
    inc(conti);
    cadena:=IntToStr(conti);
    InsertarBetayNombre(wecarga,cadena,linv);
   end;
   nodocarga:=nodocarga+1;
   writeln(archspi2, 'X'+IntToStr(contx)+ ' '+nodoprint+' '+IntToStr(nodocarga)+ ' INV'+
       cadena);
   for i:=1 to 2 do begin
    inc(contx);
    writeln(archspi2, 'X'+IntToStr(contx)+ ' '+IntToStr(nodocarga)+ ' '+
      IntToStr(nodocarga+1)+ ' INV'+cadena);
    nodocarga:=nodocarga+1;
   end;
  end;
  CloseFile(archspi2);
 end;
```

```
begin
 if Editor.Text<>'' then begin
  if not Editor.Modified or SalvarCambios then begin
   valido:=false;
   Liberar;
   ReSim1.Enabled:=false;
   spicefich:=copy(NombreFich,1,pos('.',NombreFich));
   spicefich:=spicefich+'SPI';
   TAS.Edittran.Text:='';
   TAS.Editprint.Text:='';
   if TAS.ShowModal=mrOK then begin          {se muestra cuadro de diálogo para}
    input:=TAS.Edit1.Text;                   {seleccionar ficheros de entradas y de la}
    archtecno:=TAS.Edit1.Text;               {tecnología y especificar datos de análisis}
    if (input<>'') and (archtecno<>'') then begin
     AssignFile(archmodel,input);
     {$I-}
     Reset(archmodel);
     {$I+}
     if IOResult=0 then begin
      CloseFile(archmodel);
      AssignFile(archmodel,archtecno);
      {$I-}
      Reset(archmodel);
      {$I+}
      if IOResult=0 then begin
       CloseFile(archmodel);
       AssignFile(archspi,spicefich);
       Rewrite(archspi);
       writeln(archspi,copy(spicefich,1,pos('.',spicefich)-1));
       CloseFile(archspi);
       InModelo(archspi,archmodel);        {inserta modelo transistores}
       Append(archspi);
       writeln(archspi,'.options acct opts itl5=0 limpts=100000');
```

```
writeln(archspi,' ');
CloseFile(archspi);
invlista:=nil;
nor2lista:=nil;
if (TAS.Edittran.Text<>'') and (TAS.Editprint.Text<>'') then begin
 InfoAnalisis:=true
else
 InfoAnalisis:=false;
Circuito(archspi,invlista,nor2lista,InfoAnalisis);    {inserta circuito}
Append(archspi);
if InfoAnalisis then begin                            {escribe datos análisis}
 writeln(archspi,'* ANALISIS');
 writeln(archspi,'.tran '+TAS.Edittran.Text);
 writeln(archspi,'.print tran '+TAS.Editprint.Text);
end;
writeln(archspi,'.END');
CloseFile(archspi);
NombreFich:=spicefich;
Editor.Text:='';
Editor.Lines.LoadFromFile(NombreFich);
Caption:='SiDSen-'+NombreFich;
Entradas(input,InfoAnalisis);                 {inserta entradas}
l:=16;
alimentacion:=valorfich('vdd',archmodel);
p_liar:=invlista;
while p_liar<>nil do begin          {inserta subcircuitos de inversores}
 Editor.Lines.Insert(l,'.SUBCKT INV'+p_liar^.nombre+ ' 1 2');
 Editor.Lines.Insert(l+1,'ZEI'+p_liar^.nombre+' 2 1 0 EFET03 '+p_liar^.bet);
 Editor.Lines.Insert(l+2,'ZDI'+ p_liar^.nombre+' 3 2 2 DFET03 5');
 Editor.Lines.Insert(l+3,'VDD 3 0 DC '+alimentacion);
 Editor.Lines.Insert(l+4,'.ENDS');
 l:=l+5;
 p_liar:=p_liar^.next;
end;
```

