CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

GENERADOR DE HOJAS DE ESQUEMATICOS

GENES

Tesis que presenta el Ingeniero **Lenin Guillermo Lemus Zuñiga** para obtener el grado de **Maestro en Ciencias**, en la especialidad de **Ingeniería Eléctrica**, con opción en **Computación**.

El asesor de esta tesis fue:

Dr. Manuel E. Guzmán Rentería

México D.F., a 6 de Diciembre de 1990.

INTRODUCCION:

La invención del transistor en el año de 1947 marcó una nueva era para la electrónica, esto se vino a consolidar en los años 60 con la aparición de los primeros circuitos integrados de baja escala de integración (SSI) que contenian decenas de transistores, pero que hacian posible la compactación de los instrumentos diseñados con dichos Circuitos Integrados.

En la década de los setentas, se dislumbró un panorama diferente con el desarrollo de Circuitos Integrados de mediana (MSI) y alta escala de integración (LSI) y a principios de los ochentas una gran parte de instrumentos empleaban circuitos integrados de muy alta escala de integración (VLSI).

Para poder realizar los circuitos integrados de muy alta escala de integración, se vio la necesidad de contar con herramientas de diseño que facilitarán su diseño.

A finales de la década de los ochentas y principios de los noventas, aparecieron en el mercado sistemas automáticos de diseño, en los cuales se acepta como entrada una definición de la lógica de un circuito y se obtiene como salida la mascarilla para producir el C.I..

Los primeros chips VLSI fabricados eran de propósito general (microprocesadores, memorias, convertidores A/D, etc.), que hasta la fecha han tenido una gran aceptación entre los diseñadores de sistemas.

Sin embargo, no fue hasta mediados de los ochentas que las compañias dedicadas a la fabricación de C.I. (fundidoras) empezaron a lanzar productos para que los diseñadores pudieran realizar sus diseños en un chip, con esto se logra mayor confiabilidad y reducción de costos de producción.

Debido a lo anterior, surge el concepto de Circuito Integrado de Aplicación Especifica (ASIC).

Para desarrollar los ASIC's fue necesario que las metodologias de Diseño de C.I. y las herramientas de diseño evolucionaran y que se desarrollaran herramientas de software que facilitarán el diseño de los ASIC's.

Con el desarrollo de las herramientas de diseño surgen los conceptos:

- Diseño asistido por computadora (CAD).
- Ingeniería Asistida por computadora (CAE).

En el presente trabajo, se desarrolla una herramienta de diseño que ayuda en el desarrollo del diseño de circuitos integrados de propósito específico (ASIC's).

La estructura de la tesis es la siguiente:

En el Primer Capítulo se presenta una metodología de diseño de ASIC's, con el fin de ubicar en donde encaja el trabajo que se presenta y su contribución al desarrollo de ASIC's.

En el Segundo Capítulo se da la especificación de la herramienta de diseño y se presentan los pasos seguidos para implementarla.

En el Capítulo Tres se presentan como fue desarrollado cada uno de los módulos que constituyen a la herramienta.

En el Apéndice A se presentan los conceptos de captura esquemática, empleados en el desarrollo de este trabajo.

En el Apéndice B se presenta el concepto de lista de interconexiones (NETLIST) y de los formatos empleados para transferir la información grafica de un diseño, asi como también se presenta el formato de transferencia y el netlist que serán empleados en este trabajo.

En el Apéndice C se presentan los conceptos básicos de los algoritmos de enrutado automático más comunmente empleados.

Debido a que la cantidad de programas desarrollados es extensa, y con el objeto de facilitar su distribución el código fuente de los programas desarrollados se almacenaron en un diskette de 3.5 pulgadas en el formato del Sistema Operativo DOS de IBM.

La organización del diskette se explica en el apéndice D.

Noviembre, 1990.

AGRADECIMIENTOS

Agradezco al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional por el apoyo recibido para realizar esta TESIS.

A los Jefes de Sección del Departamento de Ingeniería Eléctrica, por las facilidades recibidas para usar el equipo de sus respectivas secciones.

Dr. Hildeberto Jardón.
Dr. René Asomoza.
Dr. Guillermo Morales.

Al Jefe del Departamento de Ingeniería Eléctrica, por su apoyo para la realización de este Trabajo:

Dr. David Muñoz.

A los Gerentes del Centro de Tecnología de Semiconductores.

A mis profesores y compañeros de trabajo y de clase por el apoyo y la ayuda prestada.

A mis amigos del CTS y del CINVESTAV por su apoyo moral.

A Mr. Joe Hutt por su tiempo, comentarios y ayuda prestada para llevar a cabo gran parte de esta tesis.

Un sincero agradecimiento a mis asesores de Tesis, por la paciencia tomada para dirigir y revisar esta tesis:

Dr. Manuel Guzmán Reinteria.
Dr. Arturo Veloz Guerrero.

Un agradecimiento especial a mis amigos Jesús Palomino Echartea y Virginia Chávez por su apoyo, ayuda incondicional y confianza para llevar a cabo este trabajo.

Finalmente, quisiera agradecer al Dr. Manuel Guzmán por su amistad y confianza depositada en mí.

DEDICATORIA

Esta tesis se la dedico con todo cariño a mis padres, como un pequeño tributo al inmenso cariño, ayuda y comprensión que me han otorgado.

A mis hermanos:

Claudia.
Juan Carlos.
Lilia.

Por la confianza y apoyo que me han dado.

Noviembre, 1990.

INDICE

Capítulo 1: METODOLOGIA DE DISEÑO

Una metodología de diseño sirve para guiar el flujo de diseño de un circuito integrado (C.I.) con el fin de optimizar el tiempo de diseño, para ello la metodología debe determinar que herramientas deben ser usadas, seguir los estandares de diseño de sistemas y hacer que el C.I. cumpla con las reglas de diseño impuestas por la tecnología que será usada para fabricarlo.

En el caso concreto de diseño de Circuitos Integrados de Aplicación Especifica (ASIC), en el Centro de Tecnología de Semiconductores (CTS) del Centro de Investigación y de Estudios Avanzados (CINVESTAV) del Instituto Politécnico Nacional (IPN) se desarrolló la siguiente metodología de diseño de ASIC's:

1.1 Metodología de Diseño de ASIC's

El primer paso para diseñar un ASIC consiste en definir las especificaciones con las cuales debe cumplir, en el ambiente de diseño en el que operará.

Dentro de estas especificaciones, se debe determinar la frecuencia máxima de operación del ASIC, ya que ésta puede ser un factor determinante en la elección de la tecnología de fabricación que deba ser empleada para fabricar el chip, además se debe definir perfectamente la función que desempeñará el ASIC.

Una vez determinada la función del ASIC, se propone una Arquitectura y se procede a realizar un Modelo Funcional que refleje la arquitectura propuesta. A este modelo se le denominará Modelo Lógico I.

Al proponer la arquitectura, es altamente recomendable el que se trate de tener una arquitectura con vistas a obtener coberturas de falla apropiadas, para ello se pueden usar metodologías del tipo DFT (Design For Testability), SCAN PATH, etc..

El modelo Funcional es un programa de computadora escrito en un lenguaje de programación de alto nivel (C, Pascal, Fortran).

El modelo lógico I es un diagrama esquemático, realizado a base de compuertas y bloques.

Una vez que se tiene el modelo funcional se procede a generar datos de entrada (vectores de prueba) que verifiquen que cumple con las especificaciones.

También es factible realizar pruebas que determinen si es posible llevar a cabo cambios en la arquitectura propuesta, determinar el tipo de empaquetado en el cuál puede ser llevado a cabo, tomando en cuenta la disipación de potencia que requiere el ASIC y hacer una estimación del número de compuertas necesarias para su realización.

Si se llevan a cabo cambios en la arquitectura, se debe verificar que después de realizar los cambios, los vectores de prueba sigan dando los mismos resultados.

Una vez que se tiene el modelo funcional operando satisfactoriamente, se procede a realizar el modelo funcional con componentes ideales, esto significa que los componentes básicos que conforman al modelo lógico tienen tiempos de subida, bajada y propagación de 0 seg.

Es factible realizar algunas partes del modelo lógico en paralelo al desarrollo del modelo funcional, es recomendable hacer esta tarea cuando se tiene certeza de que la parte que se está desarrollando es poco probable que cambie su arquitectura o que los cambios que sea necesario realizar, sean fáciles de llevar a cabo.

Una vez que se cuenta con el modelo lógico I, se procede a realizar su verificación, para dicho fin se emplean los vectores de prueba desarrollados para verificar al modelo funcional y si los resultados proporcionados por el modelo lógico son satisfactorios, se procede a realizar un nuevo modelo lógico pero esta vez se emplean componentes reales, lo cual significa que tienen tiempos de propagación, de subida, de bajada, restricciones de frecuencia máxima, tiempo de inicialización (set up time) y tiempos de retención (hold time) proporcionados por los fabricantes de C.I., a este nuevo modelo se le denominará modelo lógico II.

Las bibliotecas que contienen la información de los componentes reales son proporcionados por las compañías que se dedican a fabricar ASIC's.

El siguiente paso consiste en verificar la temporización del modelo lógico II, si existen problemas de temporización es probable que sea necesario rediseñar la arquitectura del chip, lo cual implica volver a realizar los pasos descritos anteriormente.

Si el modelo lógico II no tiene problemas de temporización, el siguiente paso consiste en verificar la cobertura de fallas del circuito.

Si su cobertura de fallas no es el apropiado, será necesario rediseñar la arquitectura e incluso agregar lógica extra para incrementar la cobertura de fallas.

Para obtener la cobertura de fallas no es necesario tener el modelo lógico II, esta puede ser llevada a cabo con el modelo lógico con componentes ideales.

Finalmente una vez que la cobertura de fallas del circuito es la apropiada, y que el modelo lógico II no tiene problemas de temporización, el siguiente paso consiste en enviar a fabricación el ASIC.

Para ello se debe enviar el esquemático del ASIC o en su defecto la lista de interconexiones (NETLIST) del ASIC con la restricción de que todos los componentes básicos que conforman al ASIC procedan de la biblioteca de funciones de la casa fundidora que lo va a fabricar, asi como también se les deben enviar los vectores de prueba empleados para determinar la cobertura de fallas.

Al mandar a fabricar el ASIC, puede llegar a suceder que el costo de fabricación del ASIC varie entre diferentes fundidoras, o que el tiempo de entrega de alguna fundidora particular sea mejor que el de las otras.

En dicho caso, se debe crear un modelo lógico II con los componentes básicos procedentes de la compañia fundidora elegida.

Para crear el modelo lógico II se debe llevar a cabo un proceso de captura esquemática, verificar que no haya

problemas de funcionamiento lógico ni de temporización y traducir los vectores de pruebas al formato establecido por la fundidora.

De acuerdo a la descripción anterior, el flujo de diseño de un ASIC es:

a) Especificar el ASIC.

b) Desarrollo de la arquitectura, empleando técnicas de diseño de ASIC's para obtener coberturas de fallas altas (mayores al 95%).

c) Obtener el Modelo Funcional.

d) Verificar que el modelo funcional cumpla con las especificaciones. Para realizar esta verificación se deben desarrollar vectores de prueba.

   En caso de que no se cumpla con las especificaciones modificar el modelo Funcional, esto es regresar al inciso c.

e) Realizar el modelo lógico I con componentes ideales.

f) Verificación Lógica del Modelo Lógico I, empleando los vectores de prueba desarrollados en el inciso c.

   En caso de que no se cumplan las especificaciones regresar al inciso e y realizar los cambios necesarios.

g) Realizar el modelo lógico II con componentes reales.

h) Verificación Lógica del modelo lógico II, empleando los vectores de prueba desarrollados en el inciso c.

   Si no se cumplen las especificaciones regresar al inciso g.

i) Verificación de la temporización del modelo lógico II.

j) Pruebabilidad del modelo lógico I ó II.

Para llevar a cabo este punto se deben generar vectores de prueba, y llevar a cabo simulaciones de fallas, empleando dichos vectores.

Los vectores de prueba pueden ser desarrollados manualmente o empleando generadores automáticos de patrones de prueba (ATPG).

En caso de que la cobertura de fallas no sea la especificada, modificar la arquitectura, esto implica regresar al inciso b.

k) Enviar a la fundidora el esquemático del ASIC, o un NETLIST del mismo, asi como los vectores de prueba.

Puede ser necesario que antes de enviar los esquemáticos, sean ejecutados programas, proporcionados por la compañia fundidora, para verificar que el esquemático y los vectores de prueba están de acuerdo a sus formatos.

1.2 Definición del tema de Tesis

Si se toma en cuenta que la captura esquemática de un ASIC de 4 mil compuertas puede llegar a tomar hasta 2 meses, si trabajan 3 Ingenieros, sería provechoso contar con una herramienta de diseño que se encargue de generar el modelo lógico II a partir del modelo lógico I y la biblioteca de componentes de la casa fundidora que fabrica los ASIC's.

Suponiendo que con el uso de esta herramienta tome del orden de dos semanas el llevar a cabo la traducción el tiempo de diseño se reduce muy favorablemente para los diseñadores.

Con las siguientes ventajas adicionales:

a) Poder cambiar de tecnología de proceso.

Por ejemplo:
En vez de usar GATE ARRAYS poder emplear Standard CELLS,

b) Emplear bibliotecas con componentes que sean más rápidos.

c) Tener la alternativa de fabricar el ASIC con una compañía fundidora alternativa, en caso de obtener mejores precios de fabricación o tiempos de entrega más cortos.

d) Poder realizar la verificación lógica y la simulación de fallas independientemente de la tecnología que se va a emplear en la fabricación del ASIC, permitiendo al diseñador trabajar con una biblioteca genérica.

En el presente trabajo se desarrolla dicho generador de esquemáticos con la restricción de que los diseños que se traduzcan serán realizados hoja por hoja de esquemáticos y la generación será llevada a cabo únicamente para la fundidora Texas Instruments (TI).

Capítulo 2: ESPECIFICACION DEL GENERADOR DE HOJAS DE ESQUEMATICOS

El objetivo que se desea lograr con este trabajo es contar con una herramienta de software que permita generar una hoja de esquemático a partir de una hoja de esquemático ya existente, con el fin de que los componentes de la nueva hoja de esquemáticos este compuesta por componentes de una fundidora particular.

Además, los bloques que configuren al generador de esquemáticos deben ser reutilizables, esto es, deben poder ser usados de manera independiente, con el objeto de generar nuevas herramientas de diseño.

Para llevar a cabo la generación de la nueva hoja, se propone el siguiente flujo de datos:

a) Obtener información acerca del nombre, colocación, propiedades e interconexión de los componentes de la hoja de esquemáticos original.

   Esta información será obtenida mediante el uso de un archivo que sea ejecutado (Script) por el editor de redes (NETED) y que proporcione como salida un archivo de datos con un formato preestablecido.

   A este script (archivo ejecutable por NETED) se le denominará NETLISTER.

   El archivo de datos obtenido se denominará FORNET (FORmat NETlist).

b) Intercambiar los componentes primitivos del archivo FORNET por los componentes de la fundidora TEXAS INSTRUMENTS.

   Para llevar a cabo este intercambio se empleará un analizador lexicográfico, un analizador sintáctico y un archivo que indique como sustituir los componentes.

   A este módulo le denominará CAMBIADOR y al archivo que genere se le llamará FORNET_UNO.

c) Debido a que los componentes de Texas Instruments (TI) tienen tamaños diferentes a los componentes primitivos de Mentor Graphics (MG), será necesario cambiar la información de colocación de los componentes del archivo FORNET_UNO, para llevar a cabo esta tarea se empleará un algoritmo de descompactación cuya función será la de reubicar a los componentes con el fin de evitar que se traslapen.

   Este módulo, que se encarga de reubicar a los componentes, se denominará DESCOMPACTADOR y como salida proporcionará un archivo denominado LAYOUT, con las nuevas posiciones de los componentes.

d) A partir de la información del archivo FORNET_UNO se deberá obtener la información para enrutar a los componentes que han sido reubicados, para ello se empleará un algoritmo de enrutado.

   A este módulo se le llamará ENRUTADOR.

   Como salida deberá proporcionar un archivo con las trayectorias de enrutado.

e) Finalmente se deberá generar un script (archivo ejecutable por NETED) que al ser ejecutado por el editor de redes de MG genere la nueva hoja de esquemático.

   El nombre que recibirá este módulo será GENSCR.

Además de los módulos anteriores se desarrollará un módulo que se encargue de invocar a los módulos anteriores, de tal forma que el usuario proporcione la mínima información necesaria para llevar a cabo la generación de hojas de esquemático.

A este módulo se le denominará GENES.

A continuación se describirá el flujo de datos de genes y posteriormente se presentarán las especificaciones de cada uno de estos módulos.

2.1 Flujo de Datos de Genes

Con el fin de presentar de manera más clara el flujo de datos de GENES, a continuación se presenta como se efectúa dicho flujo de datos:

FLUJO DE DATOS DE GENES

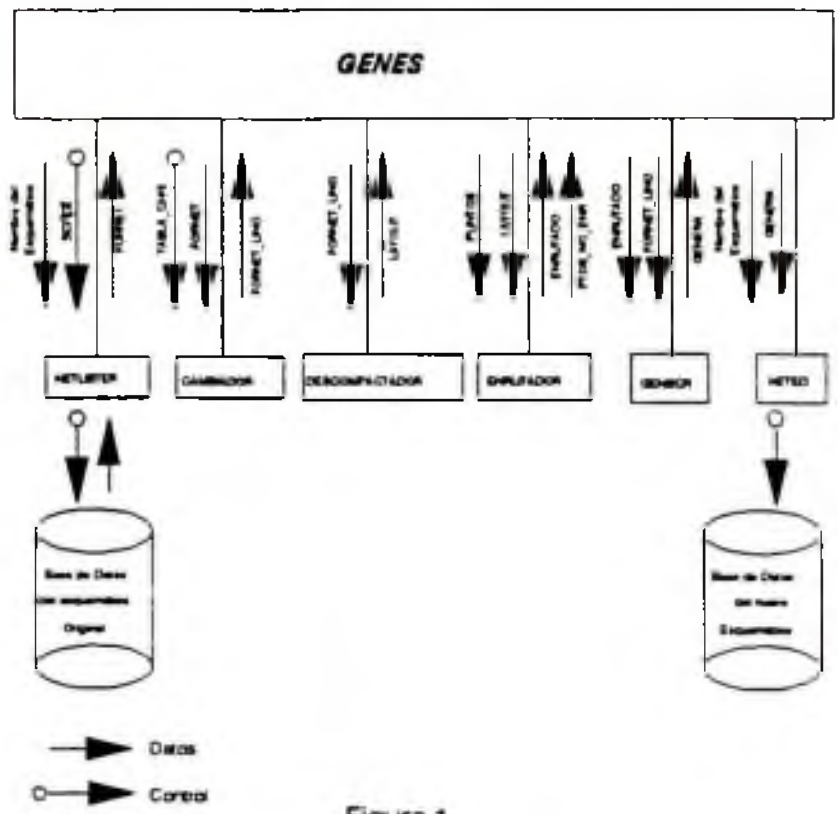


Figura 1.

De acuerdo al flujo de datos presentado en el Capitulo 2 y a la figura 1., se observa que hay una incompatibilidad entre los datos que espera el enrutador y los datos que se le envian.

Esto es, el módulo enrutador acepta como datos de entrada un archivo con los puntos a ser enrutados y lo que se le envía son los archivo FORNET_UNO.

Para que el enrutador opere adecuadamente se pueden seguir las siguientes alternativas:

i) Agregar un analizador lexicográfico al módulo enrutador, de manera que este analizador lea la lista de interconexciones FORNET_UNO y la información de la nueva posición de cada bloque, y a partir de esta información genere la lista de puntos a ser enrutados, y las áreas restringidas.

ii) La segunda opción es no incorporar el analizador lexicográfico al enrutador, sino agregar un módulo que se encargue de leer la lista de interconexciones FORNET_UNO y la información de la nueva posición de cada bloque (LAYOUT_UNO), a partir de esta información genere la lista de puntos a ser enrutados, las áreas restringidas y la lista de interconexiones FORNET_DOS.

Dado que uno de los objetivos de la tesis es lograr que los módulos sean independientes unos de otros, se optará por elegir la alternativa de generar un nuevo bloque cuya función sea la de generar los datos que espera el módulo enrutador.

El nuevo módulo se denominará INTERFAZ, como entrada recibirá a los archivos con la topología y la lista de interconexiones, y como salida proporcionará archivos con información acerca de los puntos a enrutar, la topología y el netlist.

Finalmente, el flujo de datos de GENES se llevará a cabo de acuerdo al siguiente diagrama de flujo de datos:

FLUJO DE DATOS DE GENES

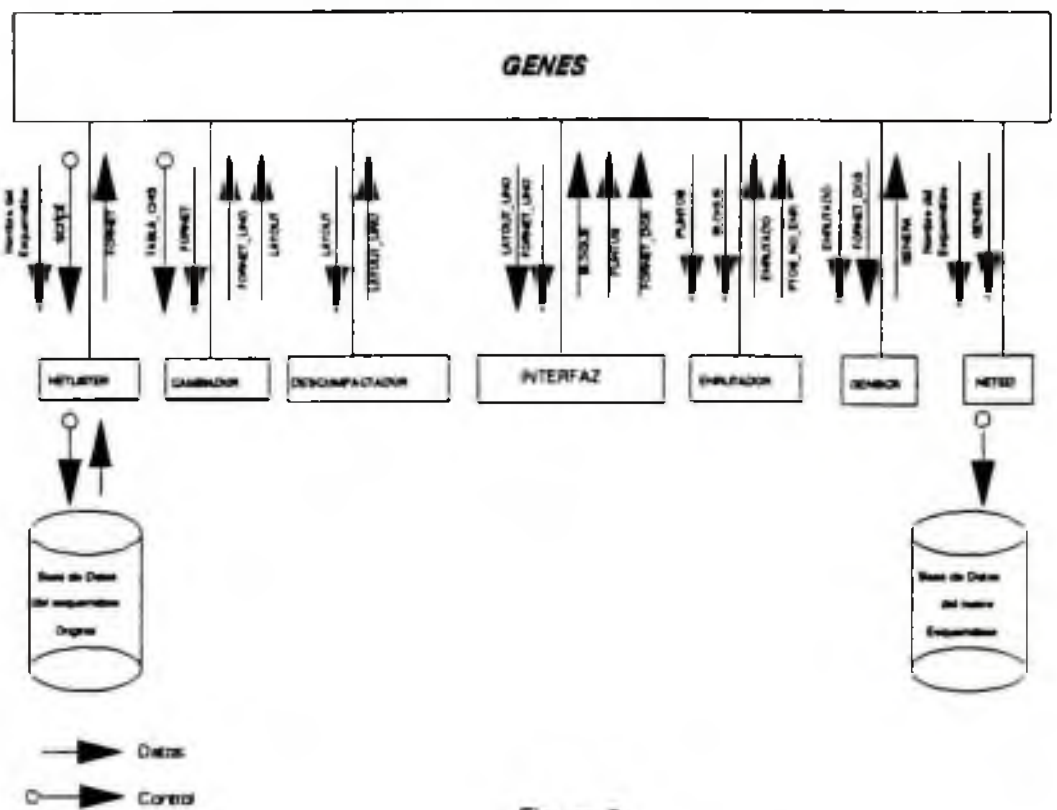


Figura 2.

