Detección de los Transistores de un Circuito Integrado a Partir del Layout

Carlos E. Christoffersen, Luis A. Lahoz, María I. Schiavon

Resumen— **En este trabajo describimos la representación en memoria del layout de un circuito integrado y los algoritmos necesarios para reconocer transistores MOS. Estos algoritmos se utilizaron en el programa MICRO, que es un extractor de circuitos desarrollado en el Laboratorio de Microelectrónica de la Universidad Nacional de Rosario.**

I. Introducción

Veremos primero un diagrama de bloques con los pasos que sigue el algoritmo que hacemos referencia en este trabajo (figura 1). Consideramos que disponemos del layout del circuito integrado representado de manera conveniente. Aunque solo describiremos la extracción de transistores, utilizaremos un algoritmo que nos permitirá reconocer cualquier elemento que se encuentre en el circuito.

En el primer bloque se efectúan operaciones lógicas entre capas. Estas operaciones dan como resultado nuevas capas que sirven para reconocer los elementos del circuito.

El paso siguiente es agrupar los rectángulos en islas. Llamamos *isla* a un conjunto de rectángulos, todos pertenecientes a la misma capa, cuya unión produce una sola figura geométrica. Como consecuencia de esta definición, los rectángulos pertenecientes a una isla están conectados eléctricamente. De esa forma agrupamos por ejemplo a todos los rectángulos que forman el canal de un transistor, o todos los rectángulos que forman una conexión entre dos puntos del circuito, siempre que esta se realice utilizando una sola capa. Además, cada isla contiene información de cómo se relaciona con las demás islas.

Por medio del análisis de las islas creadas en la etapa anterior, se extraen los elementos que componen el circuito. Para ello vamos recorriendo las islas, y cada vez que encontramos una isla asociada al canal de un transistor, extraemos el transistor correspondiente. Con un razonamiento similar se pueden encontrar los diodos, las capacidades y resistencias. Al mismo tiempo que extraemos un elemento del circuito, debemos conectar sus terminales.

II. Representación del layout en memoria

Consideraremos layouts con geometría manhatan, es decir que todas las figuras intervinientes se pueden descomponer en rectángulos. El problema se reduce entonces a buscar la forma más conveniente de representar un conjunto de rectángulos.

Carlos E. Christoffersen pertenece al grupo de investigación del Laboratorio de Microelectrónica, Facultad de Cs. Exactas, Ing. y Agrimensura de la Universidad Nacional de Rosario, Av. Pellegrini 250, 2000 Rosario, Argentina. e-mail: cchristo@unromi.edu.ar

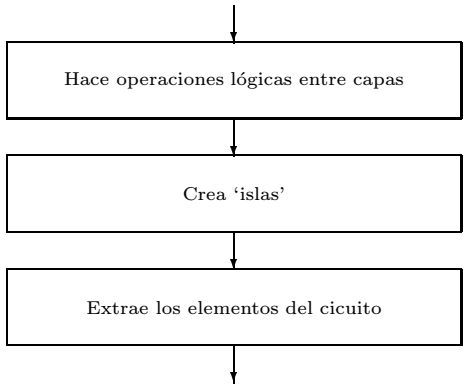


Fig. 1. Diagrama de bloques

La estructura que contiene cada rectángulo tiene la siguiente forma:

```
struct rectang {
long left;
long right;
long upper;
long lower;
LAYER layer;
unsigned num;
};
```

Los campos *left, right, upper* y *lower* contienen los límites del rectángulo, el campo *LAYER* contiene la capa a la que pertenece y el campo *num* es un número que en principio es único para cada rectángulo y sirve para su identificación biunívoca.

A su vez, todos los rectángulos están agrupados en un árbol binario balanceado [8], ordenados por su número de orden. En la figura 3 tenemos el árbol correspondiente a los rectángulos de la figura 2. En cada nodo del árbol hay en realidad un puntero hacia una lista de rectángulos, pero hasta ahora todas las listas contienen exactamente un elemento que es el rectángulo correspondiente al nodo. Observemos que el número correspondiente a cada rectángulo solo sirve para identificarlo. Ese número se usa como clave para el árbol.