```
      p_liar:=nor2lista;
      while p_liar<>nil do begin          {inserta subcircuitos de NOR2}
       Editor.Lines.Insert(l,'.SUBCKT NOR2'+ p_liar^.nombre+' 1 2 3');
       Editor.Lines.Insert(l+1,'ZEN'+ p_liar^.nombre+'1 3 1 0 EFET03 '+p_liar^.bet);
       Editor.Lines.Insert(l+2,'ZEN'+p_liar^.nombre+'2 3 2 0 EFET03 '+p_liar^.bet);
       Editor.Lines.Insert(l+3,'ZDN'+p_liar^.nombre+' 4 3 3 DFET03 5');
       Editor.Lines.Insert(l+4,'VDD 4 0 DC '+alimentacion);
       Editor.Lines.Insert(l+5,'.ENDS');
       l:=l+6;
       p_liar:=p_liar^.next;
      end;
      Eliminar(invlista);
      Eliminar(nor2lista);
      Editor.Lines.Insert(l,' ');
      Editor.Lines.SaveToFile(NombreFich);
      Editor.Modified:=false;
      Editor.SelStart:=0;
     end else
      MessageDlg('Error. El fichero '+archtecno+' no se encuentra.',mtError,[mbOK],0);
    end else
     MessageDlg('Error. El fichero '+input+' no se encuentra.',mtError,[mbOK],0);
   end else begin          {maneja el error de no especificar ficheros ent y/o dat}
    if input='' then
     MessageDlg('Error. Debe especificar un fichero de entradas (*.ent) para crear el
       fichero SPICE.',mtError,[mbOK],0);
    if archtecno='' then
     MessageDlg('Error. Debe especificar un fichero de la tecnología (*.dat) para crear
       el fichero SPICE.',mtError,[mbOK],0);
   end;
  end;
 end;
end;
```

```
procedure TFormPFC.NuevEntClick(Sender: TObject);
begin
if not Editor.Modified or SalvarCambios then begin
 if pos('.net',NombreFich)<>0 then begin
  valido:=false;
  ReSim1.Enabled:=false;
  Liberar;
 end;
 Editor.Text:='';
 Editor.Lines[0]:='* nodo pendiente retardo';
 Editor.Modified:=false;
 NombreFich:='';
 Caption:='SiDSen-[Sin título]';
 end;
end;

procedure TFormPFC.Ayuda2Click(Sender: TObject);
begin
 Application.HelpCommand(HELP_FINDER,0);
end;

procedure TFormPFC.inicficheros (var f_ent,f_dat,f_net:TextFile);
begin
 Rewrite(f_ent);
 writeln(f_ent,'*nodo pendiente retardo');
 CloseFile(f_ent);
 Rewrite(f_dat);
 writeln(f_dat,'tphl0 6.12');
 writeln(f_dat,'tplh0 25.41');
 CloseFile(f_dat);
 Rewrite(f_net);
 writeln(f_net,'*fichero netlist generado desde el archivo SPICE');
 CloseFile(f_net);
end;
```

```
procedure TFormPFC.ConvertirClick(Sender: TObject);          {traduce desde SPICE}
type
 vectorcad=array [1..7] of string;
 parametrotx=record          {tipo para almacenar parámetros de la tecnología}
              partros:vectorcad;
              nombre:string;
              cuenta:byte;
         end;
 PListCad=^ListCad;
 ListCad= record             {tipo para almacenar señales indeterminadas}
        cad:string;
        sig:PListCad;
    end;

 function extraer(const cadext:string):real;
 var
  i:integer;
  wcad:string;
  valor:real;
 begin
  i:=length(cadext);
  while cadext[i]<>' 'do
   dec(i);
  wcad:=copy(cadext,i+1,length(cadext)-i);
  val(wcad,valor,i);
  if i=0 then
   Result:=valor
  else
   Result:=0;
 end;

 procedure buscaparametrosspi(const cadb:string;var registro:parametrotx);
 const
  parametros: array [1..9] of string = ('RS','CDVC','NP','BETA','DELTA',' VC',
```

```
                    ' VSB','+VC','+VSB');
var
 i,j,posp:integer;      {busca parámetros de la teecnología definidos en la línea SPICE}
 aux:string;
begin
 i:=1;
 while (registro.cuenta<7) and (i<10) do begin
  if i>7 then
   j:=i-2
  else
   j:=i;
  if registro.partros[j]='' then begin
   posp:=pos(parametros[i],cadb);
   if posp<>0 then begin          {parámetro encontrado en la cadena que lee}
    j:=posp+3;
    while (j<length(cadb)) and (cadb[j]<>' ') do
     inc(j);
    if j<>length(cadb) then
     aux:=copy(cadb,posp+length(parametros[i])+1,j-posp-length(parametros[i])-1)
    else
     aux:=copy(cadb,posp+length(parametros[i])+1,
                    length(cadb)-posp-length(parametros[i]));
    if parametros[i]='RS' then
     aux:=ParSPICE(aux,-2);
    if parametros[i]='CDVC' then
     aux:=ParSPICE(aux,4);
    if i>7 then                     {copia el parámetro en el registro}
     registro.partros[i-2]:=aux
    else
     registro.partros[i]:=aux;
    inc(registro.cuenta);
   end;
  end;
  inc(i);
```