2.2 Especificaciones del módulo NETLISTER

Datos de entrada:

- Nombre del esquemático que indica a NETED, cuál es Base de Datos de la hoja de esquemáticos con componentes básicos proporcionados por Mentor Graphics.
- Archivo ejecutable (Script) por el editor de Redes (NETED).

Datos de Salida:

- Archivo con un formato definido (FORNET), que presente la información de colocación, propiedadades y conectividad de los bloques que conforman a la hoja de esquemáticos.

Restricciones:

El archivo de salida FORNET debe tener una estructura válida de acuerdo a la gramática que sea definida para presentar la información de colocación y conectividad de la hoja de esquemáticos proporcionada.

2.3 Especificaciones del módulo CAMBIADOR

Datos de entrada:

- Archivo FORNET.
- Tabla con información para llevar a cabo la sustitución de componentes.

Datos de Salida:

- Archivo de datos FORNET_UNO, el cual contiene la lista de interconexiones (NETLIST) con elementos de Texas Instruments.

- Archivo LAYOUT con información acerca de los nombres, tamaño colocación y separación mínima de los componentes que conforman a la hoja de esquemáticos.

2.4 Especificaciones del módulo DESCOMPACTADOR

El algoritmo de descompactación debe separar a los bloques con el fin de evitar traslapamientos, sin tener restricciones en el tamaño de la hoja que sea necesario utilizar.

Datos de entrada:

- Archivo LAYOUT con información acerca de los nombres, tamaño colocación y separación mínima de los componentes que conforman a la hoja de esquemáticos.

Datos de salida:

- Archivo LAYOUT_UNO, con la nueva colocación de los componentes.

2.5 Especificaciones del módulo INTERFAZ

La función de este módulo será la de leer una lista de interconexiones y un archivo con la nueva colocación de los componentes de la lista de interconexiones y genera la información requerida por el módulo enrutador.

Datos de entrada:

- Archivo FORNET_UNO.
- Archivo LAYOUT_UNO

Datos de salida:

- Archivo con la lista de puntos a ser enrutados, denominado PUNTOS.

- Archivo BLOQUES, con la información acerca de las áreas que no deben ser cruzadas.

- Archivo FORNET_DOS.

2.6 Especificaciones del módulo ENRUTADOR

El enrutador debe ser capaz de enrutar parejas de puntos sin cruzar por áreas restringidas y sin provocar la sobreposición de segmentos de un alambrado sobre otro alambrado, se permitirá el cruce de segmentos de alambrados diferentes, siempre y cuando el punto de cruce no sea un vértice de ninguno de los alambrados en cuestión.

Datos de entrada del enrutador:

- Archivo con los puntos a enrutar, denominado PUNTOS.

- Archivo BLOQUE con información acerca de las áreas que no deben ser cruzadas (representadas por segmentos de línea horizontales y verticales).

Como salida el enrutador deber proporcionar:

- Archivo, llamado ENRUTADO, con las coordenadas de los puntos que constituyen los segmentos que representan a las nets.

- Archivo con las coordenadas de los puntos que no fue posible alambrar, PTOS_NO_ENR.

2.7 Especificaciones del módulo GENSCR

Datos de entrada:

- Archivo FORNET_DOS
- Archivo ENRUTADO

Datos de salida:

- Archivo ejecutable por el editor de redes (GENERA), con la información para generar la nueva hoja de esquemáticos.

2.8 Especificaciones del Módulo GENES

La función de este módulo será la de invocar a los módulos NETLISTER, CAMBIADOR, DESCOMPACTADOR, INTERFAZ, ENRUTADOR y GENSCR, con el fin de hacer la tarea de generación lo más sencilla posible.

Datos de entrada:

- Nombre del esquemático que indica a NETED cuál es la Base de datos de la hoja de esquemáticos que se desea traducir.

Datos de salida:

- Archivo ejecutable por el editor de redes, que contenga la información necesaria para generar la nueva hoja de esquemáticos con componentes de Texas Instruments.

Capítulo 3: DISEÑO Y CONSTRUCCION DEL GENERADOR DE HOJAS DE ESQUEMATICOS

De acuerdo al plan de trabajo presentado en el capítulo 2, en este capítulo se describirá como fue diseñado y construido cada uno de los bloques que conforman al generador de hojas de esquemáticos.

3.1 Módulo NETLISTER

Especificaciones del bloque NETLISTER.

Datos de entrada:

- Nombre del esquemático que indica a NETED cuál es la Base de Datos de la hoja de esquemáticos con componentes básicos proporcionados por Mentor Graphics.
- Archivo ejecutable (Script) por el editor de Redes (NETED).

Datos de Salida:

- Archivo con un formato definido (FORNET), que presente la información de colocación, propiedades y conectividad de los bloques que conforman a la hoja de esquemáticos.

Restricciones:

El archivo de salida FORNET debe tener una estructura válida de acuerdo a la gramática que sea definida para presentar la información de colocación y conectividad de la hoja de esquemáticos proporcionada.

3.1.1 Estrategia de diseño.

Para poder diseñar y construir el bloque encargado de obtener la lista de interconexiones de una hoja de esquemáticos (NETLISTER), se proponen las siguientes actividades:

a) Definir la gramática con la cual debe cumplir el archivo que contiene a la lista de interconexiones.

b) Diseñar el archivo ejecutable por NETED (Script) encargado de obtener la lista de interconexiones en un formato tal que cumpla con la gramática definida en el inciso anterior.

c) Diseñar y construir un analizador lexicográfico y un analizador sintáctico que verifique que la lista de interconexiones obtenida realmente cumple con la gramática propuesta.

3.1.2 Gramática de la lista de interconexiones

Para definir la gramática, se tomaron en cuenta los siguientes factores:

Los datos obtenidos de la hoja de esquemáticos deben proporcionar suficiente información acerca de las características de la misma, de cada uno de los componentes, e información referente a la conectividad de los componentes que conforman al esquemático.

La información obtenida de la hoja de esquemático es referente al tamaño de la hoja, el espaciado de la malla de enrutado y las unidades empleadas (cm. o pulgadas).

En lo concerniente a la información obtenida de cada uno de los componentes, el script debe proporcionar la siguiente información de cada uno de ellos:

- Trayectoria.
- Handle.
- Parámetros.
- Ubicación.
- Tamaño.
- Rotacion y espejo
- Centro de rotación
- Propiedades.

Para definir a una propiedad es necesario definir su nombre y su valor, sin embargo para poder mostrar una propiedad en la hoja de esquemáticos, se requiere la siguiente información:

- Sí la propiedad es visible o no.
- La colocación, rotación y justificación del texto que representa al valor de la propiedad.

  La colocación de la propiedad se debe dar como un desplazamiento del punto origen del elemento al cual se encuentra ligada la propiedad.

  Además, para evitar que el valor de una propiedad sea cambiado de manera accidental, se debe determinar si el valor de la propiedad puede ser cambiado o no.

Finalmente, en lo referente a la información de conectividad, se propone presentar la información en forma de conectividad orientada a net, por lo que la información que el script obtiene es:

- Nombre de la net.
- Nombre Handle y posición de los pines que interconecta la net.
- Propiedades de la net.

Tomando en cuenta lo anterior, a continuación se presenta la gramática propuesta en la forma de Backus Nauer.

Símbolos terminales de la gramática:

PAGINA
ESPACIADO
UNIDADES
COMPONENTE

HANDLE
NET
INS_HANDLE
NET_HANDLE
CENTRO_ROT
TAMANO
ORIGEN
DOS_PTS
COMA
FLOTANTE
ENTERO
CADENA
HANDLE
PATH
ROTACION
ESPEJO
PIN
PROPIEDAD
VALOR
FIJA
VISIBLE
JUSTIFICACION
ORIGEN
OFFSET
TRUE
FALSE
TAMANO_PROP
PARAMETRO
COMILLA
IDVALUE

La gramática propuesta es:

<netlist> ::= <hoja> <comp> <conexion>

<hoja> ::= <pagina> <espacio> <unidades>

<pagina> ::= PAGINA : FLOTANTE , FLOTANTE

<espacio> ::= ESPACIADO : FLOTANTE

<unidades> ::= UNIDADES : inches
| UNIDADES : cm

GENES

```
      <propiedad>   ::=    <nombre>   <valor>   <datos_prop>
<propiedad>
                       | EPSILON
     <nombre> ::= PROPIEDAD : <id>

     <valor> ::= VALOR : <val_prop>

      <datos_prop>    ::=    <fija>    <visible>    <rotación>
<tamaño>
                          <justificación> <origen> <offset>

     <fija> ::=     FIJA : TRUE
                  | FIJA : FALSE

     <visible> ::=    VISIBLE : TRUE
                      VISIBLE : FALSE

     <rotación> ::= ROTACION : <grados>

     <grados> ::=    0
                   | 90
                   | 180
                   | 270

     <tamaño> ::=  TAMANO : FLOTANTE

     <justificación> ::=  JUSTIFICACION :  <dir> <dir>

     <dir> ::=    C
                | L
                | R
                | U
                | D

     <origen> ::=  ORIGEN : FLOTANTE , FLOTANTE


      <componente>     ::=     <datos_compo>     <propiedades>
<pines>
                            <componente>
                          | NULL

       <datos_compo>    ::=     <nombre_comp>    <handle_comp>
<centro_rot>
                              <origen>    <tamaño>    <rotación>
<espejo>
              |   <nombre_comp>   <parametro>   <handle_comp>
<centro_rot>
```

```
                              <origen>    <tamaño>    <rotación>
<espejo>
            | <nombre_comp> <handle_comp> <centro_rot>
                 <origen>  <tamaño>  <incremento>  <rotación>
<espejo>

            |    <nombre_comp>    <parametro>    <handle_comp>
<centro_rot>
                 <origen>  <tamaño>  <incremento>  <rotación>
<espejo>


     <nombre_comp> ::= COMPONENTE : <path>

      <parametro>     ::=   PARAMETRO   <nombre_parametro>   :
<valor_parametro>

     <nombre_parametro> ::=  CADENA

     <valor_parametro>  ::= CADENA

     <incremento>  ::=  INCREMENTO : FLOTANTE , FLOTANTE

     <handle_comp> ::=  INST_HANDLE

     <centro_rot> ::=  CENTRO ROTACION : FLOTANTE , FLOTANTE

     <espejo> ::=   ESPEJO : TRUE
                  | ESPEJO : FALSE

    <conexión> ::= NET NET_HANDLE  <net>

      <net>  ::=  COMPONENTE  :  PATH  [  INS_HANDLE  ]  PIN  :
NET_HANDLE net_name net
        | EPSILON

     <net_name> ::= [ CADENA ]
            | [ ENTERO ]
            | [ ]
```

3.1.3 Pseudocódigo del Script

Para que el script genere la información que se desea, se propone el siguiente algoritmo:

```
Begin

Abrir el archivo de Datos FORNET

    Obtener tamaño de la hoja
    Obtener tipo de unidades
    Obtener espaciado de la malla

    Escribir al archivo NETFOR (tamaño de la hoja,
    espaciado de la malla y tipo de unidades)

    Guardar en una lista los nombres de todas las
    instancias de la hoja de esquemáticos.


    Para cada elemento de la lista de instancias obtener:
      Begin

      Trayectoria
      handle,
      Centro de rotación,
      Origen,
      Tamaño,
      Rotación,
      Espejo.


      escribir al archivo FORNET (Trayectoria, handle,
      centro de rotación, origen, tamaño, rotación y
      espejo de la instancia analizada)

      Obtener una lista con los nombres de todas las
      propiedades que contenga la instancia.

      Para cada propiedad de la lista de propiedades
      obtener:
        Begin
```

Nombre de la propiedad,
Valor de la propiedad,
Sí la propiedad es fija o no,
Sí es visible o no,
Rotación,
Tamaño,
Justificación,
Origen,
Desplazamiento respecto al origen.

escribir al archivo FORNET (Nombre de la propiedad, valor de la propiedad, sí la prop. es fija o no, si es visible, la rotación tamaño, justificación, origen, y desplazamiento del texto que identifica al valor de la propiedad)

End;

Obtener una lista con los nombres de los pines de la instancia.
Para cada pin de la lista de pines la siguiente información:
Begin

Nombre del pin
Sí el nombre del pin es una propiedad fija o no,
Sí el nombre del pin es visible o no,
Rotación del nombre del pin,
Tamaño del nombre del pin,
Justificación,
Origen del pin,
Desplazamiento del nombre del pin respecto al origen del
mismo.

Obtener una lista con los nombres de todas las propiedades ligadas al pin.

Para cada propiedad de la lista de propiedades obtener:
Begin

Nombre de la propiedad,
Valor de la propiedad,
Sí la propiedad es fija o no,
Sí es visible o no,
Rotación,
Tamaño,

```
                    Justificación,
                    Origen,
                    Desplazamiento respecto al origen.

                    escribir al archivo FORNET (Nombre de la
                    propiedad, valor de la propiedad, si la
                    prop. es fija o no, si es visible, la
                    rotación tamaño, justificación, origen, y
                    desplazamiento del texto que identifica al
                    valor de la propiedad)

               End;
               End;
          End;
        Obtener una lista con los nombres de todas las nets de
        la hoja de esquemáticos.
        Para cada elemento de la lista de nets obtener:
          Begin
            Nombre de la NET,
            Handle de la NET,
            Trayectorias de los Componentes que se conectan a la
            NET,
            Handles de los componentes que se conectan a la NET.

            escribir al archivo FORNET(nombre de la net, handle
            de la net, trayectorias y handles de los componentes
            que están unidos a la net)
          End;
    Cerrar el archivo FORNET.
End.
```

3.1.4 Analizador Lexicográfico y Sintáctico

Para realizar el analizador lexicográfico, se empleó el programa LEX, de UNIX, las regla gramaticales indicadas fueron:

```
%%
letra           [a-zA-Z]
num             [0-9]
signo           [+-]
pt              \.
coma            \,
comilla         \'
under           "_"
slash           "\/"
pesos           "\$"
space           [ \t\n]
dos_pts         ":"
param           {comilla}({letra}|{num})*{comilla}
int             {num}+
float           {signo}?{int}{pt}{int}?[eE]?{int}
net_handle      [NV]{pesos}{int}
ins_handle      [I]{pesos}{int}
handle          {letra}{pesos}{int}
path            {slash}(({id}|{cadena}){slash})*{componente}
componente      {pesos}{id}
id              {letra}({letra}|{num}|{under})*
cadena          {letra}+
%%
{id}        {
            if (strcmp("PAGINA",yytext) == 0 )
                return  PAGINA;
            else if (strcmp("ESPACIADO",yytext) == 0 )
                return  ESPACIADO;
            else if (strcmp("UNIDADES",yytext) == 0 )
                return  UNIDADES;
            else if (strcmp("COMPONENTE",yytext) == 0 )
                return  COMPONENTE;
            else if (strcmp("HANDLE",yytext) == 0 )
                return  HANDLE;
            else if (strcmp ( "CENTRO_ROTACION",
                 yytext) == 0 )
                return  CENTRO_ROT;
            else if (strcmp("TAMANO",yytext) == 0 )
                return  TAMANO;
            else if (strcmp("ROTACION",yytext) == 0 )
```

```
            return  ROTACION;
        else if (strcmp("ESPEJO",yytext) == 0 )
            return  ESPEJO;
        else if (strcmp("ORIGEN",yytext) == 0 )
            return  ORIGEN;
        else if (strcmp("PIN",yytext) == 0 )
            return  PIN;
        else if (strcmp("PROPIEDAD",yytext) == 0 )
            return  PROPIEDAD;
        else if (strcmp("VALOR",yytext) == 0 )
            return  VALOR;
        else if (strcmp("FIJA",yytext) == 0 )
            return  FIJA;
        else if (strcmp("VISIBLE",yytext) == 0 )
            return  VISIBLE;
        else if (strcmp (" JUSTIFICACION",
             yytext) == 0 )
            return  JUSTIFICACION;
        else if (strcmp("OFFSET",yytext) == 0 )
            return  OFFSET;
        else if (strcmp("TRUE",yytext) == 0 )
            return  TRUE;
        else if (strcmp("FALSE",yytext) == 0 )
            return  FALSE;
        else if (strcmp("NET",yytext) == 0 )
            return  NET;
        else if (strcmp("PARAMETRO",yytext) == 0 )
            return  PARAMETRO;
        else
            return  CADENA;
    }
{net_handle}        { return  NET_HANDLE;}
{ins_handle}        { return  INS_HANDLE;}
{handle}        { return  HANDLE;}
{int}           { return  ENTERO;}
{float}         { return  FLOTANTE;}
{path}          { return  PATH;}
{componente}    { return  ;}
{dos_pts}       { return  DOS_PTS;}
{coma}          { return COMA;}
{param}         { return CADENA;}
{space}         ;
\[              { return( '[');}
\]              { return( ']');}
.               {printf("Desconocido %s\n",yytext);}
%%
```

Para realizar el analizador sintáctico se empleó el programa YACC de UNIX, con la siguiente gramática no ambigua:

```
inicio:     pagina   componente nets

pagina:     tamaño_pag espaciado unidades

tamaño_pag: PAGINA DOS_PTS  coordenadas

espaciado:  ESPACIADO DOS_PTS FLOTANTE

unidades:   UNIDADES DOS_PTS CADENA

componente:
        | path datos_componente componente
path:       COMPONENTE DOS_PTS PATH

datos_componente:   datos_a propiedades  pines

datos_a:      handle centro_rotacion origen size rotacion
espejo propiedades
        | handle centro_rotacion origen size incremento
rotacion espejo propiedades
        | parametro handle centro_rotacion origen size
rotacion espejo propiedades
        | parametro handle centro_rotacion origen size
incremento rotacion espejo propiedades

parametro:  PARAMETRO CADENA DOS_PTS CADENA

handle:     HANDLE DOS_PTS tipo_handle

tipo_handle:      HANDLE
        | NET_HANDLE
        | INS_HANDLE

centro_rotacion:    CENTRO_ROT DOS_PTS  coordenadas

origen:     ORIGEN DOS_PTS  coordenadas
```

GENES

```
size:       TAMAÑO DOS_PTS  coordenadas

incremento:     INCREMENTO : coordenadas

rotacion:   ROTACION DOS_PTS entero

espejo:     ESPEJO DOS_PTS entero

coordenadas:    flotante COMA flotante

flotante:   FLOTANTE

entero:     ENTERO

propiedades:
        | propiedad propiedades

propiedad: nombre  valor  tipo  visible   rotacion   tamaño
justificacion  origen  offset

nombre:     PROPIEDAD DOS_PTS CADENA
    | PROPIEDAD DOS_PTS PIN

valor:      VALOR DOS_PTS CADENA
    | VALOR DOS_PTS entero

    | VALOR DOS_PTS IDVALUE

tipo:       FIJA DOS_PTS  boolean

boolean:    TRUE
    | FALSE

visible:    VISIBLE DOS_PTS boolean

tamaño:     TAMAÑO DOS_PTS  flotante

justificacion: JUSTIFICACION DOS_PTS CADENA CADENA
offset:     OFFSET DOS_PTS flotante COMA flotante
```

```
pines:
    | pin pines
pin:    PIN DOS_PTS pin_a propiedades

pin_a:  CADENA
    |   CADENA bus
    |   ENTERO

bus:        ( bus_a : ENTERO)
    |   ( bus_a : CADENA )
    |   ( CADENA signo ENTERO : ENTERO )
    |   ( bus_a : CADENA signo ENTERO )

bus_a ::=   CADENA
        |   ENTERO

signo ::=   -
        |   +

nets:
    |   net nets

net:    NET NET_HANDLE  net_elem

net_elem:
        |COMPONENTE  DOS_PTS  PATH   '['  INS_HANDLE  ']'  PIN
DOS_PTS NET_HANDLE pin_name net_elem

pin_name ::=    [ CADENA ]
            |   [ CADENA BUS ]
            |   [ ENTERO ]
            |   [ ]
```

3.1.5 Formato de los datos de entrada y salida

Formato de los archivos con los datos de entrada:

SCRIPT: Archivo de datos en formato ASCII.

Formato de los archivos con los datos de salida:

FORNET: Archivo de datos en formato ASCII, que cumplen con la gramática definida en la seccion 3.1.2.

3.2 Módulo CAMBIADOR

Para el caso del bloque encargado de llevar a cabo el intercambio de componentes en la lista de interconexiones, se propuso que este bloque cumpliera con las siguientes especificaciones:

Datos de entrada:

- Archivo FORNET.
- Tabla con información para llevar a cabo sustitucion de componentes (TABLA_CHG).

Datos de Salida:

- Archivo de datos FORNET_UNO pero con los componentes básicos proporcionados por MG sustituidos por correspondientes elementos de Texas Instruments.

3.2.1 Estrategia de diseño

En el caso de este bloque, la estrategia propuesta es la siguiente:

a) Determinar la estructura de datos de la tabla de sustitución.

b) Determinar la organización de la tabla de sustitución.

c) Generar la tabla de sustitución para efectuar el cambio de los componentes de la biblioteca de gen_lib y la de Texas Instruments.

d) Diseñar y construir el programa que se encargue de generar la tabla de sustitución.

e) Diseñar y construir el programa que lleve a cabo la sustitución de los componentes del archivo FORNET.