III. Operaciones lógicas entre capas

Las operaciones que se realizan en esta etapa con las capas del layout son para permitir el reconocimiento de los elementos de circuito más adelante. Cuando mencionamos a

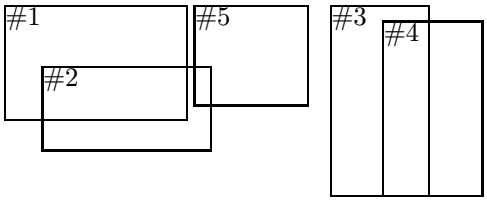


Fig. 2. Rectángulos de un layout numerados

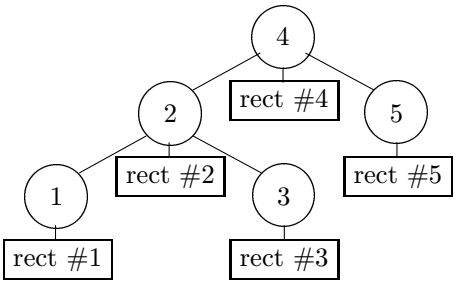


Fig. 3. Arbol AVL conteniendo rectángulos

una capa en realidad estamos pensando en todos los elementos que pertenecen a la misma, que en nuestro caso son rectángulos. La siguiente es una lista con los nombres que daremos a cada una de las capas del layout que nos interesan:

CNWI: *pozo N*
CTOX: *área activa*
CPOL: *polisilicio*
CNPI: *implante N*
CPPI: *implante P*
CCON: *contacto*
CMEI: *metal1*
CVIA: *contacto metal1-metal2*
CME2: *metal2*

El resultado de la operación AND ($\cdot$) entre dos capas es una capa que solo contiene todas las porciones de área que estaban cubiertas por las dos capas originales a la vez, es decir, la intersección de las mismas (figura 4). En la operación XOR, en cambio, la capa resultante contiene el complemento de la unión de las dos capas originales con respecto a la intersección, como se ve en el ejemplo de la figura 5. Notemos que una vez hecha la operación, podemos reemplazar los elementos de una de las capas originales con los resultantes de la operación.

Las operaciones que se realizan en esta parte del programa son las que siguen, y se realizan en el orden indicado:

$$ACTN \leftarrow CTOX \cdot CNPI \quad (1)$$
$$CTOX \leftarrow (CTOX \oplus CNPI) \cdot CTOX \quad (2)$$
$$SUBN \leftarrow ACTN \cdot CNWI \quad (3)$$

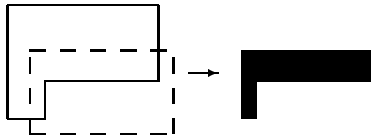


Fig. 4. Ejemplo de una operación AND

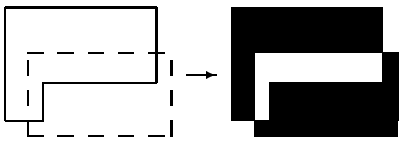


Fig. 5. Ejemplo de una operación XOR

$$ACTN \leftarrow (ACTN \oplus CNWI) \cdot ACTN \quad (4)$$
$$SUBP \leftarrow CTOX \cdot CPPI \quad (5)$$
$$CTOX \leftarrow (CTOX \oplus CPPI) \cdot CTOX \quad (6)$$
$$ACTP \leftarrow SUBP \cdot CNWI \quad (7)$$
$$SUBP \leftarrow (SUBP \oplus CNWI) \cdot SUBP \quad (8)$$
$$NCHA \leftarrow ACTN \cdot CPOL \quad (9)$$
$$ACTN \leftarrow (ACTN \oplus CPOL) \cdot ACTN \quad (10)$$
$$PCHA \leftarrow ACTP \cdot CPOL \quad (11)$$
$$ACTP \leftarrow (ACTP \oplus CPOL) \cdot ACTP \quad (12)$$

Como vemos, se repiten las mismas operaciones de 1 y 2 para distintas capas. Esto es así porque se realizan mediante una función llamada *andxor*() que realiza las dos operaciones a la vez, lo cual es conveniente para cuando se quiere separar el drain y el source de los transistores.

Analicemos ahora el significado de las nuevas capas. La capa ACTN es inicialmente la intersección entre la capa de difusión y el implante N, por lo que se corresponde casi con todas las partes del circuito integrado en las que hay zona activa N, la excepción en esta correspondencia es en los lugares que hay intersección entre ACTN y CPOL. Luego de la operación 2, la capa CTOX queda con un 'agujero' en todos los lugares en los que hay ACTN. Si seguimos todas las operaciones hasta el final podemos concluir cual es el significado de las nuevas capas:

ACTN: contiene todos los rectángulos que forman los drain y sources de los transistores MOS, así como también cualquier región activa tipo N que esté sobre el sustrato P, es decir que forme un diodo.

SUBN: está formada por las regiones N dentro de un pozo N, es decir que no forman una juntura PN, y que son generalmente utilizadas como contactos al pozo N.

ACTP: es análoga a la ACTN pero con regiones P sobre pozo N, que forman diodos y pueden ser drain o source de transistores.

SUBP: es análoga a la SUBN pero con regiones P sobre sustrato P, suele utilizarce para hacer contactos a sus-

trato ya que no se forma un diodo.

NCHA: contiene todos los canales de los transistores tipo N del circuito integrado.