```
 end;
end;


function ValorBeta(const Identif:string;var lista:PInfoS):string;        {devuelve el }
var                                {valor de la relación de aspecto que define al subcircuito}
 b:PInfoS;                          {cuyo nombre recibe, de la lista que lo contiene}
begin
 b:=lista;
 while (b<>nil) and (b^.nombre<>Identif) do
  b:=b^.next;
 if b<>nil then
  Result:=b^.bet
 else
  Result:= '!!ERROR!!';
end;

function ValorDC (const c:string): string;        {devuelve la cadena situada al final de}
var                               {la cadena fuente en que se especifica un nivel cte. }
 posc:integer;                                    {de tensión en el fichero SPICE}
begin
 posc:=length(c);
 while (posc<>1) and (c[posc]<>' ') do
  dec(posc);
 if posc<>1 then
  Result:=copy(c,posc+1,length(c)-posc)
 else
  Result:= '';
end;

procedure InEntrada (var fent:TextFile;var cadIn:string);        {inserta en el fichero}
var                               {de entradas una señal de nivel constante encontrada}
 a:integer;                                       {en el fichero SPICE}
 nod,nivel:string;
```

```
begin
 nivel:=ValorDC(cadIn);
 if (nivel='DC') or (nivel='dc') or (nivel[1]= '0') then        {señal de nivel bajo}
  nivel:= 'L'
 else                                                           {señal de nivel alto}
  nivel:= 'H';
 for a:=1 to 2 do
  nod:=CopiaSubCad(cadIn);
 Append(fent);
 writeln(fent,nod+' '+nivel);
 CloseFile(fent);
end;


procedure lineav(var fichentv,fichtecv:TextFile;var cadv:string;
     var vddinsertado,subcircuitov:boolean;var CListEntv,FListEntv:PListCad);
var
 nodo,pend,retdo,v:string;          {maneja las líneas SPICE que comienzan}
 pcion:integer;                     {con el carácter V y definen una señal}
 q:PListCad;
begin
 pcion:=pos('DC',cadv);
 if pcion=0 then
  pcion:=pos('dc',cadv);
 if pcion<>0 then begin
  if subcircuitov then begin        {línea incluida en la definición de subcircuito}
   if not vddinsertado then begin   {especifica tensión de alimentación}
    v:=ValorDC (cadv)
    Append(fichtecv);
    writeln(fichtecv,'vdd '+v);
    CloseFile(fichtecv);
    vddinsertado:=true;
   end;
  end else begin                    {línea fuera de la definición de subcircuito}
   if vddinsertado then             {vdd ya insertada luego define entrada cte.}
```

```
    InEntrada(fichentv,cadv)
   else begin
    v:=ValorDC(cadv);                          {vdd no insertada}
    if (v='DC') or (v='dc') or (v[1]= '0') then    {define nivel bajo, es  entrada}
     InEntrada(fichentv,cadv)
    else begin                    {define nivel alto, función indeterminada}
     new(q);                      {se almacena en lista enlazada}
     q^.cad:=cadv;
     q^.sig:=nil;
     if CListEntv=nil then begin
      CListEntv:=q;
      FListEntv:=q;
     end else begin
      FListEntv^.sig:=q;
      FLitsEntv:=q;
     end;
    end;
   end;
  end;
 end else begin
  if pos('Pwl',cadv)<>0 then begin              {entrada de tipo Pwl}
   v:=CopiaSubCad(cadv);
   nodo:=CopiaSubCad(cadv);
   pcion:=pos('(0 ',cadv);
   delete(cadv,1,pcion+2);
   pend:=CopiaSubCad(cadv);
   if pend='0' then
    pend:='1'
   else
    pend:='-1';
   retdo:=CopiaSubCad(cadv);
   pcion:=pos('s',retdo);
   if pcion<>0 then
    delete(retdo,length(retdo)-1,2);
```