3.2.2 Definición de la estructura de datos

Para determinar la estructura de datos tenemos que tomar en consideración los siguientes factores:

a) La tabla de intercambio debe tener la trayectoria del componente que se va a sustituir (componente original) y la trayectoria del componente que va a sustituir al componente original (componente de reemplazo).
b) El tamaño del componente original y el del componente de reemplazo.
c) Las coordenadas del centro de rotación del componente.
d) La correspondencia de pines del componente original y del componente de reemplazo.

Tomando en cuenta los factores anteriores, una estructura de datos plausible para tener la información necesaria para sustituir a un componente es:

- Trayectoria del componente Original
- Trayectoria del componente Sustituto.
- Tamaño del componente Original
- Tamaño del componente sustituto
- Centro de Rotación, expresado como un desplazamiento a partir del origen del componente.
- Lista con la correspondencia entre pines del componente original y del componente sustituto.

En donde la lista de correspondencia debe tener:

- Nombre del pin del componente original

- Nombre del pin del componente sustituto que corresponde al pin del componente original.
- Desplazamiento del pin del componente sustituto, respecto al componente al cual pertenece el pin.

Sin embargo, en la estructura de datos anterior no se toma en cuenta a los FRAMES.

En la biblioteca genérica, los FRAMES tienen como valor un parámetro.

En la biblioteca de Texas Instruments el FRAME se representa por medio de una propiedad llamada STRIP.

Tomando en cuenta lo anterior, un componente de la biblioteca original puede ser representado mediante una lista, en donde cada elemento de la lista representa un valor del parametro que representa a los FRAMES.

Y en el caso de la biblioteca de Texas Instruments, cada elemento de la lista representará a un valor particular de la propiedad STRIP.

En el caso en que un componente de la biblioteca genérica o de Texas Instruments no tenga FRAMES, se tendrá una lista con un sólo elemento.

Con el fin de generalizar la estructura que contiene a la tabla de correspondencia, se propone la siguiente estructura de datos:

- Nombre del componente Original.
- Nombre del componente de Reemplazo.
- Apuntador a la Lista de los FRAMES del componente original.

Cada elemento de la lista de los FRAMES contendrá la siguiente información:

- Nombre del Parámetro que identifica al FRAME.

- Valor del parámetro.
- Tamaño del componente para este caso del FRAME.
- Centro de rotación del componente.
- Apuntador al siguiente FRAME del componente original.
- Apuntador a la Lista con los FRAMES de reemplazo.

En el caso particular en que el elemento original NO contenga FRAMES, se indicará haciendo que el Nombre del parámetro y el Valor del mismo sean igual a NULL.

Por otro lado, se propone que cada elemento de la lista con los FRAMES de reemplazo, contenga la siguiente información.

- Nombre de la propiedad que identifica al FRAME.
- Valor de la propiedad.
- Tamaño del componente para este FRAME.
- Centro de Rotación del componente.
- Apuntador al siguiente parámetro FRAME del componente de reemplazo.
- Apuntador a la Lista con la correspondencia entre pines.

De la misma manera que en el caso anterior, si un elemento de reemplazo no contiene FRAMES, se indica haciendo que el Nombre de la propiedad y su Valor sean igual a NULL.

Finalmente, se sugiere que la lista con la correspondencia entre pines, mantenga la siguiente estructura:

- Nombre del pin del componente original.
- Posición del pin del componente original respecto al origen del componente original.
- Nombre del pin del componente de reemplazo.

- Posición del pin del componente de reemplazo respecto al origen del componente de reemplazo.
- Apuntador al siguiente elemento de la lista de pines.

A manera de resumen, la estructura de datos que se propone para almacenar la información de la tabla de cambios es:

Estructura de cada Elemento de la tabla:

Nombre del componente Original.
Nombre del componente de Reemplazo.
Apuntador a la Lista de los FRAMES.

Lista con los FRAMES del componente Original:

Nombre del parámetro que identifica al FRAME
Valor del parámetro.
Tamaño del componente para este caso.
Centro de Rotación del componente para este caso.
Apuntador al siguiente FRAME del componente original.
Apuntador Lista con los FRAMES de reemplazo.

Lista con los FRAMES del componente de reemplazo:

Nombre de la propiedad que identifica al FRAME.
Valor de la propiedad.
Tamaño del componente para este FRAME.
Apuntador al siguiente FRAME del componente de reemplazo.
Apuntador a la Lista con la correspondencia entre pines.

Lista de la correspondencia entre pines:

Nombre del pin del componente original.
Posición del pin del componente original respecto al origen del componente original.

Nombre del pin del componente de reemplazo.
Posición del pin del componente de reemplazo respecto al origen del componente de reemplazo.
Apuntador al siguiente pin.

3.2.3 Organización de la tabla de sustitución

Para determinar la organización de la tabla de sustitución, el parámetro que se consideró más importante es el hecho de que cada vez que se vaya a realizar una sustitución, será necesario realizar una búsqueda en la tabla de reemplazo.

Tomando en cuenta lo anterior, se decidió organizar la información de reemplazo como un árbol binario balanceado.

De esta manera el tiempo de búsqueda es logaritmico respecto al número de elementos que contenga la tabla.

Para generar el árbol binario, basta con agregar dos apuntadores a la estructura del inciso 4.2.2, de tal manera que se tendrá:

Estructura de cada Elemento del árbol binario balanceado:

Nombre del componente Original
Nombre del componente de Reemplazo
Apuntador a la Lista de los FRAMES
Apuntador al siguiente elemento de la tabla cuyo nombre es lexicograficamente menor al de este componente.
Apuntador al siguiente elemento de la tabla cuyo nombre es lexicográficamente mayor al de este componente.

Con el agregado de dos apuntadores, a la estructura de la sección 4.2.2 estamos en condiciones de representar a la tabla de intercambio como un árbol binario balanceado.

3.2.4 Generación de la Tabla de Intercambio de componentes

Para generar la tabla de reemplazo, se empleó la siguiente estrategia:

1) Generar un esquemático con todos los componentes de la biblioteca generica de Mentor Graphics.

   Tomando en cuenta los FRAMES.

2) Generar un esquemático con todos los componentes de la biblioteca de Texas Instruments.

   Tomando en cuenta a los FRAMES.

3) Diseño y construcción de un analizador sintáctico (parser) cuya función es la de verificar la sintaxis de las listas de interconexión de las dos bibliotecas, además de generar 2 árboles binarios balanceados en donde cada uno de ellos contiene la información de los componentes de cada biblioteca.

Para llenar la información de la tabla de intercambio es necesario almacenar de manera provisional la lista de pines de cada uno de los elementos, y su tamaño, para ello se debe modificar la estructura de datos de los elementos del árbol de la siguiente manera:

Estructura de cada Elemento de la tabla:

Nombre del componente Original.
Nombre del componente de Reemplazo.
Apuntador a la Lista de los FRAMES.

Tamaño del componente original. * Dato agregado.
Centro de Rotación del componente original, expresado como un desplazamiento a partir del origen del componente. *Dato agregado.

Lista con los FRAMES del componente Original:

Parámetro.
Valor.
Tamaño del componente para este caso.
Centro de Rotación del componente.
Apuntador al siguiente FRAME del componente original.
Apuntador Lista con los FRAMES de reemplazo.

Apuntador a la lista de pines. * Dato agregado.

Lista con los FRAMES del componente de reemplazo:

Nombre del parámetro de reemplazo.
Valor del parámetro.
Tamaño del componente para este FRAME.
Centro de Rotación del componente.
Apuntador al siguiente FRAME del componente de reemplazo.
Apuntador a la Lista con la correspondencia entre pines.

Lista de la correspondencia entre pines.

Nombre del pin del componente original.
Posición del pin del componente original respecto al origen del componente original.
Nombre del pin del componente de reemplazo.
Posición del pin del componente de reemplazo respecto al origen del componente de reemplazo.
Apuntador al siguiente pin.

4) Diseño y construcción de funciones que se encarguen de generar la tabla de intercambio, mediante el uso de los árboles binarios generados en el inciso anterior y la interacción del usuario para que determine cómo deben ser llevados a cabo la correspondencia entre pines.

3.2.5 Rotaciones y Reflexiones

Para finalizar con la tarea de intercambio de componentes, se debe tomar en cuenta las rotaciones y reflexiones que tienen los componentes originales en el esquemático original, con el fin de que los componentes que los reemplazan queden en las mismas orientaciones.

Para efectuar las reflexiones y rotaciones, se debe tomar en consideración que el símbolo de un componente ocupa un área rectangular, la cual tiene un origen, un centro de rotación y un tamaño particular.

El centro de rotación es el punto a partir del cual se efectúa una rotación o una reflexión.

Por ejemplo, si se tiene el siguiente rectángulo, en un sistema de coordenadas rectangulares:

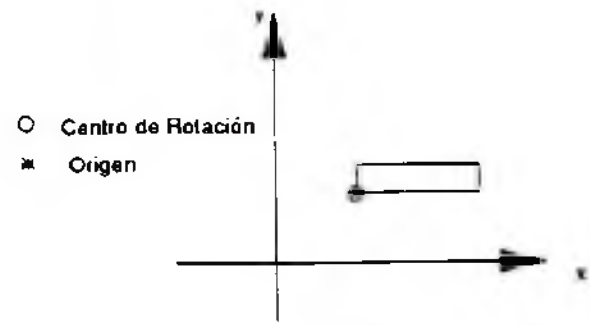


Figura 3.

Al efectuar una rotación de 90 grados en el sentido contrario al de las manecillas de un reloj, el rectángulo se verá de la siguiente manera:

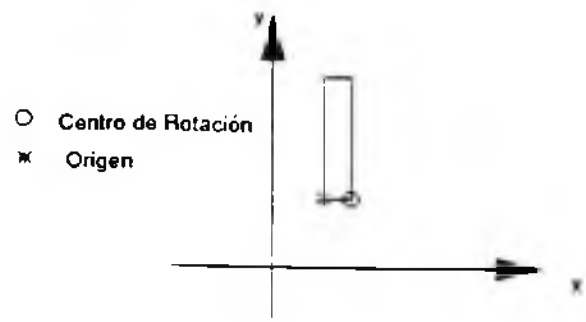


Figura 4.

Ahora bien, si cambia el centro de rotación del rectángulo en cuestión, de tal manera que se tenga:

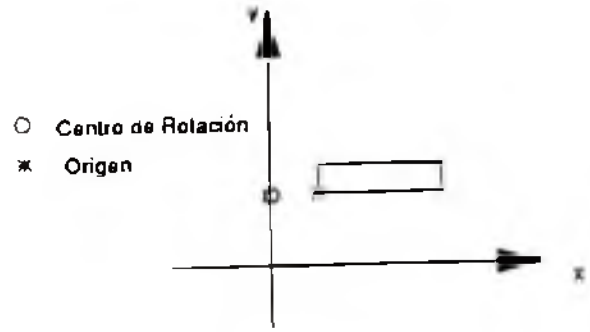


Figura 5.

Al efectuar de nuevo, la misma rotación de 90 grados en sentido contrario al de las manecillas del reloj, la figura resultante se verá de la siguiente manera:

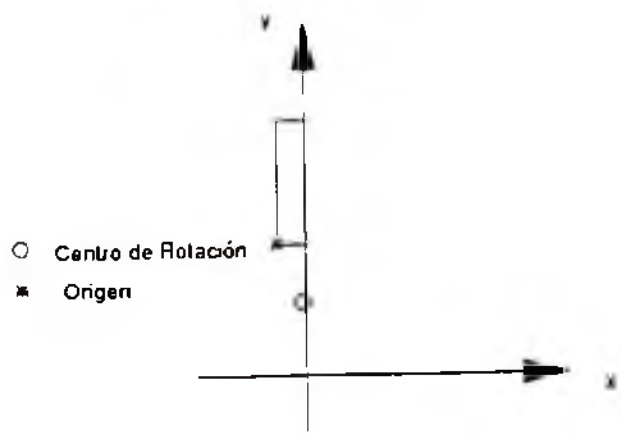


Figura 6.

En la tabla de intercambio, se ha almacenado información acerca de la posición del centro de rotación.

En base a esta información es posible colocar al componente sustituto en la misma orientación que la del componente original.

3.2.6 Intercambio de componentes

El algoritmo propuesto para llevar a cabo el intercambio es el siguiente:

i) Leer la tabla de traducción.

ii) Leer el archivo con el NETLIST de la hoja de diseño a partir de la cual se generará una nueva hoja de esquemático.

iii) Realizar los cambios de trayectoria, colocación y tamaño correspondientes, de acuerdo a la tabla de traducción.

iv) En base a la información de la orientación del componente original, proporcionada por la lista de interconexiones, evaluar cuál será la posición adecuada del componente sustituto.

La posición de los pines se dan como desplazamientos del origen del componente.

v) Escribir la nueva lista de interconexiones, en el archivo denominado FORNET_UNO.

3.2.7 Formato de los datos de entrada y salida

Formato de los datos de entrada:

FORNET: La información de este archivo, debe cumplir con la gramática definida en la sección 3.1.2.

TABLA_CHG: Esta tabla de intercambio, se encuentra contenida en un árbol binario balanceado, con la estructura de datos definida en la sección 3.4.2.

Formato de los datos de salida:

FORNET_UNO: Al igual que FORNET, este archivo debe cumplir con la gramática definida en la sección 3.1.2.

3.3 Módulo DESCOMPACTADOR

En el caso del módulo de descompactación, se deben cumplir las siguientes especificaciones:

El algoritmo de descompactación debe separar a los bloques con el fin de evitar traslapamientos, sin tener restricciones en el tamaño de la hoja que sea necesario utilizar.

Datos de entrada:

- Archivo LAYOUT con información acerca de los nombres, tamaño colocación y separación mínima de los componentes que conforman a la hoja de esquemáticos.

Datos de salida:

- Archivo LAYOUT_UNO, con la nueva colocación de los componentes.

3.3.1 Estrategia para desarrollar el descompactador

Para desarrollar este módulo se pensó que como primera fase era necesario:

a) Determinar un algoritmo de descompactación empleado para descompactar topologías de C.I..

b) Modificar el algoritmo de acuerdo a nuestras necesidades.

Para llevar a cabo el primer punto, se encontró el algoritmo desarrollado por Y.Z. Liao y C.K. Wong. [8]

Originalmente este algoritmo fue desarrollado para compactar topologías de Circuitos Integrados (C.I.), en el cual la topología del C.I. se encuentra representado por paralelepípedos y las restricciones que deben cumplir, se refieren a las reglas de diseño impuestas por la tecnología con la cual se fabrique el C.I.

En nuestro caso, lo que necesitamos es un algoritmo para separar bloques que representan una topología, y las restricciones son empleadas para dejar espacios que puedan ser empleados para que el enrutador interconecte dichos módulos, por lo que agregando restricciones de usuario se logra adaptar este algoritmo a nuestras necesidades.

Como entrada se recibe una topología de una hoja de esquemáticos y un conjunto de restricciones dadas por el usuario para la separación que deben cumplir dichos bloques.

De acuerdo a las especificaciones de este módulo, como entrada se recibe la lista de interconexiones FORNET de un esquemático y como salida se da una nueva lista de interconexiones con las nuevas posiciones de los bloques.

En lo que respecta al inciso b), para llevar a cabo la modificación del algoritmo escogido, la estrategia a seguir es:

a) Diseñar un programa que lleve a cabo el algoritmo de descompactación.

b) Obtener la información de la topología de la hoja de esquemáticos a partir de la lista de interconexiones.

c) Obtener la información de las restricciones con las cuales deben cumplir los bloques.

d) Ejecutar el programa de descompactación con la información de la topología de la hoja de esquemáticos y las restricciones a cumplir.

e) Como salida, el programa de descompactación debe proporcionar un archivo con las nuevas posiciones de los bloques, que representan a los componentes.

3.3.2 Algoritmo de Descompactación

Como se mencionó anteriormente, el siguiente algoritmo fue desarrollado por Y.Z. Liao y C.K. Wong [8].

El algoritmo emplea teoría de grafos, y es muy similar al problema de encontrar la trayectoria más larga.

El algoritmo supone que las restricciones de espaciamiento son del tipo cota inferior, en el sentido de que entre dos elementos, solamente se permite un espaciamiento igual o mayor al dado.

Para que este algoritmo nos sea útil, deben poder ser agregadas restricciones de igualdad o de cota superior sobre pares de elementos, estas restricciones son dadas por el usuario.

Como entrada se recibe una topología de una hoja de esquemáticos y un conjunto de restricciones dadas por el usuario para la separación que deben cumplir dichos bloques.

Como salida se entrega una nueva topología que satisface las restricciones de espaciamiento.

Algoritmo:

La topología que se recibe se transforma en un grafo dirigido, en donde las aristas representan las restricciones y los vértices a los bloques.

a) El primer paso consiste en agrupar a los bloques de la topología recibida.

   Para ello se agrupa a los bloques que se encuentran en una misma coordenada horizontal en un grupo, con esta operación se crean n+1 grupos, etiquetados como V0, V1, V2, ... Vn

   A los grupos V0 y Vn se les denomina frontera izquierda y frontera derecha respectivamente.

b) El siguiente paso consiste en generar el grafo dirigido G.

Los nodos de G se obtienen mapeando cada Grupo Vi (i=0...n) a un nodo de G.

La operación de mapeo es unívoca por lo que se usará la notación Vi para denotar a un grupo o a un nodo de G.

c) A continuación se convierten los requerimentos de espaciamiento entre nodos a aristas de la Gráfica G.

Se empleará la notación Vi(X) para denotar la posición del nodo Vi en la topología respecto al eje de las abscisas.

Entonces, los requerimentos mínimos de espaciamiento entre bloques se traducen a aristas dirigidas del grafo, esta operación se realiza para cada uno de los bloques y sus vecinos de la derecha.

Esto es, supongamos que se está analizando al bloque Va y el bloque Vb está a su derecha (Va(X) < Vb(X)), entonces para evitar traslapamientos se requiere que el bloque Vb esté separado por lo menos p unidades de Va, para representar esta restricción se agrega una arista dirigida a la G que vaya de Va a Vb y con un valor de p.

d) Ahora se agregan a G las restricciones impuestas por el usuario.

De nuevo asuma que Va y Vb son dos grupos y que Vb está a la derecha de Va, si el usuario requiere que el bloque Vb esté alejado del bloque Va al menos p unidades, se agrega una arista dirigida que vaya de Va a Vb con un valor p, como en el caso anterior, pero si el usuario requiere que Vb no esté más alejado de Va más de q unidades, entonces se agrega una arista dirigida de Vb a Va con un valor de -q.

Si el usuario requiere que el bloque Vb esté alejado exactamente r unidades de Va, esto se

traduce a agregar 2 aristas a la gráfica G una que vaya de Va a Vb con un valor r y una que vaya de Vb a Va con un valor de -r.

En general, una arista (Vi,Vj) de valor Lij en G representa una inecuación de la forma:

$$Vi(X) + Lij <= Vj(X)$$

Una arista dirigida (Vi,Vj) se denominará arista hacia la derecha si Vj está a la derecha de Vi, en caso contrario se denominará arista hacia la izquierda.

e) En este punto, ya contamos con la gráfica de restricciones, la cual puede tener multiples aristas para un par de vértices cualesquiera.

El siguiente paso es dejar únicamente la arista con mayor valor para obtener las trayectorias más largas dado que esta arista contiene las restricciones de las demás aristas.

Nuestro primer paso es normalizar a cero la colocación de las aristas, para ello se fuerza a que $V0(X) = 0$.

Sean V y E los conjuntos de nodos y aristas de la gráfica de restricciones, esto implica:

$$G = (V,E)$$

Sean Ef y Eb los conjuntos de aristas dirigidas a la derecha y a la izquierda respectivamente, entonces podemos definir al grafo Gf=(V,Ef).

El algoritmo de compactación es:

i) Obtener las aristas con mayor valor de Gf de tal forma que se cumpla:

Para i=0,...,n

$$Vi(X) = \max_{0<=j<=n} (\text{Valor de } V(X) + Lij)$$

Este algoritmo es del tipo de encontrar la trayectoria más larga.

ii) Para cada una de las aristas del conjunto Eb ver que se satisfaga la restricción impuesta por ellas.

Si no se cumple las restricciones de las aristas hacia la izquierda mover a los bloques para que las cumplan y volver al inciso a.

Si todas las aristas hacia la izquierda son satisfechas terminar el algoritmo.

3.3.3 Pseudocódigo del algoritmo de descompactación

```
Procedimiento de Descompactación

cuenta = 0;
V0(X) = 0;
Para i = 0  hasta n Hacer
   Begin
        Vj(X) = -infinito;
   End;
Repetir
    Bandera =  Verdadero;
    Llamar procedimiento Longest_path(G)
    Para cada arista dirigida hacia la izquierda
    (Vi,Vj) en Gf realizar lo siguiente
      Begin
```

```
          Sí Vj(x) < Vi(x) + Lij entonces
            Begin
               Vj(x) = Vi(X) + Lij;
               Bandera = Falso;
            End;
        End;
       Cuenta = Cuenta + 1;
       Sí Cuenta = b+1  Y  Bandera = Falso Entonces
          Begin
             Escribir (Restricciones Inconsistentes);
             Termina Algoritmo.
          End;

 Hasta que Bandera sea Verdadero.

            Procedimiento Longest_path:

crear una pila:
meter Vo en la pila;
Para i=0 to n Do
  Begin
     Grado_entrada[Vi]  =   Grado de entrada de Vi
                            en  el Grafo.
     Mientras la pila no esta vacia
       Begin
         Saca el vértice (Vt) que está en el tope de la
         pila
            Para cada arista que conecte a
            los vértices (Vt, Vj) en el grafo G.
               Begin
                 Vj(x) = Mayor (Vj(x), Vt(x) + Ltj;
             Grado_entrada[Vj] = Grado_entrada[Vj]-1;
             Sí Grado_entrada[Vj] == 0 Entonces
                    Meter Vj  en la pila;
               End;
        End;
  End;
```

3.3.4 Diseño del programa de descompactación.

Para diseñar este programa, basta con seguir el algoritmo de descompactación, lo que es importante en su diseño es la manera en como va a recibir la información referente a la topología y a las restricciones de espaciamiento que debe haber entre bloques.

a) Topología.