PCHA: contiene todos los canales de los transistores tipo P.

La función *andxor*() está implementada mediante una subrutina que realiza la intersección entre los rectángulos de las capas operando y mediante el análisis de los pares de rectángulos que se interceptan, va creando los rectángulos de la capa resultado. Una parte fundamental de esta rutina es entonces el algoritmo que detallamos a continuación (para más detalles, ver [6]), que se encuentra implementado en la función *inters*().

Algoritmo para encontrar intersecciones de rectángulos

Una de las operaciones fundamentales en la extracción de circuitos es la operación lógica AND entre capas. Como podemos descomponer en rectángulos a todas las figuras geométricas que componen una capa, la operación AND entre dos capas consistirá en encontrar el conjunto de intersecciones entre los rectángulos que componen cada capa.

Dadas las coordenadas de n rectángulos, podemos considerar cada par, y en tiempo $O(1)$ decidir si ese par se intersecta. Esto es, los rectángulos R y S se intersectan si y solo si al menos uno de los cuatro vértices de S está dentro de R. Probar si un punto está dentro de un rectángulo es simple. De esta forma, encontrar todos los pares de rectángulos que se intersectan en un tiempo $O(n^2)$ es muy simple. Sin embargo, en diseños con un millón rectángulos necesitaríamos 10^{12} pasos para encontrar las intersecciones.

Por lo anterior, veremos otro método mejor, debido a McCreight [1980], el cual toma un tiempo $O(n \log n + i)$, donde i es el número de intersecciones encontradas. Este método asume que las coordenadas de los rectángulos son enteras y residen en un cuadrado que tiene un lado que es $O(n)$.

El problema dinámico unidimensional

Para simplificar el problema de encontrar las intersecciones, consideremos una línea horizontal que recorre el cuadrado que contiene los rectángulos de abajo hacia arriba como se muestra en la figura 6. Llamaremos a esa línea 'línea de búsqueda'. La intersección de cada rectángulo con esa línea será un intervalo. Cuando la línea de búsqueda encuentra un nuevo rectángulo en su camino aparece un nuevo intervalo dentro de la línea, si dos rectángulos se intersectan los intervalos correspondientes se superponen. Cuando un rectángulo ya no intersecta la línea de búsqueda su intervalo desaparece.

Entonces, el problema de encontrar las intersecciones de rectángulos se puede expresar como el problema de encontrar los intervalos (unidimensionales) que se superponen con un intervalo dado. Sin embargo, el problema unidimensional es dinámico en el sentido que el conjunto de intervalos cambia a medida que la línea de búsqueda se va desplazando. Para encontrar las intersecciones de los rectángulos, cada vez que aparece un nuevo intervalo reportamos todas las intersecciones entre los intervalos de la línea de búsqueda.

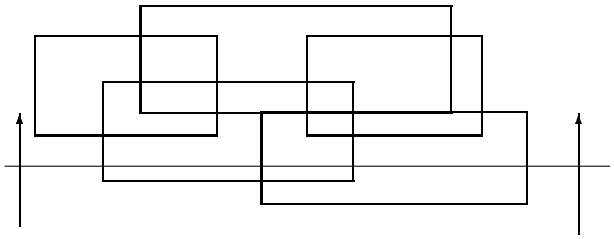


Fig. 6. Línea exploradora

Grupos de intervalos

Nuestro próximo paso es considerar cómo encontrar los intervalos que se superponen con un intervalo dado, y debemos hacer eso en un tiempo que sea proporcional al número de intersecciones que encontremos. Para eso, no podemos organizar el conjunto actual de intervalos por ejemplo por el extremo izquierdo. Aún si incluyéramos una estructura de árbol que nos permita encontrar los extremos izquierdos rápidamente, todavía nos quedaría una gran cantidad de trabajo para encontrar todas la superposiciones con un intervalo dado y encontrar unas pocas.

Para entender el algoritmo que utilizaremos introducimos aquí el concepto de grupo de intervalos. Cada grupo está definido por un par de enteros que son múltiplos consecutivos de la misma potencia de 2. Los enteros i y j definen el grupo $G_{i,j}$; por ejemplo podríamos tener $G_{10,12}$, $G_{48,64}$ o $G_{48,56}$, pero no $G_{16,48}$. Eso es debido a que 10 y 12 son 5×2 y 6×2, 48 y 64 son 2×16 y 3×16, 48 y 56 son 6×8 y 7×8. Sin embargo, no hay potencia de 2 (2^k) tal que 16 y 48 sean $i \times 2^k$ y $(i+1) \times 2^k$ para algún entero i.

El grupo $G_{i,j}$ está formado por el conjunto de los intervalos dentro del conjunto actual que:

1. Tienen el extremo izquierdo mayor que i.
2. Tienen le extremo derecho menor que j.
3. El extremo izquierdo no es mayor que $(i + j)/2$ y el derecho no es menor que $(i + j)/2$.