```
    Append(fichentv);
    writeln(fichentv,nodo+' '+pend+' '+retdo);
    CloseFile(fichentv);
   end else begin
    Append(fichentv);
    writeln(fichentv, '¡Error en las entradas declaradas en el fichero SPICE!');
    CloseFile(fichentv);
   end;
 end;
end;

procedure inicregistro(var a:parametrotx);
 var
  i:integer;
begin
 a.nombre:='';
 a.cuenta:=0;
 for i:=1 to 7 do
  a.partros[i]:='';
end;

procedure lineaplus(const linea3:string;var regA,regB:parametrotx);
begin
 if (regA.nombre<>'')and (regA.cuenta=7) then
  buscaparametrosspi(linea3,regB)
 else
  buscaparametrosspi(linea3,regA);
end;

procedure lineapto(var linea1:string;var nombre:string;
  var DCLSUB,puertai_pto:boolean;var wepto,wdpto:real;var reg1,reg2:parametrotx;
  var INVPTO,NOR2PTO:PInfoS);
var
 infolinea:string;
```

```
begin
 infolinea:=CopiaSubCad(linea1);
 if infolinea='.MODEL' then begin          {.MODEL define parámetros de los tx}
  if reg1.cuenta=0 then begin              {lee parámetros incluidos en la línea}
   reg1.nombre:=CopiaSubCad(linea1);
   buscaparametrosspi(linea1,reg1)
  end else begin
   reg2.nombre:=CopiaSubCad(linea1);
   buscaparametrosspi(linea1,reg2);
  end;
 end else begin
  if infolinea='.SUBCKT' then begin        {.SUBCKT define subcircuito}
   nombre:=CopiaSubCad(linea1);            {copia nombre de subcircuito}
   DCLSUB:=true;
   puertai_pto:=true;
   wepto:=0;
   wdpto:=0;
  end else begin
   if (infolinea='.ENDS') or (infolinea='.END') then begin      {final subcircuito}
    DCLSUB:=false;
    if (wepto<>0) and (wdpto<>0) then
     if puertai_pto then                                  {insertar en lista enlazada}
      InsertarBetayNombre(wepto/wdpto,nombre,INVPTO)      {según tipo base}
     else
      InsertarBetayNombre(wepto/wdpto,nombre,NOR2PTO);
   end;
  end;
 end;
end;

procedure lineaz(var cadenaz:string;var wez,wdz:real;var puertainv:boolean;
      var nombreEz,nombreDz: string);
var                              {líneas Z... se incluyen en subcircuito}
 cd1,cd2:string;
```

```
  i:integer;
 begin
  if (nombreEz<>'') and (nombreDz<>'') then begin
   if pos(nombreEz,cadenaz)<>0 then begin        {define tx. enriquecimiento}
    if wez=0 then
     wez:=extraer(cadenaz)          {obtiene we}
    else
     puertainv:=false;              {puerta definida en subcircuito es NOR}
   end else
    wdz:=extraer(cadenaz);          {obtiene wd}
  end else begin            {se desconoce tipo de transistor que define cada identificador}
   for i:=1 to 3 do
    cd1:=CopiaSubCad(cadenaz);
   cd2:=CopiaSubCad(cadenaz);
   if cd1=cd2 then begin                                {define tx deplexión}
    nombreDz:=CopiaSubCad(cadenaz);                     {obtiene identificador}
    cd1:=CopiaSubCad(cadenaz);
    val(cd1,wdz,i);                                         {obtiene wd}
    if i<>0 then
     wdz:=100;
   end else begin                                       {define tx enriquecimiento}
    if nombreEz='' then begin
     nombreEz:=CopiaSubCad(cadenaz);                    {obtiene identificador}
     cd1:=CopiaSubCad(cadenaz);
     val(cd2,wez,i);                                        {obtiene we}
     if i<>0 then
      wez:=1000;
    end else
     puertainv:=false;              {puerta definida en subcircuito es NOR}
   end;
  end;
 end;

 procedure parametrosenriquecimiento(var fichtece:TextFile;
```

```
      const reg:parametrotx);
begin
 Append(fichtece);
 writeln(fichtece,'rse '+reg.partros[1]);
 writeln(fichtece,'ke '+reg.partros[2]);
 writeln(fichtece,'ne '+reg.partros[3]);
 writeln(fichtece,'ce '+reg.partros[4]);
 writeln(fichtece,'vte '+reg.partros[6]);
 CloseFile(fichtece);
end;