Para la información referente a la topología, se propone lo siguiente:

Para hacer referencia a un bloque se le debe de dar un nombre único, además se debe proporcionar su tamaño y su colocación original dentro de la topología.

Para almacenar esta información se propone la siguiente estructura de datos:

- Nombre del bloque
- Tamaño del bloque
- Posición

b) Restricciones

Dado que cada bloque posee un nombre único, las restricciones de espaciamiento se pueden indicar para cada pareja de bloques en el caso en que haya restricciones de uno respecto al otro.

Se propone que las restricciones sean dadas de la siguiente manera:

Nombre_bloque_1 Nombre_bloque_2 Número

El número anterior nos representa el número de unidades que debe de haber de separación entre los bloques.

Una característica importante del algoritmo que debe ser tomada muy en cuenta, es que la descompactación se efectúa

en un sólo sentido del eje de coordenadas, por lo que para tomar en cuenta las restricciones en el eje de las abscisas y en el eje de las ordenadas, el algoritmo debe ser ejecutado dos veces.

Lo anterior implica que se deben proporcionar 2 tipos de restricciones las que se refieran al eje de las abscisas y las que se refieren al eje de las ordenadas.

Se propone que el conjunto restricciones horizontales se almacenen en un archivo denominado rest_x y las restricciones referentes al eje vertical se almacenen en otro archivo denominado rest_y.

Finalmente, se propone que la información de la topología se almacene en un archivo denominado topología.

de tal manera que la sintaxis de ejecución del programa sea:

reubica topología rest_x rest_y

3.3.5 Obtención de los bloques que representan la topología

La tarea de determinar a los bloques que representan la topología de la hoja de esquemáticos es una tarea sencilla, con la estructura que se cuenta, debido a que en la lista de interconexiones, se tiene la posición de cada componente, que para el caso del programa espaciador es un bloque, y además se cuenta con el tamaño del componente, el cual es el tamaño del bloque.

Tomando en cuenta lo anterior, lo que hay que hacer es leer el netlist y generar una lista con los componentes que conforman al netlist guardando información relativa al tamaño y posición de cada uno de ellos.

3.3.6 Obtención de las restricciones

Para obtener las restricciones de espaciamiento, bastará con determinar el tamaño del bloque original, respecto al del bloque que lo va a reemplazar, la diferencia que exista entre estos dos valores más la distancia del bloque más cercano que se encuentre a la derecha del bloque, nos proporcionará el número de unidades mínimo que cualquier bloque debe estar separado en el eje de las abscisas respecto al bloque en cuestión.

Para determinar las restricciones, se propone emplear el siguiente algoritmo:

```
Begin
   Meter a los bloques en una lista
   Ordenar la lista respecto al eje de las abscisas, en
orden creciente
    i = 1;
    n = número de elementos de la lista


   Mientras que i < n
      Begin
         j= i+1
         Inc = Tamaño del componente Original eje X -
               Tamaño del componente de Reemplazo eje X.
          rest = infinito;
          elem = 0;
          Mientras que j <= n
           Begin
             If Bj esta enfrente de Bi Then
                 Begin
                  If rest < diferencia ( Origen_X(Bj) -
                  Origen_X(Bi))
                  Then
                   Begin
                     rest = diferencia ( Origen_X(Bj ) -
                            Origen_X(Bi)) + Inc;
                     Elem = j;
                   End;
                 End;
           End;
          If Elem <> 0 Then
            Begin
```

```
            escribe  en  rest_x(  Nombre(Belem)  Nombre(Bi)
            rest);
            End;
      End;

   i = 1;

   Mientras que i < n
      Begin
         j= i+1
         Inc = Tamaño del componente Original eje Y -
               Tamaño del componente de Reemplazo eje Y.
          rest = infinito;
          elem = 0;
          Mientras que j <= n
           Begin
             If Bj esta arriba de Bi Then
                 Begin
                 If rest < diferencia ( Origen_Y(Bj) -
                                        Origen_Y(Bi))
                  Then
                      Begin
                         rest = diferencia(Origen_Y(Bj)-
                                 Origen_Y(Bi)) + Inc;
                         Elem = j;
                      End;
                 End;
           End;
          If Elem <> 0 Then
            Begin
              escribe  en  rest_y  (Nombre(Belem)  Nombre(Bi)
rest);
            End;
      End;

End
```

3.3.7 Formato de los datos de entrada y salida

Los formatos de los archivos de datos de entrada para el módulo descompactador son:

Formato de los archivos con datos de entrada:

NETFOR_UNO: Debe cumplir con la gramática definida en la sección 3.1.2

LAYOUT: Archivo de datos escritos en formato ASCII, de la siguiente manera:

<Nombre del bloque> <tamaño> <origen> <sep. mínima>
.
.
.
FIN

La información de tamaño, origen y sep. mínima se deben expresar como parejas de números que indiquen la información referente a las abscisas y a las ordenadas.

La palabra FIN debe ser el último dato en el archivo ya que esto sirve para indicar donde termina la información.

Un ejemplo de como debe verse el contenido del archivo LAYOUT es:

```
I$1  3.0  4.0  4.5  5.0  0.0   0.0
1$2  3.0  4.0  7.0  5.0  0.0   0.0
FIN
```

Formato de los archivos con los datos de salida:

LAYOUT_UNO: El formato de este archivo es idéntico al del archivo de entrada LAYOUT:

<Nombre del bloque> <tamaño> <origen> <sep. mínima>
.
.
.
FIN

Un ejemplo de como se verá el contenido de este archivo es:

```
I$1   3.0   4.0   4.5   5.0   4.0   2.5
I$2   3.5   4.0   7.0   5.0   4.0   2.5
FIN
```

3.4 Módulo ENRUTADOR

3.4.1 Especificación del enrutador

Diseñar y construir un enrutador que sea capaz de enrutar parejas de puntos sin cruzar por áreas restringidas y sin provocar la sobreposición de segmentos de un alambrado, sobre otro alambrado, se permitirá el cruce de segmentos de alambrados diferentes, siempre y cuando el punto de cruce no sea un vértice de ninguno de los alambrados en cuestión.

Como entradas el enrutador recibirá:

- Archivo con los puntos a enrutar, denominado PUNTOS.
- Archivo BLOQUE con información acerca de las áreas que no deben ser cruzadas (representadas por segmentos de línea horizontales y verticales).

Como salida el enrutador deberá proporcionar:

- Archivo con las coordenadas de los puntos que constituyen los segmentos que representan a las nets, el cual se denomina ENRUTADO.

- Archivo con las coordenadas de los puntos que no fue posible alambrar.

3.4.2 Estrategia

De la misma manera que en el caso del descompactador, nuestra primera tarea es la de obtener un algoritmo de enrutado que nos pueda ser útil, teniendo esto en mente, se encontraron que los algoritmos del tipo "Maze-Runners"

cumplen con la característica deseada de alambrar parejas de puntos, con la ventaja de que si existe una ruta para llevar a cabo la interconexión, éstos la encontrarán.

Sin embargo, su principal desventaja es el necesitar de grandes cantidades de memoria.

Teniendo en cuenta lo anterior, se decidió buscar un algoritmo alternativo, encontrándose el algoritmo de Hightower, el cual emplea la metodología Línea de escape, Punto de escape.

Como primera opción, se programó el algoritmo de Hightower, obteniendo las siguientes características ventajosas:

a) El algoritmo emplea listas ligadas para almacenar a las líneas que representan a las restricciones horizontales, y que no son más que los contornos de los bloques de la topología de la hoja de esquemático.

   Para representar una línea se emplea 2 parejas de números.

b) El proceso de mejorar a las trayectorias encontradas es efectivo.

c) El tiempo de ejecución para las primeras trayectorias es del orden de milisegundos.

Sin embargo, se encontraron las siguientes desventajas:

a) Si no existe una trayectoria de enrutado el algoritmo se degrada en el algoritmo de Lee y la memoria necesaria para almacenar las líneas de escape ya usadas crece explosivamente, de la misma manera, el tiempo de ejecución para determinar que una línea no tiene escape es relativo al espaciamiento de la malla de enrutado.

Para espaciamientos tales que se tuviera una malla de 500 por 500, el tiempo invertido para determinar que no habia trayectoria de enrutado es del orden de 4 minutos.

b) A medida que el enrutado va siendo más denso el tiempo empleado en enrutar una pareja de puntos se va incrementando.

c) Debido a su propia definición el algoritmo de Hightower puede ser que no encuentre una trayectoria de enrutado, a pesar de que ésta exista. Aunque en el caso de los problemas de enrutado realizados nunca se encontró este problema.

Debido a las caracteristicas adversas anteriores, se decidió programar el algoritmo de Lee, obteniéndose las siguientes caracteristicas:

Para programar el algoritmo de Lee se declara un arreglo de dos dimensiones que representa a la malla de enrutado, el tamaño máximo de un arreglo bidimensional que puede ser pedido en la computadora GOULD es de orden 500 x 500, de números enteros de 32 bits, lo cual representa en memoria alrededor de 1 Mega Byte.

El tiempo empleado en determinar que no pueden ser encontrada una trayectoria de enrutado es alrededor de dos minutos.

Para todos los casos en los que existe trayectoria de enrutado, el algoritmo la encontró.

El algoritmo de Lee presenta las siguientes desventajas:

a) El enrutado presentado contiene demasiados codos.

b) En ocasiones, los codos de las trayectorias de enrutado bloquean puntos de enrutado, haciendo imposible el poder enrutar los puntos bloqueados.

Por ejemplo:

Se desea alambrar los siguientes puntos:

A con D

C con B

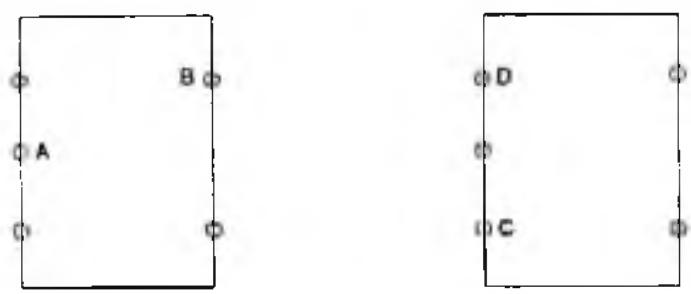


Figura 7.

Los puntos se encuentran en la frontera de los siguientes bloques:

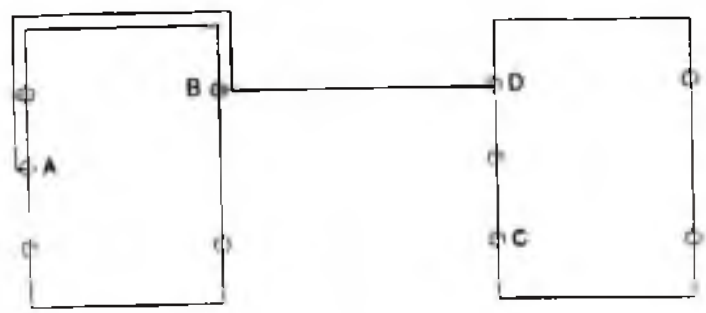


Figura 8.

Como se puede observar ya no existe manera de alambrar B con C debido a que el codo z de la trayectoria de enrutado del punto A al punto D bloque a B y si se alambra esta parte se causa conexión entra los alambrados de los puntos AD y BC lo cual no se desea.

Tomando en cuenta los argumentos anteriores se decidió diseñar un enrutador que empleará las caracteristicas más sobresalientes de los dos enrutadores presentados:

- La determinación de la trayectoria de enrutado se encuentra mediante el algoritmo de Lee.
- Para eliminar la mayor cantidad de codos de la trayectoria encontrada se emplean métodos del algoritmo de Hightower.

Empleando la misma filosofía se hace que las líneas de interconexión no queden muy juntas a los cuerpos de los bloques, con el fin de evitar bloquear las posibles salidas de los puntos de enrutado.

En el caso particular de éste bloque, la estrategia de diseño fue la de programar los algoritmos de Lee y de Hightower y en base a su ejecución determinar el mejor, sin embargo al programar ambos algoritmos y probarlos se decidió el diseñar un algoritmo híbrido que hiciera uso de las características más sobresalientes encontradas de ambos algoritmos.

3.4.3 Pseudocódigo del Algoritmo de enrutado

Como se mencionó anteriormente, el algoritmo de enrutado empleado es una mezcla de los conceptos desarrollados en el algoritmo de Lee [7] y en el de Hightower [11].

En el algoritmo de Lee se emplea una malla que representa la malla de enrutado.

Además, se empleará una segunda malla que contendrá las trayectorias de las nets.

Para realizar esta representación a cada net se le asignará un número que será único.

Por otro lado, en el algoritmo de Hightower, se emplean listas de líneas que representan a las restricciones y listas de líneas que representan a las nets, por lo tanto, las restricciones y las mallas tendrán una doble representación, una para el algoritmo de Lee y la otra para el de Hightower.

Cabe mencionar, que para cada Net se tiene una lista de puntos que deben ser enrutados.

La sintaxis del archivo con los puntos a enrutar, es:

NET <Nombre de la Net>
Número de puntos que conforman a la net
<abscisa> <ordenada>
.
.
.
<abscisa> <ordenada>

```
    NET <Nombre de la Net>
Número de puntos que conforman a la net
<abscisa> <ordenada>
.
.
.
<abscisa> <ordenada>

.
.
.
```

El pseudocódigo del algoritmo resultante es el siguiente:

Algoritmo del Programa Principal

a) Abrir el archivo denominado LAYOUT, el cuál contiene la información de la topología.

b) Abrir el archivo LIST_PTS, el cuál contiene la lista de puntos a ser enrutados.

c) Leer la topología.

d) Inicializar la malla que representa al enrutado insertando -1 en todas sus posiciones.

e) Inicializar la malla que contiene las trayectorias de las nets con un valor de 0.

f) En las áreas que representan a los bloques que no deben cruzarse, colocar el valor de 0.

g) Inicializar las listas de lineas que contienen las restricciones.

h) Invocar al enrutador

i) Liberar la memoria ocupada por las listas de restricciones.

j) Cerrar archivos.

k) Fin del programa

Función del enrutador

a) Leer del archivo LIST_PTS el nombre de la net.

b) Hacer que la variable num_net, tenga un valor de 1.

c) Leer todos los puntos que conforman a la net y almacenarlos en la lista de puntos denominada La.

d) Si el número de elementos de la lista La, es menor o igual a 1:

   i) Leer del archivo LIST_PTS el nombre de la siguiente net.

      Si ya no hay más información que leer, terminar el algoritmo de enrutado.

   ii) Incrementar el valor de num_net en una unidad.

   iii) Ir al inciso c).

e) Leer el primer elemento de la lista La y denominarlo pt1.

f) Leer el segundo elemento de la lista La y nombrarlo pt2.

g) Hacer que el punto origen sea igual a pt1.

h) Hacer que el punto destino sea igual a pt2.

i) Hacer que las posiciones correspondientes al punto destino y al punto origen en la malla tengan un valor de 0.

j) Generar el frente de onda a partir del punto origen.

k) Determinar el siguiente punto a ser considerado, para llegar al punto destino y almacenar su contenido en px, las coordenadas de este punto se almacenan en las variables posx y posy.

l) Meter a la lista de puntos (Lp) el punto origen.

m) Determinar cuál es el valor de la malla que representa a las nets en la posición (posx,posy), si es igual al número de la net que se está alambrando, esto significará que ya se ha encontrado una trayectoria válida, y por lo tanto se debe continuar con el siguiente punto de la net o con la siguiente lista de puntos que representan a una net.

n) Si las coordenadas del punto px son iguales a las del punto destino:

   i) Mejorar la trayectoria encontrada.

   ii) Reflejar en la malla de nets la trayectoria de la net.

iii) En la malla de enrutado, colocar ceros en las esquinas de la trayectoria de enrutado, para evitar contactos no deseados.

iv) Eliminar el primer punto de la lista de puntos La.

v) Hacer que pt1 = pt2.

vi) Leer el primer punto de la lista La y almacenarlo en pt2.

vii) Ir al inciso f).

o) Ir al inciso k).

Algoritmo para mejorar la trayectoria de las nets.

a) Eliminar los puntos que no representan esquinas en la trayectoria.

b) Hacer que la trayectoria tenga el menor número posible de esquinas.

c) Hacer que la net se aleje de las áreas restringidas.

A continuación se presentan ejemplos de cada uno de estos refinamientos:

Puntos que no son esquinas:

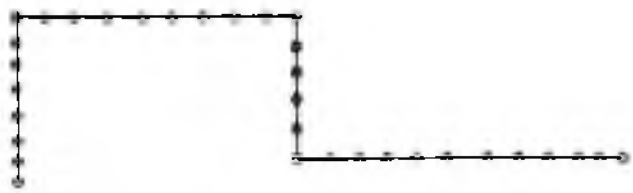


Figura 9.

Tratar de que la trayectoria de enrutado tenga el menor número de esquinas:

La siguiente net contiene esquinas que pueden ser eliminadas:

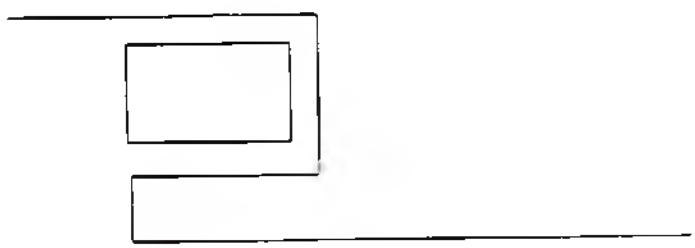


Figura 10.

Quedando como:

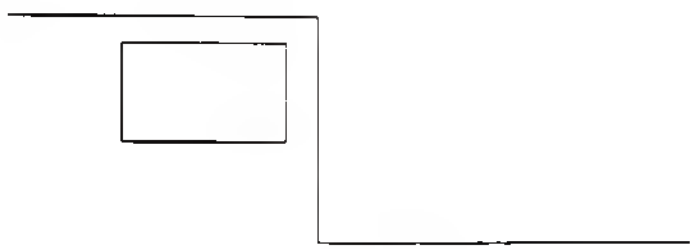
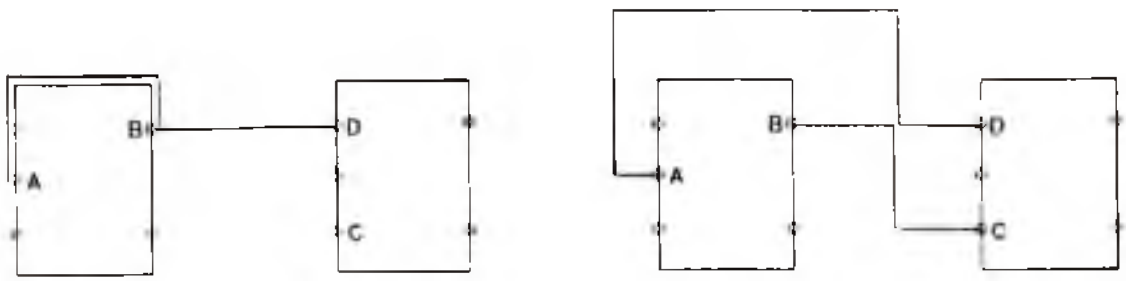


Figura 11.

Hacer que las nets se alejen de las áreas restringidas para evitar caso como el siguiente:



Antes de la mejora

Despues de la mejora

Figura 12.

3.4.4 Formato de los datos de entrada y salida

Formato de los archivos con los datos de entrada:

PUNTOS: NET <número de puntos> <número de la net>
<abscisa> <ordenada>
.
.
.
NET <número de puntos> <número de la net>
<abscisa> <ordenada>
.
.
.

BLOQUE <Nombre del bloque> <tamaño> <origen> <sep. mínima>
.
.

Formato de los archivos con los datos de entrada:

ENRUTADO: <abscisa> <ordenada> <abscisa> <ordenada>
.
.
.

3.5 Módulo GENSCR

La especificación de este bloque es la siguiente:

Datos de entrada:

- Archivo FORNET_DOS
- Archivo ENRUTADO

Datos de salida:

- Script ejecutable por el editor de redes, con la información para generar la nueva hoja de esquemáticos.

Para generar este módulo, se requiere de un analizador lexicográfico que se encargue de identificar la información almacenada en FORNET_DOS y genere las instrucciones que indiquen a NETED como generar el nuevo esquemático.

Por ejemplo:

A partir del siguiente fragmento de un NETLIST:

```
COMPONENTE:  /user/gen_lib/$and2
HANDLE:      I$10
CENTRO_ROT:  0.0 , 0.1
ORIGEN:      3.0 , 3.0
TAMANO:      0.9 , 0.7
ROTACION:    90
ESPEJO:      0

PROPIEDAD:  INST
    VALOR:   @#FF1#@
    FIJA:    FALSE
    VISIBLE: FALSE
    ROTACION:   90
    TAMANO:     0.075
```

```
JUSTIFICACION:    B L
ORIGEN:           3.0 , 3.0
OFFSET:           -0.102 , 0.09
```

El generador de script producirá las siguientes instrucciones:

```
MARK 20,20
ACTIVATE COMPONENT /user/gen_lib/$and2  -SELECT
ROTATE 90
MOVE 3 3
TEMPLATE TEXT  B L 90  0.075  -HIDDEN
ADD PROPERTY INST FF1    2.898  3.09
```

Posteriormente, se requiere de un programa que se encargue de leer la lista de puntos que representan a las líneas que conforman a las trayectorias, esta es una tarea sencilla, porque el comando que indica a NETED trazar una línea es:

NETROUTE <abscisa primer punto> <ordenada primer punto> -START
NETROUTE <abscisa segundo punto> <ordenada segundo punto>

Por ejemplo:

Sea la lista de líneas siguiente:

```
1  3     5  3
5  3     5  5
5  5     8  5
```

El código generado será:

```
NETROUTE 1 3 -START
NETROUTE 5 3
NETROUTE 5 3 -START
NETROUTE 5 5
NETROUTE 5 5 -START
NETROUTE 8 5
```

3.6 Módulo INTERFAZ

Como se indicó en la sección 2.6, la función de este módulo será la de leer una lista de interconexiones y un archivo con la nueva colocación de los componentes de la lista de interconexiones y genera la información requerida por el módulo enrutador.