Podemos crear un árbol de grupos, en donde el hijo izquierdo de $G_{i,j}$ es $G_{i,(i+j)/2}$ y el derecho es $G_{(i+j)/2,j}$. Un hecho importante es que cualquier intervalo propiamente contenido en el intervalo $[i, j]$, donde i y j son múltiplos consecutivos de potencias de 2, pertenecerá a $G_{i,j}$ o uno de sus descendientes. No necesariamente pertenece a $G_{i,j}$ porque es posible que no contenga el punto medio de $[i, j]$. En la figura 7 hay un ejemplo de árbol de grupos y los intervalos que pertenecen a cada nodo.

Implementación de un Grupo

Es mucho más fácil encontrar los miembros de un grupo que intersectan el intervalo $[x_1, x_2]$ que encontrar las superposiciones de un intervalo con un conjunto aleatorio de

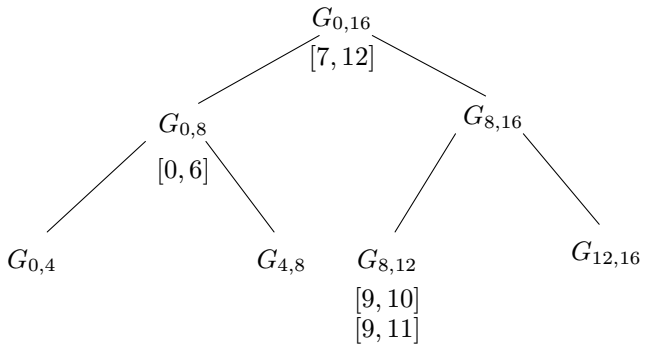


Fig. 7. Ejemplo de árbol de grupos

intervalos. Sea el grupo $G_{i,j}$, y sea m, el punto medio $(i+j)/2$. Entonces si $x_1 \leq m \leq x_2$, cada intervalo en $G_{i,j}$ se intersecta con $[x_1, x_2]$; en particular, comparten el punto m.

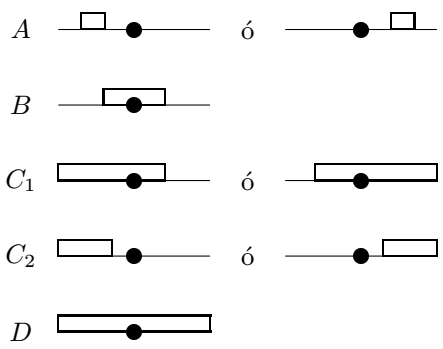
Si $x_2 < m$, entonces el intervalo $[x_3, x_4]$ en $G_{i,j}$ intersecta $[x_1, x_2]$ si y solo si $x_3 \leq x_2$. Los valores de x_1 y x_4 no importan, ya que sabemos que $x_4 \geq m$ porque el intervalo pertenece al $G_{i,j}$. Por lo tanto no es posible que $[x_3, x_4]$ quede enteramente a la izquierda de $[x_1, x_2]$. Podemos de esta manera examinar los intervalos en $G_{i,j}$ en orden de sus extremos izquierdos. Al principio, cada intervalo que encontremos superpone $[x_1, x_2]$, y tardamos un tiempo proporcional al número de intersecciones reportadas. Cuando encontramos un intervalo cuyo extremo izquierdo es mayor que x_2, dejamos de examinar los intervalos restantes. Notemos que la propiedad del grupo que todos sus intervalos contienen el punto medio, es esencial. De no ser por esto, haciendo la búsqueda en orden de los extremos izquierdos encontraríamos muchos intervalos enteramente a la izquierda de $[x_1, x_2]$.

De la misma manera, si el intervalo $[x_1, x_2]$ no contiene el punto m porque $x_1 > m$, deberemos examinar los intervalos en orden de sus extremos derechos, primero los mayores, hasta que encontremos uno cuyo extremo derecho sea menor que x_1.

Buscando en el Arbol de Grupos

Debemos considerar ahora como encontrar todos los intervalos que superponen un intervalo dado en un tiempo que es $O(\log n)$ más un término que es proporcional al número de intersecciones encontradas. Lo que haremos será una rutina recursiva que busque intersecciones en un grupo y todos sus descendientes en el árbol de grupos, y luego aplicaremos esa rutina a la raíz del árbol. Otra idea clave es que debemos clasificar la forma en que un intervalo dado se relaciona con el intervalo $[i, j]$, donde $G_{i,j}$ es el grupo de la raíz del árbol considerado. Las cinco situaciones posibles se muestran en la figura 8.