procedure parametrosdeplexion(var fichtecd:TextFile;const regd:parametrotx);
begin
 Append(fichtecd);
 writeln(fichtecd,'rsd '+regd.partros[1]);
 writeln(fichtecd,'kd '+regd.partros[2]);
 writeln(fichtecd,'dd '+regd.partros[5]);
 writeln(fichtecd,'vtd '+regd.partros[6]);
 writeln(fichtecd,'vsbd '+regd.partros[7]);
 CloseFile(fichtecd);
end;

procedure construirfichtec(var fichtec:TextFile;const re1,re2:parametrotx;
      const nombreEc,nombreDc: string);
begin
 if (re1.nombre=nombreEc) and (re2.nombre=nombreDc) then begin
  parametrosenriquecimiento(fichtec,re1);
  parametrosdeplexion(fichtec,re2);
 end else begin
  parametrosdeplexion(fichtec,re1);
  parametrosenriquecimiento(fichtec,re2);
 end;
end;
```

```
procedure ConsNetlist(var fichnetcn:TextFile;var INVCN,NOR2CN: PInfoS;
      const conexvddCN; var lineaxCN:integer);        {crea el fichero netlist a partir}
var                                                   {del archivo SPICE}
 n1,n2,n3,betacn,cadcn: string;
 i,po,long:integer;
begin
 if lineaxCN<>0 then begin
  if conexvddcn<>'' then                  {vdd definida fuera de los subcircuitos hay que}
   long:=length(conexvddCN);               { borrar de cada línea del circuito nodos de}
  Append(fichnetcn);                                  {alimentación y tierra}
  while lineaxCN < (Editor.Lines.Count – 1) do begin
   cadcn:=Editor.Lines[lineaxCN];
   if cadcn<>'' then begin
    if (cadcn[1]= 'X') or (cadcn[1]= 'x') then begin          {línea define puerta}
     i:=pos(' ',cadcn);
     delete(cadcn,1,i);
     if conexvddCN<>'' then begin
      po:=pos(conexvddCN,cadcn);
      if po<>0 then                       {borra alimentación y tierra}
       delete(cadcn,po,long);
     end;
     n1:=CopiaSubCad(cadcn);
     n2:=CopiaSubCad(cadcn);
     n3:=CopiaSubCad(cadcn);
     if n3[1] in ['A'.. 'Z', 'a'.. 'z'] then begin          {puerta inversora}
      betacn:=ValorBeta(n3,INVCN);
      writeln(fichnetcn, 'inv '+n1+' '+n2+' '+betacn);
     end else begin                                  {puerta NOR}
      betacn:=ValorBeta(cadcn,NOR2CN);
      writeln(fichnetcn, 'nor2 '+n1+' '+n2+' '+n3+' '+betacn);
     end;
    end else
     if cadcn[1] in ['. ', 'V', 'v'] then          {fin bloque definición circuito}
      lineaxCN:=Editor.Lines.Count;
```

```
    end;
    inc(lineaxCN);
   end;
   Eliminar(INVCN);
   Eliminar(NOR2CN);
   CloseFile(fichnetcn);
  end;
end;

procedure InEntyTec (var CEnt: PListCad;var fichent2,fichtec2: TextFile;
       var conexvddIET:string;const lineaxIET:integer);
var                                   {escribe en el fichero de entradas o de la tecnología}
 vddIn: boolean;            {una señal de nivel alto cuyo cometido no se ha determinado}
 auxp: PListCad;
 pIET: integer;
 cadIET,NodIET: string;
begin
 vddIn:=false;
 while CEnt<>nil do begin
  cadIET:=CEnt^.cad;
  NodIET:=CopiaSubCad(cadIET);
  if not vddIn then begin                                  {vdd no insertada}
   pIET:=pos('DC',cadIET);
   if pIET=0 then
    pIET:=pos('dc',cadIET);
   pIET:=pIET-1;
   while cadIET[pIET]= ' ' do
    dec(pIET);
   NodIET:=copy(cadIET,1,pIET);              {copia nodos sobre los que se aplica}
   if pos(NodIET,Editor.Lines[lineaxIET])<>0 then begin    {si nodos coinciden con}
    conexvddIET:=NodIET;                     { nodos definidos en línea del circuito}
    Append(fichtec2);                                      {es vdd}
    writeln(fichtec2, 'vdd '+ValorDC(cadIET));
    CloseFile(fichtec2);
```