Datos de entrada:

- Archivo FORNET_UNO.
- Archivo LAYOUT_UNO

Datos de Salida:

- Archivo PUNTOS, el cual contiene la lista de puntos a ser enrutados.
- Archivo BLOQUES, con la información acerca de las áreas que no deben ser cruzadas.
- Archivo FORNET_DOS.

3.6.1 Estrategia de diseño del módulo Interfaz

Este módulo debe proporcionar la lista de puntos a ser enrutados, las áreas restringidas y la lista de interconexiones FORNET_DOS.

Para generar la lista de puntos se propone el siguiente algoritmo:

i) Leer de la lista de interconexiones la parte referente a los componentes, y almacenar la siguiente información acerca de cada uno de ellos:

handle.
path.
centro de rotación.
origen.
rotación.
espejo.
lista de pines del componente.

Cada elemento de la lista de componentes debe tener la siguiente información:

Nombre del pin.
Posición relativa respecto al origen del componente.
Apuntador al siguiente pin.

Esta información será empleada para generar la lista de puntos, debido a que la información referente a cada net en NETFOR se da como:

```
<conexión>   ::= NET NET_HANDLE  <net>
<net>        ::= COMPONENTE : PATH [ INS_HANDLE ] PIN
: NET_HANDLE net_name net
      | EPSILON

   <net_name> ::= [ CADENA ]
         | [ ENTERO ]
         | [ ]
```

Ver la gramática de la sección 3.1.2.

Por ejemplo, para indicar la conectividad de una hoja de esquematicos, se pueden tener cosas como estas:

```
NET : N$12
  COMPONENTE : /user/gen_lib/$and2 [I$12]
               PIN : V$123 [ I0 ]
  COMPONENTE:  /user/gen_lib/$or2 [I$6]
               PIN: V$24 [ OUT ]
```

Del fragmento de NETLIST anterior, se puede observar que la única manera de obtener la posición del componenete es mediante su handle, debido a que el handle es valor del handle único en la hoja de esquemáticos.

Por lo tanto, el árbol balanceado debe estar ordenado de acuerdo al handle de los componentes.

ii) Leer el archivo LAYOUT_UNO y modificar la posición de cada componente.

iii) Leer de la lista de interconexiones la información referente a la conectividad, y para cada net generar la lista de puntos a ser conectados.

La información de la colocación de los puntos a enrutar se obtiene a partir de la posición de los pines, por lo que se debe tener presente que, la posición de los pines es relativa respecto al origen y si el componente esta rotado o reflexionado, se debe calcular cuál será la nueva posición del pin.

Para generar la lista de interconexiones FORNET_DOS, se proponen las siguientes actividades:

i) Leer la tabla con las nuevas posiciones de cada componente del esquemático.

ii) Leer la información referente a cada componente, modificar su posición y evaluar las nuevas posiciones de cada uno de sus pines, tomando en cuenta que es relativa al origen y a la rotación y reflexión del componente.

iii) Escribir el archivo FORNET_DOS

En lo que respecta a la información de las áreas restringidas, se propone que el enrutador la lea en el mismo formato en el que el módulo DESCOMPACTADOR la genera.

3.6.2 Algoritmo

Tomando en cuenta las propuestas de la sección anterior, para generar los datos de salida deseados, se propone el siguiente algoritmo:

```
Begin
   Leer la información contenida en LAYOUT_UNO, acerca de la
   nueva posición de cada componente y almacenarlo en un
   arreglo llamado BLOQUE.

   Crear un árbol binario denominado tree.

   Mientras haya componentes en FORNET_UNO:

   Begin
     Leer la información del componente actual en FORNET_UNO.

     Buscar el handle del componente en el arreglo BLOQUE.
     Modificar el valor del origen del bloque.
     Escribir la nueva información en FORNET_DOS.

     Para cada pin de la lista de pines del componente
     Begin
       Evaluar la posición del pin tomando en cuenta si el
       componente esta rotado o no y si esta reflejado.

       Alamacenar la información en tree.
       Escribir los datos evaluados en FORNET_DOS.

     End;
   Leer del esquemático la información referente a la
   conectividad.
   Para cada net:
    Begin
      Escribir en el archivo punto el nombre de la net.

      leer  la lista de pines que la forman.
      Buscar en tree al componente.

      Para cada pin de la lista:

         Begin

            Buscar el pin en la lista de pines de tree.

            Escribir en el archivo PUNTOS la posición del pin,
            encontrada en tree.

         End;

    End;
End.
```

Para realizar las tareas anteriores se propone diseñar e implementar las siguientes funciones:

a) Analizador lexicográfico que se encargue de leer la información de NETFOR_DOS y una función que se encarge de abrir y leer el archivo LAYOUT_UNO.

b) Diseño y construcción de funciones para efectuar rotaciones y reflexiones sobre un punto que sirve como centro de rotación y reflexión.

c) Diseño y construcción de funciones para manejar un árbol binario balanceado, el cual estará ordenado de acuerdo al handle de cada componente.

3.6.3 Formato de los datos de entrada y salida

Formato de los archivos con los datos de entrada:

FORNET_UNO: Debe cumplir con la gramática de la sección 3.1.2.

LAYOUT_UNO: Idéntico al formato del archivo LAYOUT, definido en la sección 3.3.7, esto es:

<Nombre del bloque> <tamaño> <origen> <sep. minima>

.
.
.

FIN

Formato de los archivos con los datos de salida:

FORNET_DOS: Debe cumplir con la gramática de la sección 3.1.2.

PUNTOS: NET <numero de puntos> <número de la net>
punto 1
punto 2
.
.
.

BLOQUES: Idéntico al formato del archivo LAYOUT, definido en la sección 3.3.7, o sea:

<Nombre del bloque> <tamaño> <origen> <sep. minima>
.
.
.
FIN

3.7 Módulo GENES

Al desarrollar los módulos NETLISTER, CAMBIADOR, DESCOMPACTADOR, ENRUTADOR y GENSCR estamos en posición de armar al generador de esquemáticos.

Lo que resta es emplear un programa maestro que se encargue de invocar a los módulos antes mencionados.

Para diseñar este módulo se propone el siguiente algoritmo:

a) Invocar al módulo GENES

b) El módulo GENES crea un proceso hijo que ejecuta al módulo NETLISTER, y suspende su operación esperando por una señal.

c) El módulo NETLISTER invoca a NETED y ejecuta el script que obtiene la lista de interconexiones (FORNET), una vez que termina su ejecución envia una señal al módulo GENES para indicarle que ya se ha obtenido la lista de interconexiones.

d) El módulo GENES despierta con la señal y genera otro proceso hijo que se encarga de ejecutar al módulo CAMBIADOR, y vuelve a suspender su ejecución.

e) El módulo CAMBIADOR se encarga de realizar los cambios en la lista de interconexiones FORNET, generando la lista de interconexiones FORNET_UNO.

Cuando termina su ejecución envia una señal al módulo GENES indicando que va a terminar su ejecución.

f) Una vez más, el módulo GENES despierta y crea un nuevo proceso hijo que se encarga de ejecutar al módulo DESCOMPACTADOR, y vuelve a suspender su ejecución.

g) Una vez que el módulo DESCOMPACTADOR ha terminado de reubicar a los componentes y ha generado el archivo con la nueva topología LAYOUT_UNO, envía una señal al módulo GENES indicando que puede seguir su operación.

h) Por cuarta vez, el módulo GENES crea un proceso hijo que se encarga de ejecutar al módulo INTERFAZ y suspende su ejecución.

i) El módulo INTERFAZ genera los archivos PUNTOS, FORNET_DOS y BLOQUES, una vez que termina su ejecución envía una señal a GENES para que prosiga con su ejecución.

j) GENES continua su ejecución generando un proceso hijo que se encarga de invocar al enrutador, y suspende su ejecución.

k) Una vez que el enrutador ha terminado su tarea, envía una señal al módulo GENES para que siga con su ejecución.

l) El módulo GENES genera un nuevo proceso hijo que se encarga de invocar al módulo GENSCR y suspende su ejecución.

m) Cuando GENSCR ha terminado de generar al script envía una señal a GENES para que prosiga con su ejecución.

n) Finalmente, el módulo GENES crea su último proceso hijo y lo hace que ejecute a NETED para que interprete el script generado y genere la nueva hoja de esquemático.

Sin embargo, en el CINVESTAV, no se cuenta con un compilador del lenguaje C para la APOLLO que pertenece al Departamento de Ingeniería Eléctrica.

Como consecuencia de lo anterior, la operación del módulo GENES se limita a ejecutar los bloques CAMBIADOR, DESCOMPACTADOR, ENRUTADOR y GENSCR.

La obtención de la lista de interconexiones FORNET se debe realizar invocando a NETED y posteriormente ejecutar el comando:

do /user/lenin/source/bin/netlister FORNET <-

Una vez que el script ha obtenido el archivo FORNET, se transfiere a la GOULD y se invoca al programa ejecutable GENES, el cual realiza las siguientes actividades:

a) GENES genera un proceso hijo que invoca al módulo CAMBIADOR y suspende su ejecución.

b) El módulo CAMBIADOR se encarga de realizar los cambios en la lista de interconexiones FORNET, generando la lista de interconexiones NEWFORNET.

Cuando termina su ejecución envia una señal al módulo GENES indicando que va a terminar su ejecución.

c) Una vez más, el módulo GENES despierta y crea un nuevo proceso hijo que se encarga de ejecutar al módulo DESCOMPACTADOR, y vuelve a suspender su ejecución.

d) Una vez que el módulo DESCOMPACTADOR ha terminado de reubicar a los componentes y ha generado el archivo con la nueva topologia LAYOUT, envia una señal al módulo GENES indicando que puede seguir su operación.

e) El módulo GENES crea un proceso hijo que se encarga de ejecutar al módulo INTERFAZ y suspende su ejecución.

f) El módulo INTERFAZ genera los archivos PUNTOS, FORNET_DOS y BLOQUES, una vez que termina su ejecución envía una señal a GENES para que prosiga con su ejecución.

g) Por cuarta vez, el módulo GENES crea un proceso hijo que se encarga de ejecutar al módulo ENRUTADOR y suspende su ejecución.

h) Una vez que el enrutador ha terminado su tarea, envía una señal al módulo GENES para que siga con su ejecución.

i) El módulo GENES genera un nuevo proceso hijo que se encarga de invocar al módulo GENSCR y suspende su ejecución.

j) Cuando GENSCR ha terminado de generar al script envía una señal a GENES para que prosiga con su ejecución.

Una vez que se tiene el script para NETED, se transfiere de la GOULD a la APOLLO y se invoca a NETED.

Y se hace que NETED ejecute el siguiente comando:

do genera_diseño <-

Ejecutando los pasos anteriores, se genera una hoja de esquemáticos a partir de una hoja ya generada.

Pero surge la pregunta de saber si la hoja de esquemáticos generada es confiable.

Para llevar a cabo una verificación de que el esquemático generado es equivalente al esquemático original, se deben tomar en cuenta los siguientes aspectos:

Si la tabla de reemplazo está bien realizada, no puede haber errores debidos a reemplazar un componente por un componente erróneo.

Para ello se pide a la gente que genera la tabla de intercambio que la verifique de manera manual para ver si esta bien o no.

Por otro lado, en lo referente a la conectividad esta puede ser la que tenga potencialmente más problemas.

Es debido a estas razones por las cuales se propone realizar un programa que verifique que la conectividad de ambas hojas de esquemáticos sea equivalente.

Para llevar a cabo este programa se sugiere emplear conectividad orientada a componentes y para tener manera de saber que componente de un esquemático corresponde al del otro, se propone agregar a cada componente del nuevo esquemático una propiedad denominada PRUEBA cuyo valor será igual al handle del componente original que está reemplazando.

Para llevar a cabo la verificación se propone el siguiente algoritmo:

a) Obtener el netlist del esquemático original, de tal forma que se obtenga la conectividad orientada a componentes.

b) Obtener el netlist del esquemático generado, en donde se exprese la conectividad orientada a componentes.

c) Crear un árbol balanceado que contenga a los componentes del esquemático original ordenados en base a su handle.

d) Crear un árbol balanceado que contenga a los componentes del esquemático generado ordenados en base al valor de la propiedad PRUEBA.

e) Determinar si la conectividad de ambos árboles es equivalente.

Capítulo 4: CONCLUSIONES

En este último capítulo de la tesis, se presentan dos secciones, en la primera se presentan ideas acerca de las aplicaciones que puede tener este trabajo y la segunda presenta las conclusiones obtenidas.

4.1 FUTUROS TRABAJOS

Entre los trabajos que se pueden desarrollar y que tienen un gran potencial, son:

- Generar herramientas que agreguen de manera automática propiedades comunes a los componentes de un esquemático, por ejemplo para agregar la propiedad INST.

  Un algoritmo apropiado puede ser:

  a) Obtener la lista de interconexiones de un esquemático particular.

  b) Desarrollar un analizador sintáctico que acepte la gramática desarrollada en la sección 3.1.2 y que a cada componente le aumente la propiedad INST.

  c) Emplear al módulo GENSCR para obtener el script que genere al nuevo esquemático.

  d) Emplear el mismo enrutado, ya que este no va a cambiar.

- Hacer que GENES acepte otros tipos de listas de interconexiones, por ejemplo EDIF.

  Para ello lo que se tendría que desarrollar es un analizador sintáctico que a partir de un formato en EDIF se obtenga la lista de interconexiones con la gramática de la sección 3.1.2.

4.2 CONCLUSIONES

En esta tesis se han desarrollado 6 bloques.

En el desarrollo de cada uno de ellos se trató de que fueran utilizables de manera independiente con el objeto de que puedan ser empleadas por otras aplicaciones.

En mi opinión personal considero que cada uno de los bloques puede ser utilizado de manera independiente, para desarrollar nuevas herramientas.

Esta característica se logró apegándose a las especificaciones propuestas al inicio del trabajo.

Y como se planteó en la sección 2.6, es preferible agregar módulos, que modificar las especificaciones originales.

Por otro lado, el módulo más complicado de desarrollar fué el del enrutador.

Esta dificultad se conocia desde el inicio de la tesis, y fue discutida en su momento con el asesor de tesis, a partir de esa discución se decidió realizar el bloque, con las especificaciones propuestas en la sección 2.4.

En lo que se refiere a tiempos de desarrollo, los tiempos propuestos para cada bloque fueron los siguientes:

BLOQUE	TIEMPO ESTIMADO	TIEMPO CONSUMIDO
NETLISTER	1 mes	1 mes
CAMBIADOR	2 meses	2.5 meses
DESCOMPACTADOR	1 mes	2 meses
ENRUTADOR	3 meses	5 meses
GENSCR	2 semanas	2 semanas
GENES	2 semanas	3 semanas

Cabe mencionarse que otro proceso que tomo tiempo fué el de la definición del tema de tesis, en el cual se invirtieron 6 meses.

Tomando en cuenta los tiempos anteriores, el tiempo total empleado para desarrollar esta tesis fué de aproximadamente 2 años.

El tiempo promedio dedicado a desarrollar la tesis fué del orden de 24 horas a la semana.

La utilidad de esta herramienta falta ser determinada, ya que para ello se requiere que la gente que se encarga de desarrollar ASIC's la utilice.

En base a este uso se podrán obtener valiosas ideas de como mejorar al generador de esquemáticos.

Finalmente, considero haber desarrollado un trabajo que presenta una útilidad práctica, y por consiguiente es una aportación al desarrollo tecnológico de nuestro pais.

Apéndice A. CONCEPTOS DE CAPTURA ESQUEMATICA

En este apéndice se presentan los conceptos más relevantes de captura esquemática, con el fin de facilitar la comprensión del desarrollo de este trabajo.

Un Diagrama Esquemático es la representación del diseño de un circuito, en dicha representación se muestran los componentes que conforman al diseño y las conexiones que existen entre los componentes.

1 Captura Esquemática:

Es el proceso de dibujar y almacenar un diagrama esquemático en una computadora, con el fin de poder usarlo posteriormente.

Para llevar a cabo la captura esquemática es necesario contar con una computadora y herramientas de diseño que apoyen a dicha tarea.

Las herramientas de diseño que se mencionan son: Editores de Redes, Editor de Símbolos.

Con el editor de redes se dibujan los diagramas que representan sistemas de hardware. Para realizar tales diagramas, se emplean símbolos y líneas que representan a los alambres de conexión.

Los símbolos representan bloques funcionales o componentes primitivos, posteriormente se explica qué es un bloque funcional y qué es un componente primitivo.

Para dibujar los símbolos que son empleados por el editor de redes, se emplea al editor de símbolos.

Los símbolos pueden representar elementos básicos de diseño, tales como compuertas, transistores, así como también pueden representar diseños completos de Circuitos Integrados de Propósito Específico, o de tarjetas, e incluso pueden representar hojas de esquemáticos.

La captura esquemática no es más que un medio eficiente de dibujar y almacenar diagramas esquemáticos.

APENDICES

Una hoja de esquemáticos es la puerta de entrada a las herramientas de CAE/CAD, siendo posible efectuar simulaciones lógicas de circuitos analógicos o digitales, simulaciones de fallas, diseñar topologías de Circuitos Impresos o de Circuitos Integrados (C.I.), etc.

Con el fin de poder procesar una hoja de esquemático en las áreas de CAE/CAD, es necesario agregar propiedades a los componentes, este agregado de propiedades es realizado mediante los editores de red y de símbolos.

Las propiedades y sus valores varían de acuerdo a la aplicación para la cual vayan a ser empleadas las hojas de esquemáticos.

La función básica de las propiedades es agregar información que no es posible deducir a partir de la hoja de esquemáticos.

El medio ambiente en el cual se desarrolló este trabajo fue empleando las Herramientas de Mentor Graphics (MG), ejecutándose en la plataforma de Estaciones de Trabajo APOLLO, y empleando una minicomputadora GOULD, con el sistema operativo UNIX.

A continuación se presentan los conceptos y las definiciones empleadas por las Herramientas de software de MG.

El Editor de REDES empleado se llama NETED (NETwork EDitor).

Al Editor de simbolos se le denomina SYMED (SYMbol EDitor).

Además de estos programas, MG proporciona un Navegador de diseños (DTR Design Traverser), y el programa EXPAND el cual sirve para extraer la información del diagrama esquemático, que requieren los simuladores.

A continuación se define la terminología que será empleada con los editores de simbolos, de redes y con el DTR, con el fin de evitar confusiones.

La terminología que se dará, es válida para las herramientas de Mentor Graphics, y puede variar para otros sistemas de Captura Esquemática.

Para mayor información acerca del proceso de captura esquemática, consultar las referencias [1], [2], [3], [4], [5] y [6].

2 Términos empleados en NETED

En Neted, un diagrama esquemático representa a todo un circuito electrónico, que puede ser el prototipo de un ASIC, o de una tarjeta de PCB.

Un diagrama esquemático puede constar de una sola hoja de esquemático o de un conjunto de hojas de esquemáticos interconectadas. Esta representación es análoga a los diagramas eléctricos dibujados a mano de un circuito electrónico, como una computadora personal, o el diagrama de un televisor.

En el Caso de Neted el diagrama esquemático, formado por una o más hojas de esquemáticos, representa un diseño.

Cada hoja de esquemático está constituido por 3 objetos gráficos, denominados:

Instancias, Nets y Pines.

Instancia:

Una Instancia representa a un dispositivo electrónico o a una hoja de esquemático.

El hecho de que una hoja de diseño pueda ser representada por un símbolo, facilita la comprensión de las hojas de esquemáticos.

El símbolo representa a la hoja de esquemático y es creado usando SYMED, sin embargo existe otra forma de asignar un símbolo a una hoja de esquemático y es mediante un bloque funcional.

Del párrafo anterior, se desprende que un bloque funcional nos sirve para representar el esquemático de un circuito y hacer más legible una hoja de esquemático para el usuario.

En resumen, las instancias, pueden ser componentes de la biblioteca de partes (componentes primitivas), símbolos de hojas de esquemáticos o bloques funcionales.

Net:

Una net es un alambre o cable (en el caso de buses) de interconexión.

Gráficamente, las nets son segmentos de línea y a los puntos que las delimitan se les denomina vértices.

Pin:

Un pin es el punto donde se conectan las entradas y las salidas de un componente a una net o a otro componente.

Cada Instancia, Net o Pin puede ser manipulado de manera independiente, además a cada uno de ellos se les pueden agregar una o más propiedades.

A continuación se presenta un pequeño esquemático indicando a las instancias, a los pines y a las nets:

FIGURA

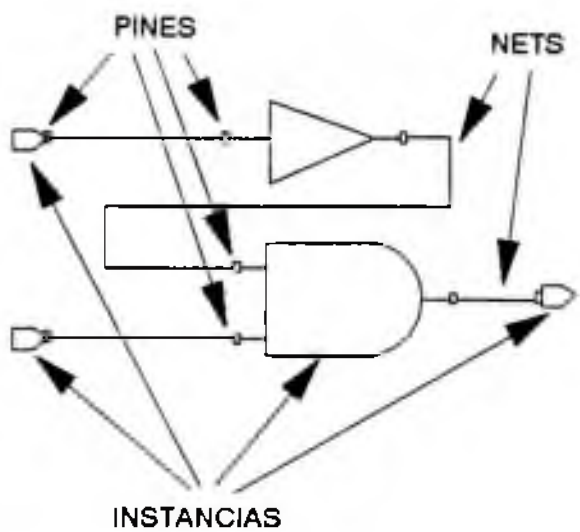


Figura 13.

En la sección 6 de este apéndice se explican ampliamente las propiedades.

Para facilitar el proceso de captura esquemática, se cuenta con tres bibliotecas de partes (gen_lib, misc_lib y spice_lib), en cada una de estas bibliotecas se encuentran simbolos de los componentes y modelos de software (partes de modelo).

A los componentes que se encuentran en las bibliotecas de partes se les conoce con el nombre de partes de la bilioteca o como componentes primitivos.

Creación de Hojas de Esquemáticos empleando a NETED

a) Se colocan los componentes (Instancias) que conforman al diseño en hojas de esquemáticos y se interconectan mediante el uso de nets.

b) Una vez que se tienen conectadas (alambradas) las instancias de una hoja de esquemáticos, por medio de nets, el siguiente paso consiste en agregar las propiedades adecuadas al diseño, el agregado de la información que concierne a las propiedades se realiza agregando TEXTO al esquemático.

c) Una vez finalizada la captura esquemática de un diseño, el siguiente paso es verificar que haya consistencia en las propiedades de las instancias y que no haya nets sin conectar.