Por ejemplo, si el intervalo en cuestión queda enteramente a un costado del punto medio, la superposición es tipo A. En la tipo B, el segmento contiene el punto medio, pero no alcanza ninguno de los extremos. Sabemos que un intervalo con superposición tipo B intersecta a todos los



Clase	Clase del hijo izquierdo	Clase del hijo derecho
A	A ó B	Intersección vacía
B	C	C
C_1	D	C
C_2	C	Intersección vacía
D	D	D

Fig. 8. Tipos de superposiciones y reglas de transición

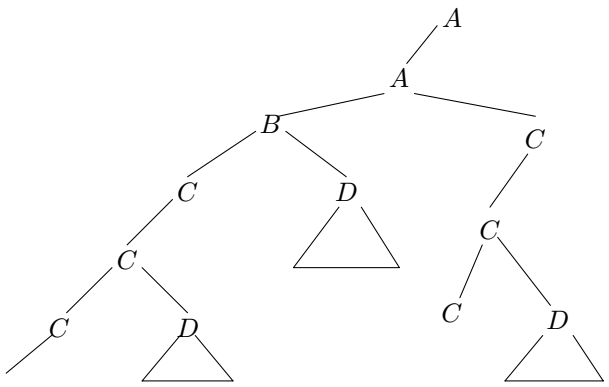


Fig. 9. Propagación del tipo de intersección en un árbol de grupos típico

miembros del grupo.

Otra observación importante es que la clase de un intervalo con respecto al $G_{i,j}$, afecta la clase de todos los grupos descendientes. En la figura 9 están resumidas las reglas que determinan como se propaga la clase hacia abajo en el árbol de grupos. Cuando hay dos formas simétricas (casos A y C), asumimos la primera forma, en donde el intervalo está en la parte izquierda. Además, clase "C" significa tanto clase C_1 como C_2.

Cuando comenzamos la búsqueda de intersecciones entre los intervalos que están en el árbol de grupos y un intervalo dado, necesitamos atravesar el árbol de grupos de la manera sugerida en la figura 9.

La búsqueda de intersecciones con un intervalo dado por el árbol de grupos tarda un tiempo $O(\log n + i)$, en donde i es el número de intersecciones reportadas. El tiempo transcurrido lo cargamos (en clases distintas de la D) al nodo en sí si no hay intersecciones de segmentos en ese nodo o a las intersecciones encontradas si las hay.

No podemos cargar el costo en tiempo de examinar los

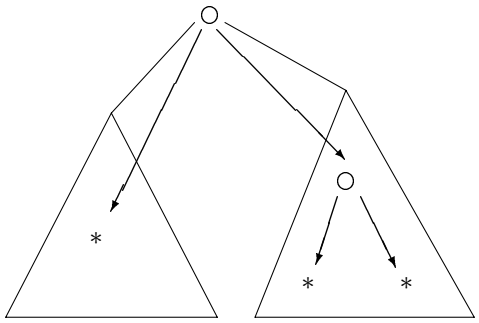


Fig. 10. Uso de punteros extra para encontrar grupos no vacíos

nodos clase D a los nodos en sí porque puede haber más de $O(\log n)$ de ellos. Desafortunadamente, podría suceder que todos, o casi todos los nodos tipo D estén vacíos (esto es, no contengan segmentos), y no habría superpocisiones a las que atribuir el tiempo transcurrido. Por lo tanto, debemos modificar el árbol de grupos de manera que cada nodo tenga un registro con el número de intervalos que contiene él y sus descendientes. Además, un nodo con un solo descendiente en el subárbol izquierdo que contiene un conjunto no vacío de intervalos deberá apuntar directamente hacia ese nodo, de manera que no sea necesario seguir caminos largos a través del árbol para alcanzar el nodo no vacío. Por la misma razón, grupos con más de un descendiente no vacío en el subárbol izquierdo apuntarán directamente hacia el menor común ascendiente de esos grupos no vacíos. Se usa un puntero similar para encontrar rápidamente los grupos no vacíos en el subárbol derecho.

La figura 10 ilustra esos punteros extra en el árbol de grupos. Notemos que los punteros hacia los hijos derecho e izquierdo aún son necesarios, ya que son necesarios cuando insertamos y eliminamos intervalos en los distintos grupos. Sin embargo, para encontrar los grupos cuyos miembros intersectan un intervalo dado clase D, usamos los punteros indicados con flechas.

Si cuando examinamos un subárbol de nodos clase D seguimos los punteros que acabamos de describir, esa porción del árbol que está siendo examinada puede verse como un árbol binario. Podemos usar el hecho que un árbol binario tiene una hoja más que el número de nodos interiores para concluir que más de la mitad de los nodos atravesados tienen grupos no vacíos. A la vista de un nodo clase D, reportamos que el intervalo en cuestión superpone todos los intervalos dentro del grupo de ese nodo. De esa manera, podemos cargar el costo de buscar en todos los nodos clase D a las intersecciones reportadas, siendo que el número de superposiciones es al menos la mitad del número de nodos analizados.