```
        vddIn:=true;
       end else begin                          {si no es entrada a nivel alto}
        pIET:=pos(' ',NodIET);
        delete(NodIET,pIET,length(NodIET)-pIET+1);
        Append(fichent2);
        writeln(fichent2,NodIET+' h');
        CloseFile(fichent2);
       end;
      end else begin                    { vdd insertada, luego es entrada a nivel alto}
       NodIET:=CopiaSubCad(cadIET);
       Append(fichent2);
       writeln(fichent2,NodIET+' h');
       CloseFile(fichent2);
      end;
      auxp:=CEnt;
      CEnt:=CEnt^.sig;
      dispose(auxp);
     end;
     Append(fichent2);
     writeln(fichent2, 'end');
     CloseFile(fichent2);
    end;
                                  {Convertir1Click}
var
 fichent,fichdat,fichnet:TextFile;
 contador,lineax:integer;
 vddinsertada,pinversor,subdec,almacenada:boolean;
 aux,cadena,netnombre,entnombre,datnombre,nombresubckt,nombreE,nombreD,
       conexvdd:string;
 we,wd:real;
 registro1,registro2:parametrotx;
 INV,NOR2: PInfoS;
 CListEnt,FListEnt: PListCad;
begin
```

```
if Editor.Text<>'' then begin
 if (not Editor.Modified)or SalvarCambios then begin
  netnombre:=copy(NombreFich,1,pos('.',NombreFich));          {inicializa variables}
  entnombre:=netnombre+'ent';
  datnombre:=netnombre+'dat';
  netnombre:=netnombre+'net';
  AssignFile(fichent,entnombre);
  AssignFile(fichdat,datnombre);
  AssignFile(fichnet,netnombre);
  vddinsertada:=false;
  almacenada:=false;
  lineax:=0;
  conexvdd:= '';
  pinversor:=true;
  subdec:=true;
  we:=0;
  wd:=0;
  nombresubckt:='';
  nombreE:= '';
  nombreD:= '';
  inicregistro(registro1);
  inicregistro(registro2);
  inicficheros(fichent,fichdat,fichnet);
  contador:=0;
  INV:=nil;
  NOR2:=nil;
  CListEnt:=nil;
  FListEnt:=nil;
  while contador<(Editor.Lines.Count-1) do begin        {lee fichero SPICE hasta final}
   cadena:=Editor.Lines[contador];
   if cadena<>'' then begin
    case cadena[1] of                          {según carácter inicial extrae información}
     '.':lineapto(cadena,nombresubckt,subdec,pinversor,we,wd,registro1,registro2,
          INV,NOR2);
```

```
        '+':lineaplus(cadena,registro1,registro2);
        'X','x': if not almacenada then begin
                    lineax:=contador;
                    almacenada:=true;
                 end;
        'V','v':lineav(fichent,fichdat,cadena,vddinsertada,subdec,CListEnt,FListEnt);
        'Z','z':lineaz(cadena,we,wd,pinversor,nombreE,nombreD)
       end;
      end;
      inc(contador);
    end;
    FListEnt:=nil;
    construirfichtec(fichdat,registro1,registro2,nombreE,nombreD);    {fichero tecnología}
    InEntyTec(CListEnt,fichent,fichdat,conexvdd,lineax);
    ConsNetlist(fichnet, INV,NOR2,conexvdd,lineax);                   {fichero netlist}
    aux:='Finalizó la conversión de ficheros. Se han creado los archivos '+
                netnombre+', '+datnombre+', '+entnombre;
    MessageBox(0,PChar(aux),'SiDSen',MB_OK+MB_ICONINFORMATION);
   end;
 end else
   MessageDlg('No hay fichero SPICE abierto',mtError,[mbOK],0);
end;

end.

program SiDSen;

uses
 Forms,
 princ5 in 'princ5.pas' {FormPFC},
 cuadro_t in 'cuadro_t.pas' {TAS},
 cuadro_sim in 'cuadro_sim.pas' {DatSim},
 sec4 in 'sec4.pas' {SelSalv},
 sim4 in 'sim4.pas',
```

```
  res4 in 'res4.pas' {Resultado};

{$R *.RES}
begin
  Application.Initialize;
  Application.HelpFile := 'C:\trabajos\AYUDASiDSen.HLP';
  Application.CreateForm(TFormPFC, FormPFC);
  Application.CreateForm(TTAS, TAS);
  Application.CreateForm(TDatSim, DatSim);
  Application.CreateForm(TSelSalv, SelSalv);
  Application.CreateForm(TResultado, Resultado);
  Application.Run;
end.
```