La verificación de la hoja de esquemático se lleva a cabo mediante un comando especial de NETED (CHECK SHEET), los errores son reportados mediante mensajes de error o warnings.

Para mayor información acerca de NETED consultar las referencias [1], [2] y [3].

3 Términos empleados en SYMED

APENDICES

El símbolo completo de un componente o de un bloque funcional contiene una imagen gráfica del componente (forma del componente), pins de conexión y propiedades agregadas (ligadas) al símbolo.

Con el propósito de agrupar símbolos se tiene el concepto de FRAMES.

Existen dos tipos de FRAMES:

FRAMES IF
FRAMES FOR

- FRAMES IF

Los FRAMES IF sirven para poder seleccionar versiones de un mismo componente pero que tienen características diferentes, por ejemplo, se pueden tener dos compuertas "Y" de dos entradas, pero una de ellas tiene un abanico de salida de 5 compuertas mientras que la otra tiene un abanico de salida de 10 compuertas, entonces ambas compuertas son colocas dentro de un mismo símbolo, y la elección del símbolo que se desea se lleva a cabo mediante el valor de un parámetro.

A estos símbolos que contienen varias imágenes de un mismo objeto pero que tienen una característica particular (en lo que respecta a la lógica) se les denomina FRAMES IF.

- FRAMES FOR

Un FRAME FOR sirve para crear pines con entradas múltiples.

Los pasos para crear un símbolo son:

a) Dibujar el cuerpo del símbolo (Forma que tendrá el símbolo).
   El cuerpo del símbolo puede ser un simple rectángulo o cualquier forma geométrica formada a base de líneas, arcos y circunferencias.

b) Agregar los pines de conexión.

c) Agregar las propiedades al cuerpo del símbolo, a los pines y a los FRAMES.

d) Verificar que el símbolo no contenga errores en las declaraciones de los FRAMES, que haya consistencia de las propiedades respecto a su dueño, que los parámetros sean usados de manera apropiada.

Uso de parámetros en SYMED

Un parámetro es un valor temporal de una propiedad, que permite al sistema evaluar el valor final de la propiedad, posteriormente.

Los parámetros también permiten el uso de los FRAMES, que son simples componentes del diseño que tienen múltiples representaciones simbólicas.

Cada FRAME es identificado por una propiedad de expresión FRAME (FREXP) y controlada con el parámetro CASE.

Ejemplo:

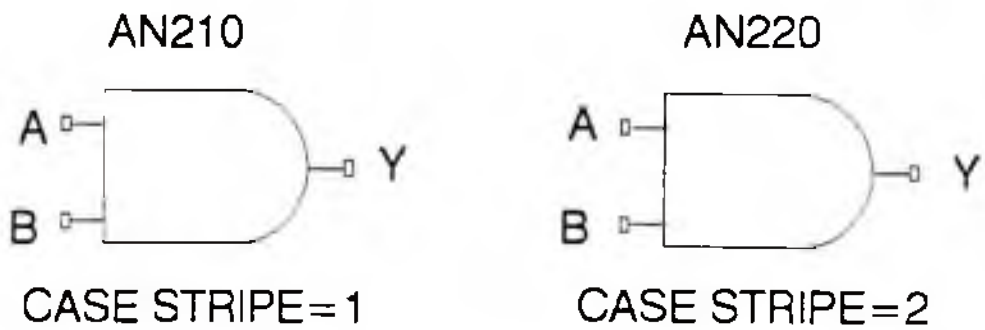


Figura 14

El comando Parameter, tiene un propósito diferente en cada una de los programas NETED, SYMED y EXPAND.

En SYMED el comando parameter es utilizado para identificar FRAMES.

En Neted, se emplea el comando parameter para proporcionar valores fantasma que son usados únicamente para verificar la sintaxis de las expresiones lógicas.

Los parámetros son locales a NETED, esto significa que no son incluidos en los archivos de los esquemáticos cuando se almacena en disco duro el diseño.

En EXPAND, los parámetros pueden ser declarados como parte del proceso de inicialización (set up) de EXPAND.

Mediante el comando parameter, se especifican configuraciones de inicialización especiales.

Para mayor Información acerca de SYMED consultar las referencias [1], [2] y [4].

4 Terminología del Navegante de Diseños (DTR)

Empleando la característica particular de que una hoja de esquemático puede ser representada por un símbolo o por un bloque funcional, es posible realizar diseños jerárquicos, esto es, se puede tener una hoja de esquemáticos con bloques funcionales o con símbolos de hojas de esquemáticos, y en las hojas de esquemático de los bloques funcionales se pueden contener otros símbolos de hojas de esquemático o bloques funcionales.

Por ejemplo sea un diseño que en una primera hoja de esquemáticos A se tienen 3 instancias que corresponden a 3 bloques funcionales B, C y D, ahora bien, el bloque B contiene dos instancias de símbolos de hojas de esquemáticos E y F, mientras que las hojas C y D contienen únicamente componentes primitivas, entonces el diseño se vería como:

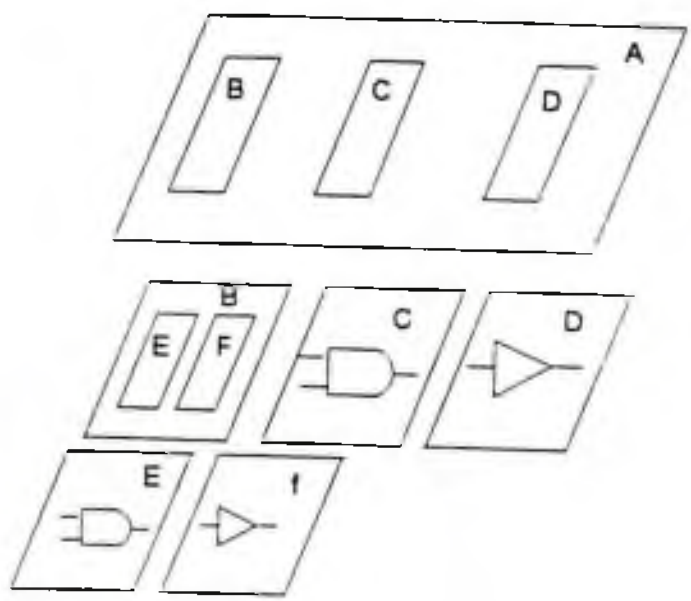


Figura 15.

Se observa que las hojas de esquemático que conforman al diseño forman una jerarquía.

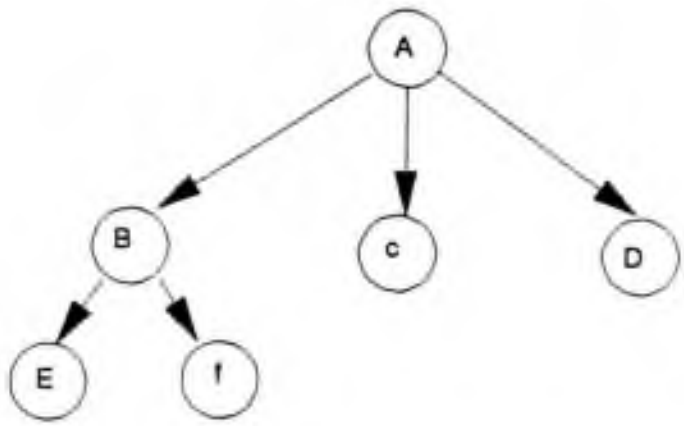
A la hoja A, que contiene a los bloques funcionales B,C y D, se le considera que está en el nivel cero y que corresponde a la hoja con mayor jerarquía.

Las Hojas B,C y D están en el nivel uno y las hojas de esquemático representadas por E y F están en el nivel dos.

Debido a la estructura que se tiene, se dice que este es un DISEÑO MULTINIVEL o JERARQUICO.

La herramienta llamada navegador de diseños (DTR) nos permite analizar la estructura de la jerarquía con el fin de determinar si hay errores en su estructura.

DTR crea un árbol de referencias de componentes, que para el ejemplo que tenemos queda como:



Para mayor información consultar la referencia [5].

5 Biblioteca de Componentes

Anteriormente se mencionó, que para facilitar el proceso de captura esquemática, se cuenta con bibliotecas de componentes.

La compañía Mentor Graphics proporciona tres bibliotecas con componentes primitivos TTL, CMOS, ECL y dispositivos electrónicos analógicos.

Las bibliotecas de componentes se encuentran en los directorios /user/gen_lib, /user/misc_lib y /user/spice_lib.

En cada uno de los directorios de las bibliotecas de componentes existen archivos con la descripción grafica del componente y programas ejecutables (partes de modelo).

Los archivos que describen la forma grafica de los componentes son usados para generar los esquemáticos y los modelos de software son usados por los simuladores para determinar el comportamiento de los componentes bajo las condiciones en las que se encuentre el componente en una simulación lógica.

Los modelos de software son programas ejecutables que describen el funcionamiento de un dispositivo electrónico, como por ejemplo el comportamiento de una compuerta "Y", de un inversor, de una compuerta "O" exclusiva, el de un multivibrador JK, etc..

Las compañías que se dedican a fabricar IC y ASIC's, cuentan con bibliotecas de partes propias, la diferencia de estos componentes respecto a los de MG y de otras compañías fundidoras, radica en las propiedades y en los valores de dichas propiedades que emplean, y en las partes de modelo que son usados en las simulaciones digitales y analógicas.

6 Propiedades

Al iniciar un diseño, se debe saber que tipos de propiedades son necesarias por la aplicación, en donde

será usado la información del diagrama esquemático, i no se asignan las propiedades necesarias, a aplicación no podrá procesar adecuadamente el diseño.

Los conceptos de las propiedades que es necesario manejar son:

- Definición general de las propiedades.

- Posesión de propiedades.

- Diferencia entre el concepto de nombre de propiedad y valor de la propiedad.

- Restricciones para formar el nombre de las propiedades y las restricciones para asignar valores a las propiedades.

  i) Evaluación de expresiones empleadas como valores de las propiedades.

  ii) Ambito de validez en diseños multinivel.

  iii) Handle de un objeto.

  iv) Parámetros.

  v) Propiedades del diseño lógico estructurado (SLD).

  vi) Información de conectividad.

  vii) Información proporcionada por las propiedades.

Un ejemplo sencillo de la aplicación de las propiedades es el siguiente:

En un diagrama esquemático como el siguiente:

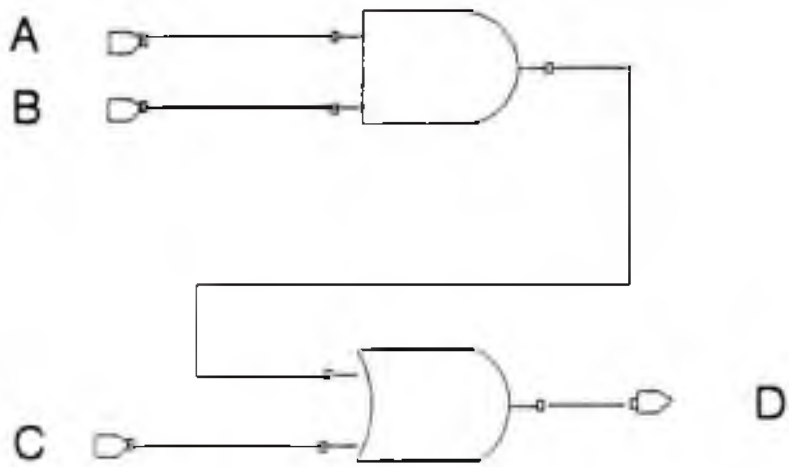


Figura 16.

Se tiene información acerca de la conectividad de los componentes, es decir, del diagrama se puede observar que la salida de la compuerta "Y" se conecta a una de las entradas de la compuerta "O", sin embargo no hay manera de determinar cuál es el tiempo de propagación de la compuerta "Y" o el tiempo de subida del pin de la compuerta O.

Para agregar este tipo de información al diagrama esquemático, se ligan propiedades a los componentes de tal forma que se agrega la propiedad "DELAY" al cuerpo de la compuerta "Y" y se liga a cada pin de entrada de la compuerta "O" las propiedades "RISE" y "FALL".

Con este agregado de propiedades, estamos en condiciones de poder obtener el tiempo que tarda una señal en propagarse a través de la compuerta "Y" y el tiempo que le toma a la compuerta "O" sensar el cambio de estado en las nets que conectan sus pines de entrada.

En general, las propiedades son usadas para agregar información a los diseños que no puede ser obtenida únicamente a partir de las imágenes de los componentes (información gráfica del símbolo que representa al componente) y de las nets que conforman al diseño (diagrama esquemático).

Las propiedades pueden ser asignadas a las instancias, a los pines, a las nets, y a los FRAMES. Cabe hacer la aclaración de que una instancia puede ser un componente primitivo, un bloque funcional o el símbolo de una hoja de esquemático.

La información acerca de las propiedades se añade al diseño esquemático en forma de TEXTO, en formato ASCII y es ligado a los diferentes objetos que van a poseer dicha información.

Para obtener la información adicional que proporcionan las propiedades se emplea al programa de aplicación llamado EXPAND cuya función es aplanar la jerarquía y extraer la información de conectividad y de propiedades que son necesarias para que los programas de aplicación, tales como los simuladores, tenga toda la información que requieren.

Algunas de las propiedades son asignadas de manera automática por NETED y SYMED, sin embargo el usuario puede agregar todas las propiedades que desee a un objeto dado, siempre y cuando se sigan las reglas que se explican posteriormente.

1.6.1 Posesión de propiedades

La posesión de propiedades es un concepto clave para entender el concepto de propiedad.

Dependiendo del tipo de propiedad que un objeto posee, se le puede ligar al objeto dicha propiedad con un valor.

Si un objeto no es dueño de una propiedad particular, no se le podrá ligar un valor al objeto mediante esa propiedad.

La posesión de propiedades de un objeto, depende del tipo de clase de objeto que sea (instancia, net, pin, frame), ya que dependiendo de su clase, algunas propiedades tienen sentido mientras que otras no.

En resumen, antes de asignar un valor correspondiente a una propiedad dada, el objeto debe ser dueño de la propiedad.

Ejemplo:

Los pines poseen propiedades denominadas RISE y FALL que representan las propiedades de tiempo de subida y de bajada.

De tal forma que podemos agregar un valor a la propiedad RISE y ligarla a cualquier pin.

Sin embargo las nets no poseen la propiedad RISE ni la propiedad FALL, siendo imposible dar un valor que corresponde a la propiedad RISE y ligarlo a una net.

En NETED, las siguientes clases de objetos poseen propiedades:

- instancias
- nets
- pines
- FRAMES

En symed sólo tres clases de objetos pueden poseer propiedades:

- Cuerpos de simbolo
- Pines
- FRAMES

En la siguiente tabla se muestran las propiedades que poseen
los objetos en NETED por default.

Objeto	Propiedades que posee por default
net	NET
instancia	INST, CLASS, GLOBAL
pin	PIN, RULE
frame	FREXP

Las propiedades por default que poseen los objetos en SYMED son:

Objeto	Propiedad que posee por default.
cuerpo del simbolo	INST, CLASS, GLOBAL
pin	PIN, RULE, NET
frame	FREXP

1.6.2 Nombre y valor de las propiedades

Todas las propiedades deben tener un nombre y un valor, el nombre sirve para describir a la propiedad mientras que el valor es información que describe las características del diseño.

Tanto el nombre como el valor de las propiedades son almacenadas en forma de cadenas de TEXTO, en formato ASCII.

1.6.3 Restricciones en el nombrado de propiedades

- No se deben usar palabras reservadas como nombre de propiedades.
- El nombre de la propiedad debe iniciar con una letra, y puede ser seguido por cualquier combinación de letras, números o los caracteres de subrayado (_) y signo de pesos ($).
- No se deben emplear los carácteres diagonal (/) o diagonal invertida (\).

- La cadena de carácteres que representa al nombre de una propiedad no debe tener más de 256 carácteres.

1.6.4 Restricciones en los valores de las propiedades

- No se deben usar los carácteres de diagonal invertida (\), ni el de diagonal(/).
- Unicamente en el caso de nets, se permite usar el formato: <letra><$>
- El número de carácteres máximo que debe ser usado para proporcionar el valor de una propiedad debe ser menor a 256 carácteres.

Además, los valores de las propiedades pueden ser definidos mediante el empleo de expresiones lógicas, esto es, combinaciones de variables, valores constantes y operadores lógicos.

El uso de variables en expresiones lógicas empleadas para dar valor a propiedades, son útiles para cambiar el valor de las propiedades de objetos frecuentemente usados, sin tener que reeditar esas partes.

Las expresiones son evaluadas al expander el diseño.

1.6.5 Ambito de los valores de las propiedades

- Los valores asignados a propiedades ligadas a componentes primitivos (componentes de la biblioteca de partes) tienen mayor precedencia que los valores asignados a objetos que están en niveles inferiores en la jerarquía.

- Los valores asignados con el comando Parameter en EXPAND, especifican valores globales para dichas variables.

- Los valores asignados en SYMED a cuerpos de símbolos, especifican valores locales, es decir, especifican el valor de la variable que es usada en todas las hojas debajo de ese símbolo.

1.6.6 Propiedades declaradas en el Diseño Lógico Estructurado (SLD)

Cuando se emplea el software de Mentor Graphics, denominado IDEA Series Aplication, se requieren propiedades particulares, a las cuales se les denomina propiedades del diseño lógico estructurado (SLD). Estas propiedades son indispensables para dar al sistema la información necesaria para ejecutar simulaciones lógicas, analógicas, y diseño de circuitos impresos.

Las propiedades SLD usadas en SYMED/NETED son:

Objeto	Propiedades Declaradas
Instancia	CLASS, GLOBAL, INST
Cuerpo de símbolo	CLASS, GLOBAL, INST
Pines	PIN, RULE, NET
Vértices	PIN, RULE, NET
FRAME	FREXP

Con la definición de estas propiedades, se asegura que las herramientas de MG para diseño de ASIC's y de tarjetas de Circuitos Impresos funcionen apropiadamente, siendo libre el usuario de añadir todas las propiedades que desee, con el fin de hacer más explícito el diseño.

1.6.7 Propiedades asignables a INSTANCIAS

Las propiedades predefinidas por el sistema, que puede poseer una instancia son:

- BLOCK_PATHNAME
- CLASS
- MODEL
- COMP
- INST
- REF

La propiedad BLOCK_PATHNAME tiene como valor la trayectoria del archivo que contiene la información grafica de un símbolo, en el caso de bloques funcionales.

La propiedad CLASS es asignada a símbolos para establecer un propósito diferente al de la función eléctrica, como por ejemplo, conector, global, puerto, bus ripper, conector de página, etc..

La propiedad MODEL es ligada a todos los componentes primitivos de gen_lib, y su valor es la trayectoria del modelo de software del componente al que está ligado. El simulador digital de MG necesita conocer el valor de esta función para saber que modelo de software emplear para simular al componente.

COMP, es una propiedad empleada para dar nombre a símbolos, el programa de aplicación PACKAGE, empleado para diseño de circuitos impresos, utiliza esta propiedad para hacer el mapeo entre componentes lógicos y dispositivos físicos.

INST, es la propiedad que sirve para dar nombres a las instancias.

El valor de la propiedad INST ligada a un objeto debe ser único.

La propiedad REF se usa como una referencia a componentes físicos.

1.6.8 Propiedades asignables a NETS

- NET
- GLOBAL

La propiedad NET se emplea para dar nombre a los alambres de conexión.

En diseños jerárquicos, la propiedad NET asignada a nets, debe corresponder con la propiedad PIN asignada a los pines que conectan las nets.

La propiedad GLOBAL es un método de abreviación para especificar conectividad sin tener que dibujar nets de conexión en el esquemático.

1.6.9 Propiedades Asignables a FRAMES

- FREXP

Esta propiedad permite llevar a cabo la selección o iteración de símbolos empleados en el diseño lógico.

7 Handles de Objetos

Dentro de NETED y SYMED, cada objeto, tiene asociado un handle.

El handle es un identificador asignado por el sistema y su valor es único en todo el diseño.

Es posible asignar (ligar) un nombre especifico a un objeto mediante la propiedad INST, de tal forma que el sistema reconozca ese nombre, sin embargo su handle es retenido.

Un handle se forma con una de las letras I, F, N o V seguida del signo de pesos ($) y un número.

La letra inicial del handle depende del tipo objeto del cual se trate.

Para nombrado de Instancias se emplea el prefijo I, para Nets N, para FRAMES F y para Vértices V.

Apéndice B. CONCEPTO DE LISTA DE INTERCONEXIONES (NETLIST)

La función de las listas de interconexiones (NETLIST) es describir la interconectividad estructural de un Circuito Integrado.

Las descripciones dadas por un NETLIST generalmente no incluye información acerca del comportamiento o rendimiento de un C.I., sin embargo, si proporciona información de las interconexiones de los componentes que conforman al Circuito Integrado.

Es común que cada una de las fundidoras y las casas que se dedican a desarrollar software para diseño de Circuitos Integrados cuenten con un NETLIST propio.

Sin embargo, existe un formato de transferencia de Información de diseños de Circuitos Integrados, denominado EDIF [15],[16].

EDIF es el acrónimo de "Electronic Design Interface Format", como indica su nombre, EDIF es un formato para el intercambio de datos entre sistemas de diseño electrónico.

EDIF sirve para transferir:

- Información de Hojas de Diseño Electrónico.
- Esquemáticos.
- Esquemáticos de Bibliotecas de Símbolos.
- Hojas de Esquemáticos.
- Listas de Interconexiones (Netlists).
- Topología Simbólica de Circuitos Integrados
- Topología de las Máscaras de Circuitos Integrados.
- Topología de las Tarjetas de Circuitos Impresos.

Para el desarrollo de este trabajo, se necesita transferir información acerca del esquemático, esto es, de las Hojas de Diseño Electrónico, y de la lista de Interconexiones.

Dado que EDIF es un formato para transferir mucho más cosas que las que se requieren en este trabajo, el tamaño de una descripción en EDIF es demasiado extensa por lo que se desarrollará un formato para transferencia de información recortado a las necesidades de este trabajo y un netlist apropiado.