El Algoritmo de McCreight para reportar las intersecciones de rectángulos

ENTRADA: un conjunto de rectángulos.
SALIDA: Una lista con los pares de rectángulos que se intersectan.
METODO: En lo que sigue, asumiremos que las coordenadas de los vértices de los rectángulos están en el rango $[0, n-1]$, y que n es un límite superior en el número de rectángulos. Si ese no es el caso, habrá que transformar las coordenadas de los rectángulos para que se cumplan las condiciones. El algoritmo usa un conjunto de intervalos $ACTIVE$, que representan los rectángulos. Cuando reportamos una intersección de intervalos, en realidad estamos listando el par de rectángulos correspondientes en el archivo de salida.
En la figura 11 están las etapas a seguir. Las etapas 7 y 9, que incluyen inserción y eliminación de intervalos se dividen en dos partes. En la primera, encontramos el grupo apropiado mediante una búsqueda binaria. Ya que es posible que estemos insertando un intervalo en un grupo vacío o eliminando el último intervalo de un grupo, puede ser necesario ajustar los punteros de la figura 10 en los nodos antecesores del nodo en cuestión. En la segunda insertamos o eliminamos el segmento de la estructura que contiene los intervalos en el grupo.

1. Ordenar los rectángulos en orden de la altura del borde inferior, los más bajos primero, y también ordenarlos en orden del borde superior, los más bajos primero.
2. $ACTIVE$ = conjunto vacío.
3. Inicializar el árbol de grupos, de manera que todos los grupos están vacíos (en la práctica, inicializaríamos el árbol a medida que es requerido por las inserciones que efectuamos).
4. **for** $y = 0$ **to** $n - 1$ **do begin**
5. **for** cada rectángulo con borde inferior y, borde izquierdo x_1 y derecho x_2 **do begin**
6. Reportar las superposiciones del intervalo $[x_1, x_2]$ con los intervalos en $ACTIVE$
   end
7. **for** cada rectángulo con borde superior y, borde izquierdo x_1 y derecho x_2 **do**
8. eliminar $[x_1, x_2]$ de $ACTIVE$
9. **end**

Fig. 11. Algoritmo para reportar las intersecciones de rectángulos

V. Creación de islas

El objetivo de agrupar los rectángulos en islas es identificar las partes que componen a cada elemento del circuito. Los rectángulos que formen el canal de un transistor MOS estarán agrupados en una isla exclusiva para ellos. Lo mismo ocurrirá para los que formen el drain, el source, o una tira de metal que conecta dos partes del circuito.

Veamos en un ejemplo como se representa una isla. En la figura 2 tenemos un conjunto de rectángulos que supondremos pertenecientes a la misma capa. Esos rectángulos estarán inicialmente agrupados en un árbol como el de la figura 3. Cuando se agrupan los rectángulos en islas, el árbol de rectángulos se modifica para que quede como en la figura 12. En cada nodo del árbol tenemos un pun-

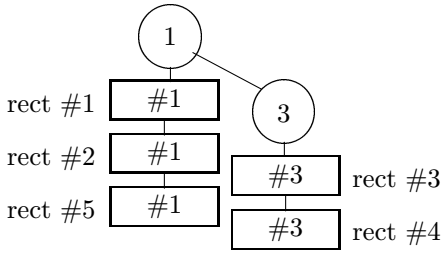


Fig. 12. Arbol AVL conteniendo islas

tero hacia una lista de rectángulos que forma una isla. Notemos que las dos islas que se han formado no necesariamente están numeradas en forma consecutiva (aunque cada número de isla es único), y que el campo *num* en cada rectángulo ahora contiene el número de isla; el número original del rectángulo se pierde porque ya no es necesario. Los números de islas son las claves del árbol binario.

Para hacer la agrupación en islas primero se encuentran las intersecciones entre todos los rectángulos del árbol binario (usando la función *inters*(), que es la implementación del algoritmo explicado) da como resultado un archivo temporario con todos los pares de rectángulos que se interceptan, sean de la misma capa o no. Luego se analizan esos pares y se van formando los grupos de rectángulos que serán las islas. La rutina que se encarga de formar los grupos se llama *merge*(), y funciona de la siguiente manera: para cada grupo, mantiene la cuenta del número de rectángulos y un puntero al primer rectángulo de una lista enlazada que contiene todos los rectángulos del grupo. Como ya vimos, la estructura que contiene los rectángulos cuenta con un campo *num* que corresponde al número de grupo al que pertenece el mismo. Cuando encontramos un par de rectángulos R y S que se intersectan, debemos mezclar los grupos i y j correspondientes a esos rectángulos. Si $i = j$, no necesitamos hacer nada, los grupos ya están mezclados. Si $i \neq j$, determinamos cual de los dos grupos, digamos i, tiene menor número de elementos. Entonces recorremos la lista de i y cambiamos el número de grupo en cada rectángulo a j. Finalmente, agregamos a la lista de j la lista de i y el grupo i deja de existir.

La estructura que forma los nodos del árbol binario que contiene las islas es la siguiente:

```
struct nodetype {
long key; /* la clave para la busqueda */
RCTLST info; /* puntero a una lista de rect.  */
RCTLST relac; /* punt.  a lista de rect.
relacionados */
unsigned numrec; /* numero de rect.  contenidos
en info */
char bal; /* da el balance del nodo */
struct nodetype *left;
struct nodetype *right;
};
```

Además de los campos propios para el manejo del árbol binario balanceado (*key, bal, left, right*), tenemos tres campos con información referida a la isla:

info: es un puntero a la lista de rectángulos que forma la isla propiamente dicha.
relac: es un puntero a una lista de rectángulos que contiene exactamente un rectángulo de cada isla que tiene intersección no vacía con la isla del nodo.
numrec: es el número de rectángulos que forman la isla.

VI. Extracción de los transistores del circuito

El punto de partida para la extracción un transistor es una isla tipo NCHA o PCHA. Por lo tanto nuestro programa deberá recorrer el árbol de islas, y detenerse a reconocer el transistor correspondiente cada vez que encuentre una de este tipo. Esa isla contiene la geometría del canal del transistor. Para encontrar las islas correspondientes al drain y source lo que hacemos es recorrer la lista de islas relacionadas a la isla del canal. Solo puede haber dos islas tipo ACTN (para el caso de una isla NCHA). Arbitrariamente tomamos una de esas dos islas como drain y la otra como source, ya que no hay distinción en el modelo SPICE entre drain y source. Los nodos a los que están conectados los terminales de transistor son los números de isla que forman el canal, el drain y el source. El terminal de sustrato irá conectado a la isla de pozo N en el que está el transistor (si el transistor es tipo N), o al sustrato si el transistor es tipo P. El sustrato se toma en el programa como nodo 0.

Para cada transistor, necesitamos además encontrar los siguientes parámetros:

L : largo del canal.
W : ancho del canal.
AS : área del source.
PS : perímetro del source.
NRS : número de cuadrados del source.
AD : área del drain.
PD : perímetro del drain.
NRD : número de cuadrados del drain.

Los datos que disponemos para determinar estos valores son las islas que componen cada parte del transistor. Tenemos una isla para cada una de las tres partes de las que debemos extraer las características geométricas, esto es, una isla para el drain, una para el canal y otra para el source. Sin embargo, no es fácil obtener por ejemplo el perímetro de la isla que forma el drain de un transistor directamente a partir de los rectángulos que la forman (que es lo que disponemos hasta ahora). Veremos ahora como se resuelve este problema en nuestro programa.

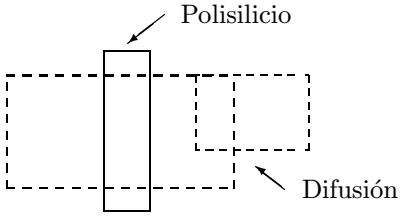


Fig. 13. Rectángulos que componen un transistor antes de las operaciones lógicas

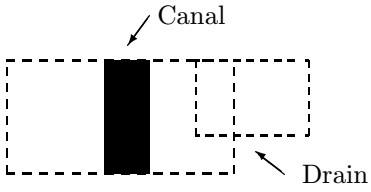


Fig. 14. Rectángulos que componen un transistor luego de las operaciones

Matriz de análisis para transistores

Tomemos el caso de la figura 13, tenemos los rectángulos de difusión y polisilicio que forman un transistor (que suponemos tipo N) tal como se lee desde el layout, observemos que el drain y source [1] comparten un rectángulo de difusión. Luego de las operaciones lógicas entre las capas, los rectángulos quedan como en la figura 14, el rectángulo de difusión grande original ha sido reemplazado por dos rectángulos en la capa ACTN, uno a cada lado del canal, y el polisilicio sobre la difusión ha originado un rectángulo de la capa NCHA.

Si queremos saber por ejemplo el área del source no tenemos más que calcular el área del rectángulo que lo forma, pero para el drain no es tan simple, porque está formado por dos rectángulos que se superponen. Para solucionar este problema, debemos usar una representación que nos permita analizar la geometría en forma más sencilla. Surge entonces la *matriz de análisis*.