Para el desarrollo del formato y del Netlist se emplearán

conceptos desarrollados en EDIF.

Para describir la conectividad entre componentes del diseño de un Circuito Integrado, existen 2 formas:

a) Conectividad orientada a nets
b) Conectividad orientada a partes

Conectividad Orientada a Red:

En este tipo de descripción se da un nombre único a cada una de las nets de interconexión, y se indica a que componente o componentes conecta.

Ejemplo:

Sea el siguiente diagrama esquemático:

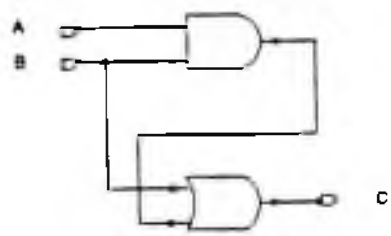


Figura 17.

La conectividad orientada a net se describe como:

```
                N1
A ------------------------------ AND1 I0
                N2
B ------------------------------ AND1 II
                                  OR1  I1
                N3
AND1 Out ----------------------  OR1 I0
                N4
OR1 Out ----------------------- C
```

Conectividad orientada a componente:

En este tipo de conectividad, se indica a qué net, conecta cada uno de los pines de cada uno de los componentes.

Empleando el ejemplo anterior, la conectividad orientada a componete es:

Componente	Pin	Net a la que se conecta el pin
A	A	N1
B	B	N2
C	C	N4
AND1	I0	N1
	I1	N2
	Out	N3
OR1	I0	N3
	I1	N2
	Out	N4

En el caso concreto de este trabajo, se emplea la conectividad orientada a red, por lo que para definir una gramática que se enfoque hacia este tipo de conectividad se debe tener en cuenta que la información que se encuentre en esta, debe ser suficiente para que los módulos del enrutador, del módulo encargado de sustituir a los componentes de Mentor Graphics por los de Texas Instruments, y del módulo encargado de generar el script para crear el nuevo esquemático, operen de manera adecuada.

En el caso del módulo enrutador, se requiere la información del tamaño de los componentes y de la posición de los pines que hay que interconectar.

En el caso del módulo para llevar a cabo la sustitución, se requiere de la trayectoria del componente.

Y en el caso del generador del script, es necesario contar con las propiedades del componente y de las nets.

En base a estas caracteristicas, se desarrollaron las gramáticas usadas para verificar que la información obtenida es la adecuada.

Por lo tanto, la gramática deberá contener la información siguiente:

a) Tamaño de la hoja de diseño.

b) Espaciamiento entre pines de los componentes.

c) Unidades en las cuales se darán las medidas.

d) Lista con los componentes que conforman a los componentes.

e) Lista con las Nets.

La información de cada uno de los elementos de la lista de componentes deberá ser:

a) Trayectoria.

b) Handle.

c) Posición dentro de la hoja de diseño.

d) Centro de Rotación.

e) Tamaño.

f) Orientación.

g) Si el componente esta reflejado respecto a un eje de coordenadas especifico.

h) Lista de propiedades.

i) Lista de pines.

En donde, cada elemento de la lista de propiedades contiene la siguiente información:

a) Nombre.

b) Valor.

c) Tipo.

d) Visibilidad.

e) Rotación.

f) Tamaño.

g) Justificación.

h) Posición.

i) Desplazamiento respecto al origen del objeto al cual pertenece la net.

En el caso de la lista de pines de cada elemento, dado que para definir un pin, se hace añadiendo la propiedad pin a los vértices del componente, la información que deberá proporcionarse, es igual a la de las propiedades, pero conviene tenerla separada de la lista de propiedades, aunque su información es igual a la de un elemento de la lista de propiedades.

Finalmente, para la lista de Nets, en esta gramática, se empleará la conectividad de redes, así que la información de cada elemento de la lista es:

a) Nombre de la NET.

b) Lista con los pines que conforman a la net.

Cada elemento de la lista de pines proporciona la siguiente información:

a) Nombre del pin

b) Posición dentro de la red

c) Nombre del componente al cual pertenece el pin.

Apéndice C. TIPOS DE ENRUTADORES

En este capítulo se dará una explicación somera de estrategias de enrutado comunes.

Los algoritmos de enrutado más conocidos, pueden agruparse de acuerdo a la estrategia empleada para llevar a cabo el enrutamiento.

1 Estrategias de Enrutado

- Conflicto Cíclico
- Dogleg
- Segmento en la dirección incorrecta
- Frente de Onda
- Línea de escape, punto de escape

A continuación se proporciona una breve descripción de cada una de estas estrategias, en caso de que el lector desee mayor información se recomienda consultar la referencia [7].

3.1.1 Conflicto Cíclico

Para explicar en que consiste el conflicto ciclico es necesario definir lo que se entiende por Gráfica Dirigida en el área de enrutadores.

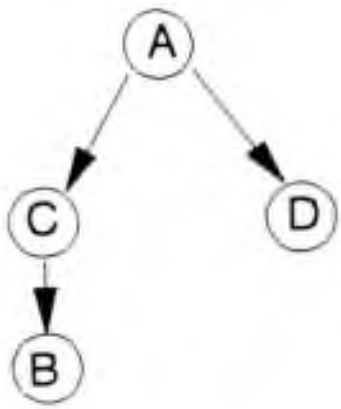
Una gráfica dirigida es una manera de representar los requerimentos de que un enrutado determinado quede colocado encima o debajo de otro alambrado.

Por ejemplo:

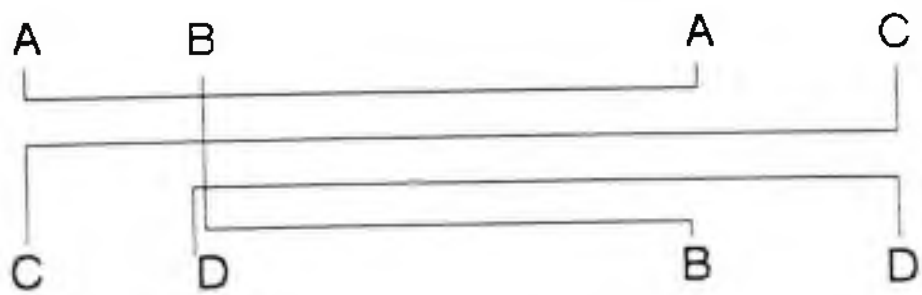
Se tienen los siguientes puntos que se desea enrutar (el alambrado debe enrutar a los puntos que tienen la misma literal).

A	B	A	C
C	D	B	D

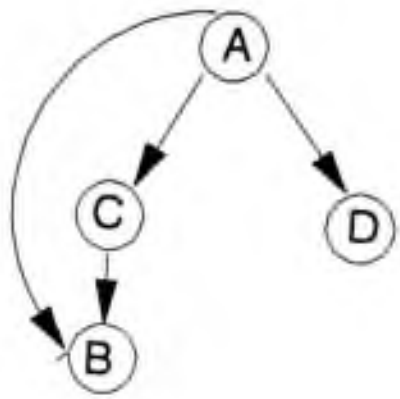
El enrutado debe cumplir con la siguiente gráfica dirigida en el alambrado realizado en dirección horizontal.



Un enrutado aceptable es:



Un conflicto cíclico surge cuando dos o más alambrados deben quedar arriba y abajo de un alambrado al mismo tiempo; esta situación se representa en la gráfica ciclica como un lazo.



3.1.2 DOGLEG

El término Dogleg se refiere al uso de segmentos horizontales por net a una bahia de alambrado con el fin de reducir el número de canales de alambrado necesarios para resolver circuitos cíclicos.

3.1.3 Segmento en Dirección Incorrecta

Para una tecnologia de interconexión metálica de 2 niveles, sí un nivel se dedica para segmentos horizontales, entonces el otro para segmentos verticales.

Con la estructura anterior, la opción que se tiene para implementar un enrutado dogleg es la de emplear un nivel de interconexión con un segmento en la dirección contraria para poder proseguir la interconexión.

Esta técnica tiene la desventaja de que se bloquean una cantidad considerable de canales adyacentes en la dirección principal por el segmento en dirección

contraria. Sin embargo, proporciona la ventaja de no necesitar 2 canales de polarización entre niveles de enrutado.

3.1.4 Frente de Onda

En lugar de atacar el problema de alambrar completamente una bahía de alambrado a la vez, existen algoritmos denominados de enrutamiento global, los cuales intentan encontrar la interconexión para una pareja de puntos a la vez.

Para encontrar la interconexión de un punto origen al punto destino, existen algoritmos globales conocidos como "maze-runners", que inician un frente de onda que se expande a partir del punto origen, dicha expansión continua hasta alcanzar al punto destino.

Los algoritmos maze-runners tienen la ventaja de encontrar una ruta de enrutado si existe algún camino.

Sus desventajas primordiales son:

- Requerir grandes cantidades de memoria.
- El tiempo de CPU necesario para propagar al frente de onda a través de la malla de alambrado, es considerable.

3.1.5 Línea de Escape, Punto de Escape

Otra clase de algoritmos de alambrado global intentan minimizar la cantidad de recursos empleados por los algoritmos maze-runners, reemplazando al frente de onda con una lista de segmentos de líneas de interconexión potenciales entre el punto origen y el punto destino.

El algoritmo de Búsqueda de líneas potenciales de enrutado emplea el concepto de Linea de escape y punto de escape.

En este algoritmo, únicamente se almacena información de segmentos de línea, información de la orientación de la

línea que se investiga e información de las líneas que no deben ser intersectadas, trayendo como consecuencia un uso eficiente del almacenamiento en memoria.

2 Enrutadores MAZE-RUNNERS

El algoritmo más conocido dentro de esta categoría es el de Lee [7], en base a este algoritmo se han realizado una gran cantidad de variaciones con el fin de optimizar el tiempo de ejecución.

El algoritmo de Lee asume que el espacio de enrutado del Chip puede ser representado mediante un arreglo bidimensional y que se puede establecer una relación entre una posición en la malla de enrutado y el arreglo, además de que se puede asignar un valor numérico a cada uno de los elementos del arreglo.

Básicamente, el algoritmo de LEE se puede resumir en los siguientes pasos:

a) Ordenar los puntos a enrutar.

b) Inicializar el arreglo que representa a la malla del alambrado.

c) Seleccionar el par de puntos a ser enrutados.

d) Iniciar el frente de onda y expandirlo hasta determinar si es posible o no enrutar la pareja de puntos dados.

e) Si existen más puntos para ser enrutados, seleccionar la siguiente pareja de puntos a ser enrutados y repetir el inciso d).

El ordenamiento de los puntos a ser enrutados cada vez se complica más y más, debido a que cada vez que se enruta un par de puntos se cierran trayectorias para los demás puntos, por lo que los criterios de ordenamiento son factores importantes para poder alambrar completamente un circuito.

Las características que se consideran en el desarrollo de estos criterios son:

- Longitud estimada del alambrado

- Tamaño mínimo de un rectángulo de aislamiento
- Número de puntos de enrutado localizados dentro del rectángulo de aislamiento, etc.

Para obtener resultados óptimos, se sugiere alambrar las trayectorias más cortas primero y que las trayectorias con rectángulos de aislamiento grandes reciban una prioridad menor.

Para inicializar el arreglo bidimensional de cada uno de los planos de interconexión, las trayectorias bloqueadas deben ser designadas de manera consistente para no tomarlos en cuenta a medida que los segmentos de alambrado son finalizados, además los canales ocupados también deben ser marcados como no disponibles.

La inicialización del frente de onda se lleva a cabo de la siguiente manera:

Una vez que se tiene el punto origen, se determina la disponibilidad de cada una de las localidades vecinas disponibles (Superior, Inferior, Izquierda y Derecha) a cada uno de los puntos vecinos disponibles,se les asigna un costo de expansión del frente de onda y el valor se guarda en el arreglo bidimensional.

Los costos de expansión se determinan en base a la dirección que lleva el enrutado y la distancia que hay de ese punto al punto destino.

En base a los valores de expansión de los puntos vecinos con disponibilidad, se crea una lista inicial de frontera, esto significa que cada uno de los puntos de la lista representa un punto potencial de expansión.

Se elige el punto que tenga un costo mínimo y a partir de este punto se sigue expandiendo el frente de onda hasta encontar al punto destino o hasta que la lista de posibles puntos de expansión esté vacía, lo cual implica que no hay modo de enrutar a los puntos.

Una vez que se ha encontrado una ruta se realiza un "back-trace" hacia el punto origen a través de los costos del frente de onda con el fin de determinar segmentos que

conformen al enrutado.

3 Algoritmo de Hightower

El algoritmo de enrutado de Hightower [11] pertenece al grupo de enrutadores globales, su principio de operación se basa en el algoritmo de Lee [7], con la diferencia de sustituir el frente de onda, característica de los maze-runners, por listas de segmentos de líneas que representan segmentos potenciales para realizar el enrutado.

El algoritmo utiliza al plano continuo como representación de la malla de alambrado, por lo que la única restricción para el tamaño y el espaciamiento entre las redes es la precisión de la computadora para representar al tipo de dato (entero, flotante o de doble longitud) empleados para representar a los segmentos de los enrutados.

Los alambrados obtenidos cumplen con la geometría de Manhattan y con la restricción de no cruzar áreas restringidas ni de pasar por vértices de otros alambrados.

Para entender al algoritmo que se presenta, el cual es una versión con pocas modificaciones del algoritmo de Hightower [11] se presentan las siguientes definiciones:

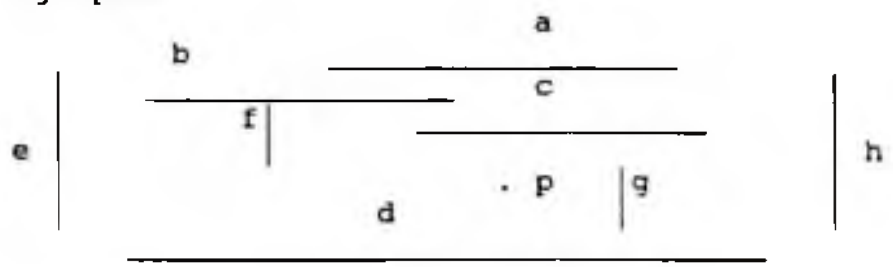
Geometría Manhattan:

> Los segmentos de un alambrado deben ser perpendiculares.

Cubrir un punto:

Un segmento de Linea l Cubre a un punto dado p, si la línea perpendicular que va del punto a la línea de la cual l es un segmento, intersecta a l.

Ejemplo:



Las lineas horizontales a,c y d y las lineas verticales e, g, h cubren al punto p.

Restriciones Horizontales (C_H)

Las Restricciones Horizontales son una lista de segmentos de linea horizontales que son los bordes de las áreas que no deben ser atravesadas.

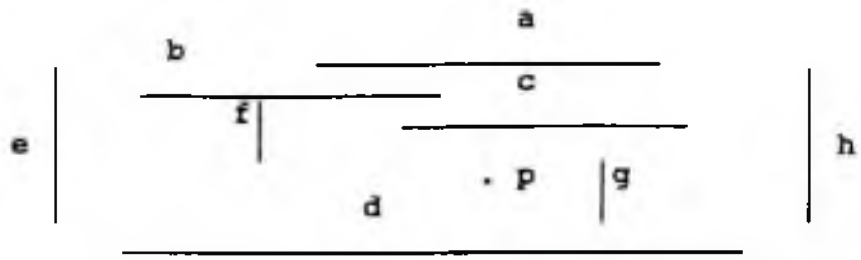
Restriciones Verticales (C_V)

Las Restricciones Verticales son una lista de segmentos de linea verticales que son los bordes de las áreas que no deben ser atravesadas.

Cobertura vertical

La cobertura vertical de un punto son los elementos de C_V tales que cubran al punto p y no haya ningún otro elemento de C_V que también cubra al punto y esté colocado entre el punto y el elemento que se cree que pertenece a la cobertura.

Ejemplo:



Las líneas verticales e y g constituyen la cobertura vertical del punto p.

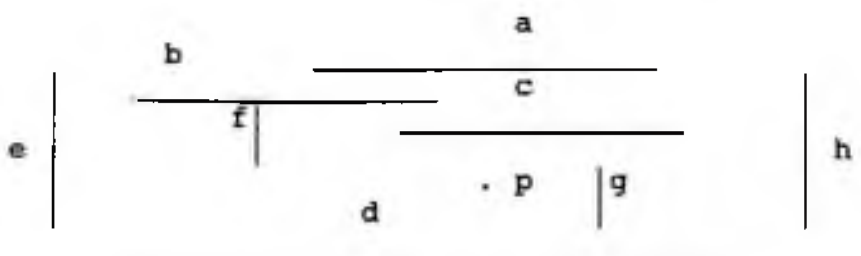
Como puede observarse, la cobertura vertical de un punto está constituida por dos elementos de C_V, uno colocado a su izquierda y el otro colocado a su derecha.

Cobertura Horizontal:

La cobertura horizontal de un punto son los elementos de C_H tales que cubran al punto p y no haya ningun otro elemento de C_H que cubra al punto y este colocado entre el punto y el elemento que se cree que pertenece a la cobertura.

La cobertura horizontal de un punto está constituida por dos elementos de C_H, uno colocado encima y el otro colocado debajo del punto.

Ejemplo:

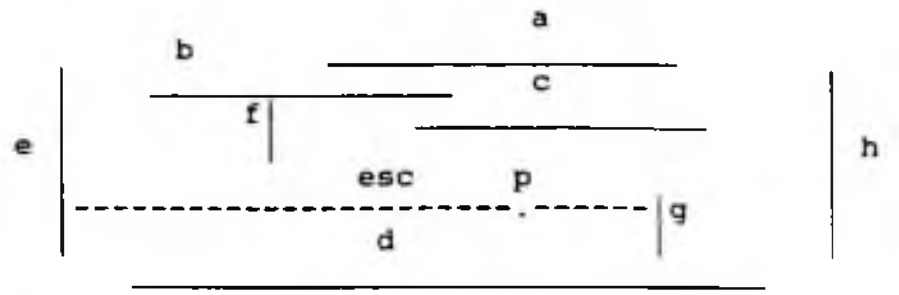


Las líneas horizontales c y d constituyen la cobertura horizontal del punto p.

Línea de escape horizontal (L_{EH})

La L_{EH} es el segmento de línea horizontal que pasa por el punto p y que está acotado por la cobertura vertical del mismo punto p.

Ejemplo:

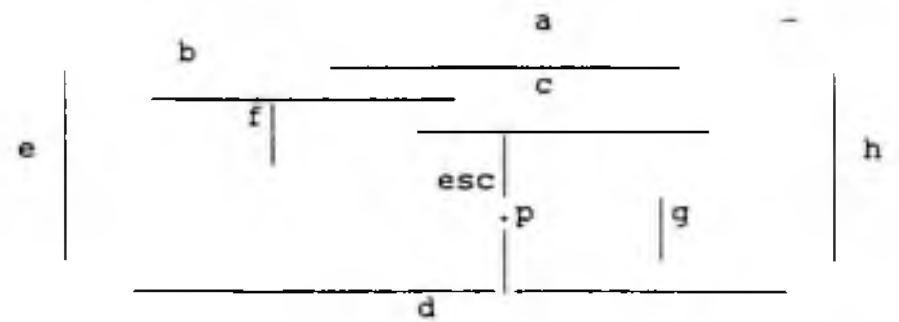


La línea esc es la línea de escape horizontal del punto p.

Línea de escape vertical (L_{EV})

La L_{EV} es el segmento de línea vertical que pasa por el punto p y que está acotado por las cobertura horizontal del punto p.

Ejemplo:

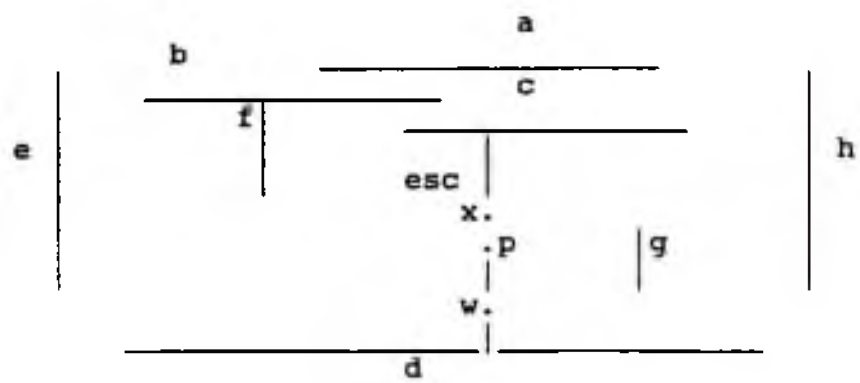


La línea esc es la línea de escape vertical del punto p.

Punto de escape horizontal

El punto de escape horizontal es un punto que pertenece a la línea de escape vertical del punto p y que no es cubierto por alguno de los 2 segmentos que constituyen a la cobertura horizontal de p.

Ejemplo:

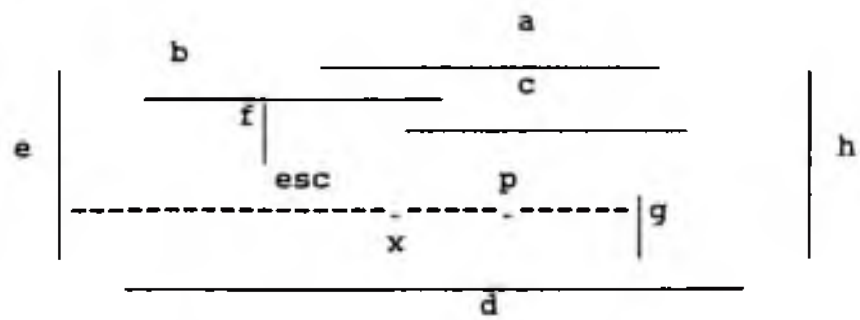


Los puntos x, w son puntos de escape horizontales.

Punto de escape vertical

El punto de escape vertical es un punto que pertenece a la línea de escape horizontal del punto p y que no es cubierto por alguno de los 2 segmentos que constituyen a la cobertura vertical de p.

Ejemplo:



El punto x es un punto de escape vertical.

Unidad:

Es la mínima distancia disponible entre dos líneas de segmento. En otras palabras, es el espacio entre los elementos de la malla de enrutado.