El número de filas y columnas de la matriz se determina en función de los bordes de los rectángulos que forman el transistor, y a cada elemento de la matriz le asignamos una parte del área total que ocupa el mismo. Para comprender mejor esto último pensemos que la matriz completa representa un rectángulo que contiene todo el transistor. A ese rectángulo lo subdividimos en rectángulos menores formando una cuadrícula en la que cada línea pasa por el borde de algún rectángulo de los que componen el transistor. En la figura 15 se ve la ilustración de la matriz para el transistor de la figura 14. Podemos ver que de esta forma cada elemento de la matriz representa una porción del área total que solo puede ser source, drain, canal o ninguna de las anteriores. Esto es porque elegimos los bordes de los rectángulos para hacer las divisiones en el área, de esta forma los límites entre regiones del transistor producen una división en los elementos de la matriz.

[1] Suponemos el source a la izquierda y el drain a la derecha.

S	C	D	D	D
S	C	D	D	N

D: Drain
S: Source
C: Canal
N: Nada

Fig. 15. Matriz de análisis para un transistor

La forma de implementar la matriz en memoria es por medio de un arreglo y dos vectores. El arreglo es la matriz propiamente dicha y tiene las mismas filas y columnas de la matriz, cada elemento contiene la región a la que pertenece. Los vectores son uno para el eje x y el otro para el eje y. El vector del eje x tiene un elemento por columna de la matriz, y cada uno de ellos contiene el ancho de los rectángulos que forman cada columna de la misma. De manera similar, el vector para el eje y contiene las alturas de los rectángulos para cada fila. El procedimiento para construir la matriz es el siguiente:

1. Primero se crean dos vectores con las coordenadas de todos los bordes de los rectángulos que forman el elemento, uno para cada eje del plano.
2. Se ordenan estos vectores de menor a mayor, se eliminan los bordes repetidos, se determina la cantidad de filas y columnas de la matriz y las medidas de los rectángulos que la componen.
3. Para cada rectángulo que pertenece a alguna de las islas que forman el elemento se asignan los elementos de la matriz que estén cubiertos por ese rectángulo como pertenecientes a la capa de la isla.

Teniendo esta matriz, el cálculo de las áreas de drain y source se simplifica, ya que por no haber superposiciones entre los rectángulos que la forman, la tarea se reduce a sumar las áreas de los rectángulos que pertenecen a la misma región. Para el caso de calcular perímetros, cada lado de un rectángulo de la matriz se suma solo si el rectángulo vecino correspondiente a ese lado no pertenece a la misma región.

Una vez que una isla ya fue considerada como drain o source de un transistor, el programa la marca para no volver a considerarla nuevamente. Esto es necesario porque se da el caso de transistores que comparten un terminal (por ejemplo el drain). El área, perímetro y número de cuadrados de esa isla debe ser considerado una sola vez en el circuito final.

Para encontrar el largo y el ancho (L y W) del canal en un transistor MOS, se ha implementado un conjunto de reglas *if—then—else* que analizan en forma eurística los elementos de la matriz que representan el canal.

El número de cuadrados de drain que se toma es el correspondiente a un rectángulo que tiene el mismo perímetro y la misma área del drain, como ancho del rectángulo se toma el lado que difiera menos del ancho del canal del transistor. Para el source se realiza un procedimiento similar.

Referencias

[1] **Carlos E. Christoffersen**: *Programa Extractor de Circuitos MICRO Version 0.5*, Publicación Interna, Laboratorio de Microelectrónica de la Universidad Nacional de Rosario.

[2] **Bill Lin and Richard Newton**: *A Circuit Disassembly Technique for Synthesizing Symbolic Layouts from Mask Descriptions*, IEEE Transactions on Computer-Aided Design, VOL. 9 NO. 9, September 1990.

[3] **Marcos Augusto Stemmer**: *Extrhibo—Um Extrator Hierárquico de Circuitos*, V Seminário Interno de Microeletrónica, Anais, Grupo de Microeletrónica, Universidade Federal do Rio Grande do Sul, 1989.

[4] **Gilberto Marchioro, Luigi Carro**: *Implementação de um Editor Simbólico Para Circuitos Integrados*, V Seminário Interno de Microeletrónica, Anais, Grupo de Microeletrónica, Universidade Federal do Rio Grande do Sul, 1989.

[5] **Carver Mead, Lynn Conway**: *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, 1980.

[6] **Geoffrey D. Ullman**: *Computational Aspects of VLSI*, Stanford University, Computer Science Press, Inc., 1984.

[7] **Neil Weste, Kamran Eshraghian**: *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1986.

[8] **Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein**: *Data Structures Using C*, Prentice-Hall International Editions, 1990.

[9] **Tanner Research, Inc.**: *Layout Editor Manual. PC version 5.00*, 1993.

[10] **Tanner Research, Inc.**: *Layout Extractor Manual. PC version 1.00*, 1992.