Punto Origen:

Es el punto de escape que está siendo procesado.

Punto Destino:

Es el punto que se desea alcanzar.

Listas simplemente ligadas empleadas

Path:

Lista de puntos que determinan un enrutado.

L_{HO}

Lista de segmentos de línea horizontales, asociados con el punto origen, y que se encuentran ordenados respecto a las ordenadas.

L_{VO}

Lista de segmentos de línea verticales asociados con el punto origen y que se encuentran ordenados respecto a las abscisas.

L_{EO}

Listas de puntos de escape asociados con el punto origen.

L_{HT}

Lista de segmentos de línea horizontales, asociados con el punto destino, y que se encuentran ordenados rspecto a las ordenadas.

L_{VT}

Lista de segmentos de línea verticales asociados con el punto destino y que se encuentran ordenados respecto a las abscisas.

L_{ET}

Listas de puntos de escape asociados con el punto destino.

El algoritmo de enrutado es el siguiente:

Dar las coordenadas de la primer pareja de puntos a ser enrutados (punto A y punto B).

Paso 1 Meter al punto A en la lista L_{EA} y al punto B en la lista L_{EB}.

Paso 2 Verificar que la bandera de no_escape asociada con el punto A, si esta encendida, verificar la bandera de no_escape asociada con el punto B, si está encendida el algoritmo no ha sido capaz de encontrar una trayectoria de enrutado, enviar un mensaje de error y seguir con el paso 5.

Si la bandera del punto B no está encendida seguir con el punto 3.

Si la bandera del punto B no está encendida hacer que el último punto de la lista L_{EA} sea el punto origen y que el punto B sea el punto destino.

Aplicar el proceso de escape.

Si la bandera de intersección está encendida continuar con el punto 4 en caso contrario seguir con el paso 3.

Paso 3 Intercambiar los puntos origen y destino, aplicar el paso 2.

Paso 4 Aplicar el algoritmo de refinamiento de trayectorias

Paso 5 Obtener la siguiente pareja de puntos a ser ordenados (A,B) y aplicar el paso 1.

En caso de que no haya más puntos a ser enrutados terminar el programa.

Proceso principal de escape:

Paso 1 Examinar la bandera de orientación y construir la línea de escape correspondiente que pase por el punto objeto.

Paso 2 Hacer que x tenga como valor el complemento de la bandera de dirección.

Verificar si la nueva linea de escape intersecta alguno de las lineas de la lista L_{xT} , en caso afirmativo hacer que la bandera de intersección tenga un valor verdadero continuar con el paso 5, en caso contrario continuar con el paso 3.

Paso 3 Intentar encontrar un punto de escape empleando el proceso de escape I.

Si no se encuentra un punto de escape aplicar el proceso de escape II, en caso de falla, hacer que el valor de la bandera de no_escape sea verdadero y terminar el proceso de escape, en caso contrario seguir con el paso 4.

Paso 4 Meter la línea de escape en L_{OE} Terminar el proceso.

Paso 5 Almacenar el punto de intersección en las variables x,w y terminar el proceso de escape.

Proceso de Escape I

Sea Z el punto Objeto.

Sean f1, f2, f3, f4 las extremidades de la cobertura (horizontal o vertical, según el caso) y sea d la función distancia euclideana de 2 puntos, definida como:

$$d[(x1,y1),(x2,y2)] = sqrt((x1-x2)sup2+(y1-y2)sup2)$$

Asociar con la tupla (f_1,f_2,f_3,f_4) la tupla $(f_{i1},f_{i2},f_{i3},f_{i4})$ de tal forma que se cumpla:

$$d(f_{im},Z) <= d(f_{in},Z),\ m < n$$

Algoritmo del proceso de escape I

Paso 1 Asocie el vector $(f_{i1},f_{i2},f_{i3},f_{i4})$ con la cobertura horizontal.

Haga m=1 y tome el punto cuya abscisa se encuentra una unidad alejada de f_{im} y cuya ordenada es la misma que la del punto Z, verifique si este punto es un punto de escape que esta localizado sobre una línea de escape vertical no utilizada.

Si no es un punto de escape incremente una unidad el valor de m hasta encontrar un punto de escape o hasta que m sea igual a 5.

En caso de encontrar un punto de escape hacer que el valor de la bandera de orientación tenga un valor de "VERTICAL" dando por terminado el proceso de escape I.

Si m = 5 continue con el paso 2.

Paso 2 Asocie al vector $(f_{i1},f_{i2},f_{i3},f_{i4})$ con la cobertura vertical.

Haga que m tenga un valor de 1, tome el punto que se encuentre localizado una unidad lejos de la ordenada f_{im} y cuya abscisa sea igual a la de Z.

Verifique si el punto obtenido es un punto de escape que se encuentra sobre una línea de escape horizontal no usada.

Si no es un punto de escape continue incremenetando el valor de m hasta encontrar un punto de escape o hasta que m tenga un valor de 5.

En caso de encontrar un punto de escape haga que la bandera de orientación tenga un valor de "HORIZONTAL" y termine el proceso de escape I.

En caso contrario, no hay escape empleando el proceso I.

Proceso de Escape II.

Se dice que un punto es un punto de escape II si este tiene un punto de Escape I en la línea de escape horizontal o vertical que pasen por dicho punto.

Algoritmo del proceso de escape II.

Sean r_1 y r_3 las intersecciones de la línea de escape vertical que pasa por el punto p y la cobertura horizontal del punto p.

Sean r_2 y r_4 las intersecciones de la línea de escape horizontal que pasa por el punto p y la cobertura vertical del punto p.

Empezando con i=1 haga que r_i se aproxime una unidad a Z, construya la línea de escape apropiada a r_i.

Si el valor de la bandera de intersección es verdadero, meter ri en la lista L_{OE} y terminar el proceso de escape II.

En caso de que la bandera tenga un valor de Falso, intentar encontrar un punto de escape I en la línea de escape empleando el proceso de escape I.

Si no se puede encontrar un punto de escape, incrementar el valor de i con la secuencia (1,2,3,4).

Algoritmo de optimización de la trayectoria.

Cuando se ha encontrado una trayectoria de enrutado, se cuenta con una lista de puntos de la forma:

Path = (A,p1,p2,....,pn) en donde cada una de las pi es un punto de escape asociados con el punto A.

Además se cuenta con un punto de intersección I y con una linea k que pasa por el punto de intersección

La finalidad del proceso de optimización consiste en aislar los puntos que son esquinas de la trayectoria y desechar los puntos superfluos.

La idea del algoritmo de refinamiento es el siguiente:

a) Crear una lista L.

b) Sea I(x,y) el punto de intersección encontrado en el paso dos del algoritmo de escape.

   Meter I en una lista denominada L.

   Sea k la linea que pasa por I y por pn.

c) Si Path está vacia terminar el algoritmo.

d) Recorre la lista Path en orden inverso hasta encontrar un punto Px que esté sobre la linea k que pasa por I. Al encontrar este punto se borran los pi tales que i > x.

   Meter a px en la lista L.
   Hacer que I sea igual a px.

   Encontar la linea de escape que pasa por px
   y que es perpendicular a k.

   Hacer que k sea igual a esta nueva linea.

   Borrar Px y los puntos que le siguen de Path.
   Continuar en el inciso c.

4 Pseudocódigo del algoritmo de Hightower

Algoritmo de Enrutado empleando procedimientos de escape.

Procedimiento de enrutado.

```
Begin
   Inicializa_variables;
   Inicializa_listas;
   Lea <- A;
   Leb <- B;
   Repeat
        If(Esc_from_A)
         Begin
           Origen <- Ultimo elemento de Lea;
           Destino <- B;
           Escape(Esc_From_A, Int_Flag, Orien_A, Path);
           If(Int_Flag)
            Begin
              Mejora_trayectoria(Path);
              Return TRUE;
            End
        If(Esc_from_B)
         Begin
           Origen <- Ultimo elemento de Leb;
           Destino <- A;
           Escape(Esc_From_B, Int_Flag, Orien_B, Path);
           If(Int_Flag)
            Begin
              Mejora_trayectoria(Path);
              Return TRUE;
            End
         End
       If(  NOT(Esc_From_A) AND NOT(Esc_From_B) )
         Begin
            break;
         End
   Until (TRUE);
   destruye_listas();
   Return TRUE;
End.
```

Procedimiento principal de Escape

```
Begin
  Case Orientación
    Begin
      Ambas_dir:
                  Obten_línea_Esc_hor(línea_Escape,punto);
                  Lho <- línea_Escape;
                  If(Intersecta_línea_hor_lista_líneas_ver
                                          (Lhv,línea_Escape))
                      Begin
                          *Int_Flag <- TRUE;
                           Return;
                      End;
                  If(Línea_en_frontera(línea_Escape))
                       Begin
                          orientación <- VER;
                       End;

                  Obten_línea_Esc_ver(línea_Escape,punto);
                  Lvo <- línea_Escape;
                  If(Intersecta_línea_hor_lista_líneas_ver
                                          (Lht,línea_Escape))
                      Begin
                          *Int_Flag <- TRUE;
                           Return;
                      End;
                  If(Línea_en_frontera(línea_Escape))
                       Begin
                          If( orientación = VER )
                             Begin
                                *Esc_Flag <- FALSE;
                                Return;
                             End;
                       End;
      Horizontal:

                  Obten_línea_Esc_hor(línea_Escape,punto);
                  Lho <- línea_Escape;
                  If(Intersecta_línea_hor_lista_líneas_ver
                                          (Lvt,línea_Escape))
                      Begin
                          *Int_Flag <- TRUE;
                           Return;
                      End;
      Vertical:
```

```
                    Obten_línea_Esc_ver(línea_Escape,punto);
                    Lvo <- línea_Escape;
                    If(Intersecta_línea_hor_lista_líneas_ver
                                        (Lht,línea_Escape))
                        Begin
                            *Int_Flag <- TRUE;
                             Return;
                        End;
    End;

    escape_uno(orien,Esc_Flag,pt_Escape);

     If(Esc_Flag)
        Begin
          Leo <- pt_Escape);
          Return;
        End;
   escape_dos(orien,Esc_Flag,pt_Escape);

   If(Esc_Flag)
      Begin
         Leo <- pt_Escape);
         Return;
      End
   Else
      Begin
        Esc_Flag <- FALSE;
         Return;
      End;
End.
```

Procedimiento de Escape Uno

```
Begin
    if(orien = AMBAS_DIR)
      {
          orientación <- HOR;
          Esc_uno(Esc_Flag,pt_Escape);
          if(Esc_Flag)
               {
                    orientación <- VER;
                    Esc_uno(Esc_Flag,pt_Escape);
               }
          if(Esc_flag = FALSE)
               orien <- AMBAS_DIR;
      }
     else
```

```
          {
           Esc_uno(Esc_Flag,pt_Escape);
          }

End;

Procedimiento ESC_UNO

Begin
          Obten_puntos_de_la_cobertura(f1,f2,f3,f4);
          j <- 1;
          Repeat

Obten_posible_punto_de_Escape(Esc_pt,Esc_aux,fj);
               If( Es_punto_de Escape(Esc_pt) )
                 Begin
                    *Esc_Flag <- TRUE;
                 End;

               If(Punto_de_Escape_usado(Esc_pt_used,Esc_pt))
                  Esc_Flag <- FALSE;
               crea_linea_Escape(Esc_line,Esc_pt,Z);

If(cruza_linea_lista_lineas(Esc_Line_used,Esc_line))
                   Esc_Flag <- FALSE;
               crea_linea_Escape(Esc_line,Esc_pt,Esc_aux);

If(cruza_linea_lista_lineas(Esc_Line_used,Esc_line))
                   Esc_Flag <- FALSE;

               If(Esc_Flag)
                  Return;

                j <- j+1;
          Until( j < 5 );
    Esc_Flag <- FALSE;
End.

Procedimiento de Escape dos

Begin
    if(orien = AMBAS_DIR)
      {
        orientación <- HOR;
         Esc_dos(Esc_Flag,pt_Escape);
        if(Esc_Flag)
          {
```

```
                orien <- VER;
                 Escape_dos(Esc_Flag,pt_Escape);
            )
        }
      else
        (
            Esc_dos(Esc_Flag,pt_Escape);
        )

End;

Procedimiento Esc_Dos

Begin
   Obten_pts_intersección_cobertura(Z,r1,r2,r3,r4);
   i <- 1;
  If(orientación = HOR)
      Begin
         r1 <- Z;
         r3 <- Z;
      End
  Else
      Begin
         r2 <- Z;
         r4 <- Z;
      End;
  Repeat
         Case (i)
           Begin
              1:
                 res_comparacion <- compara_pt(r1,Z);
              2:
                 res_comparacion <- compara_pt(r2,Z);
              3:
                 res_comparacion <- compara_pt(r3,Z);
              4:
                 res_comparacion <- compara_pt(r4,Z);
           End;
         If(res_comparacion)
           Begin
               If(compara(Z,r1,r2,r3,r4) )
                   Begin
                        Esc_Flag <- FALSE;
                        Return;
                   End;
               i <- (i+1) mod 5;
               If(i = 0)
```

```
                i <- 1;
          End;
        Else
          Begin
             Obten_punto_r(Z,ri,r);
            Obten_linea_Escape(Esc_line,r);
              If(intersecta_linea_lista_lineas(Lvt,Esc_line)
);
               int_Flag <- TRUE;
              If(intersecta_linea_lista_lineas(Lht,Esc_line)
);
               int_Flag <- TRUE;
            If(int_Flag)
              Begin
                Leo <- ri;
                 Return;
              End;
            If(orien_Flag = Ambas_dir)
              Begin
                orien_Flag <- HOR;
                Escape_dos_uno(Z,r,Esc_Flag,orien_Flag);
                If Not (Esc_Flag)
                  Begin
                   Orien_Flag <- VER;

Escape_dos_uno(Z,r,Esc_Flag,orien_Flag);
                   End;
              End;
            Else
              Escape_dos_uno(Z,r,Esc_Flag,orien_Flag);
            If(Esc_Flag)
              Begin
               Leo <- r;
               Return;
              End;
            Else
              Begin
                  i <- (i+1)mod5;
                 If(i = 0)
                      i <- 1;
              End;
        End;

  Until(TRUE);
End.
```

Procedimiento de Escape Dos_Uno

```
Begin
   Obten_pts_interseccion_cobertura(Z,r1,r2,r3,r4);
    Obten_línea_esc(pt,Esc_line);
    If(intersecta_línea_lista_líneas(Lvt,Esc_line) )
      Begin
          Leo <- pt;
          Esc_Flag <- TRUE;
          Int_Flag <- TRUE;
          Return;
      End
    If(intersecta_línea_lista_líneas(Lht,Esc_line) )
      Begin
          Leo <- pt;
          Esc_Flag <- TRUE;
          Int_Flag <- TRUE;
          Return;
      End
   If(orientación <- VER)
      Begin
         For i<-r2.x to r4.x
           Begin
              r.x <- i;
              r.y <- pt.y;
              crea_línea(seg,r,pt)
              If Not (intersecta_línea_lista(ELU,seg)
                Begin
                    Escape_dos_dos(pt,r,Esc_Flag);
                    If(Esc_Flag)
                      Begin
                         Leo <- pt;
                         Return;
                      End;
                End;
           End;
      End

   Else If(orientación = HOR)
      Begin
         For i<-r1.y to r3.y
           Begin
              r.x <- pt.x;
              r.y <- i;
              crea_línea(seg,r,pt)
              If Not (intersecta_línea_lista(ELU,seg)
                Begin
                    Escape_dos_dos(pt,r,Esc_Flag);
                    If(Esc_Flag)
                      Begin
```

```
                        Leo <- pt;
                        Return;
                    End;
                End;
            End;
        End;

End.
```

Procedimiento de Escape Dos_Dos

```
Begin
        Orien_cober  <- orientación;
         Obten_pts_cobertura(Z,f1,f2,f3,f4);
         j <- 1;
         Repeat
                Case(j)
                   Begin
                      1:  pt <- f1;
                      2:  pt <- f2;
                      3:  pt <- f3;
                      4:  pt <- f4;
                   End;

Obten_punto_escape(Z,esc_pt,esc_aux,pt,orientación);
               If(Punto_de escape_usado(EPU,esc_pt))
                  Begin
                      Esc_Flag <- FALSE;
                      Next;
                  End;
               crea_linea(seg,Z,esc_pt);
               If(Intersecta_linea_lista(ELU,seg))
                  Begin
                      Esc_Flag <- FALSE;
                      Next;
                  End;
               crea_linea(seg,esc_pt,esc_aux);
               if(Intersecta_linea_lista(ELU,seg))
                  Begin
                      Esc_Flag <- FALSE;
                      Next;
                  End;
               If(Es_punto_de_escape(esc_pt))
                  Begin
                       Esc_Flag <- TRUE;
                       Return;
                  End;
```

```
                j <- j+1;
            Until (j<=4);
End.
```

Apéndice D. ORGANIZACION DEL DISKETTE CON EL CODIGO FUENTE

Para facilitar la distribución de los programas desarrollados en esta tesis, se proporciona el texto de la tesis y los programas fuentes en un diskette de 3.5 pulgadas, en el formato del Sistema Operativo DOS.

La organización del diskette es como sigue:

En el directorio raíz se encuentran 2 directorios:

```
\TEXTO
\CODIGO
```

1 Subdirectorio \TEXTO

El texto de la presente Tesis se encuentra en este directorio, en dos formatos. El primero se encuentra en el formato del procesador de textos MANUSCRIPT y el segundo en formato ASCII.

Las figuras que se presentan, fueron desarrollados empleando el programa FREELANCE.

La información en el formato de MANUSCRIPT se encuentra en 3 archivos denominados:

```
\TEXTO\INTRO.DOC
\TEXTO\TESIS.DOC
\TEXTO\BIBLI.DOC
\TEXTO\APEN.DOC
```

El archivo INTRO contiene la introducción de la tesis, mientras que el archivo TESIS contiene el texto de los 3 capítulos que conforman a la tesis, en el archivo BIBLI se encuentran las referencias bibliográficas y finalmente, en el archivo APEN se tiene el texto de los apéndices que conforman a la tesis.

Las figuras se encuentras en los archivos *.DRW, el nombre de cada archivo indica la figura que contiene.

Finalmente, el archivo TESIS.ASC contiene el texto de toda la tesis en formato ASCII.

2 Subdirectorio \CODIGO

Para distribuir el código fuente, la información se encuentra organizada en los siguientes subdirectorios:

\CODIGO\NETLIST
\CODIGO\CAMBIO
\CODIGO\REUBICA
\CODIGO\ENRUTADO
\CODIGO\GENERA
\CODIGO\INTERFAZ
\CODIGO\GENES

En cada uno de estos subdirectorios se presentan los archivos fuentes necesarios para generar a cada uno de los módulos.

Para obtener los archivos ejecutables, se deben transfereir los subdirectorios completos de cada módulo y ejecutar el comando "make" de UNIX.

Los programas fueron desarrollados empleando el Sistema Operativo UNIX versión BSD4.2 de la compañía GOULD.

Los comandos gráficos son enviados a una Terminal Gráfica de la marca Raster Technologies.

En caso de que se quiera observar los gráficos en otra terminal diferente se deben crear las primitivas de graficación equivalentes a las del archivo denominado s.c.

BIBLIOGRAFIA

[1] IDEA SERIES Schematic Capture User's Manual
Version 6.0.
Mentor Graphics, May 1987.

[2] IDEA SERIES Schematic Capture Reference Manual Volume I and II
Version 6.0.
Mentor Graphics, May 1987.

[3] IDEA SERIES Neted User's Manual
Version 6.0.
Mentor Graphics, May 1987.

[4] IDEA SERIES Symed User'a Manual
Version 6.0.
Mentor Graphics, May 1987.

[5] IDEA SERIES DTR User'a Manual
Version 6.0.
Mentor Graphics, May 1987.

[6] QUICKSIM Family Refere.ce Manual
Version 6.0.
Mentor Graphics, May 1987.

[7] VLSI ENGINEERING
Thomas E. Dillinger.
Prentuice Hall, 1988.

[8] An Algorithm to Compact a VLSI Symbolic Layout with mixed constrains.
Y.Liao and C. K. Wong
Transactions on CAD of ICAS Vol. 3 No. 1 January, 1984

[9] An Efficient Two-Dimensional Layout Compaction Algorithm.
Hyunchul Shin and Chi Yuan Lo
26th ACM/IEEE Design Automation Conference.

[10] Proceedings of the 26th Digital Automation Conference.

BIBLIOGRAFIA

[11] A Solution to line-routing problems on the continuous plane.
David W. Hightower
Bell Telephone Laboratories Incorporated, New Jersey.

[12] Estructuras de matemáticas discretas para la computación.
Bernard Kolman.
Robert C. Busby.
Prentice Hall, 1986.

[13] EDIF Connectivity.
Electronic Industries Association.
EDIF Steering Committee, 1988.

[14] Introduction to EDIF.
Electronic Industries Association.
EDIF Steering Committee, 1988.

[15] Using EDIF for Schematic Transfer
Electronic Industries Association.
Engineering Departament, July 1989.

[1] Electronic Design Interchange Format.
Version 2 0 0
Recommended standard EIA-548.
Electronic Industries Association.
Engineering Departament, 1988.
ANSI/EIA-548-1988

[7] Computer Graphics.
A programming Approach.
Steven Harrington.
Mc. Graw Hill, 1983

[18] The Theory and practice of compiler writing.
Jean-Paul Tremblay.
Paul G. Sorenson.
Mc. Graw Hill, 1985.

[19] Compiler Principles Technics and Tools
Alfred V. Aho.
Paul Sethi.
Jeffrey D. Ullman.
Addison Wesley, 1987.

EL JURADO DESIGNADO POR LA SECCION DE COMPUTACION DEL DEPARTAMENTO DE INGENIERIA ELECTRICA, APROBO EL DIA 6 DEL MES DE DICIEMBRE DE MIL NOVECIENTOS NOVENTA, EL TRABAJO DE TESIS " GENERADOR DE HOJAS DE ESQUEMATICOS GENES" QUE PRESENTO EL ING. LENIN GUILLERMO LEMUS ZUÑIGA.

DR. ARTURO VELOZ GUERRERO

M. EN C. ANDRES VEGA GARCIA

M. EN C. JESUS PALOMINO ECHARTEA