Conexionado de Redes (Routing)

MISE DASE
Curso 2010/2011
Marisa López Vallejo

Índice (I)

- Introducción
- Clasificación
- Algoritmo de laberinto
  - Introducción
  - Modelo inicial
  - Procedimiento de operación
  - Extensiones al modelo
    - Redes multipunto
    - Capas múltiples
    - Eficiencia/Complejidad. Frente de Onda
  - Consistencia del coste
  - Esquemas inteligentes

Índice (II)

- Rutado de canal
  - Definiciones de grafos (repaso)
  - Planteamiento del problema
    - Modelo
    - Objetivos
    - Restricciones H, Restricciones V
  - Algoritmos básicos
    - Borde izquierdo (Left Edge Algorithm)
    - Basado en cliques

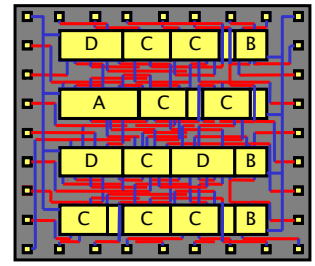
Índice (III)

- Rutado de canal (cont.)
  - YACR2
    - Variante del LEA
    - Eliminación Violaciones RV
  - Canales con RV cíclicas

Introducción (I)

- Objetivos fase de rutado
  - Conectar todas las redes
  - Verificar reglas tecnológicas
  - Maximizar prestaciones (tiempos propag. cortos)
  - Minimizar área
  - Invertir poco tiempo CPU

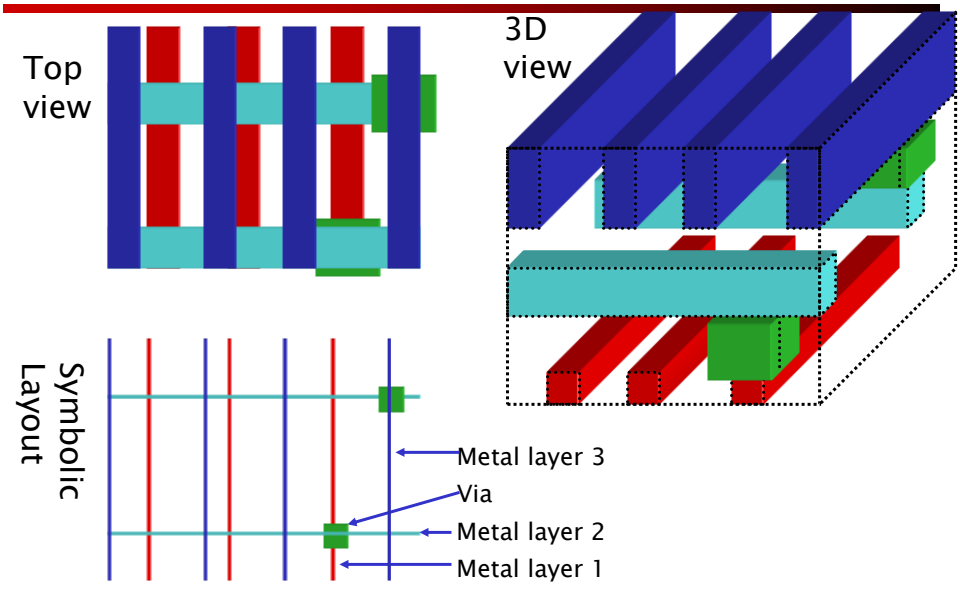
- Tras fase rutado se tiene el trazado del chip



Introducción (II)

- Información necesaria para rutado
  - Netlist
  - Colocación (situación bloques, pines)
  - Restricciones tecnológicas
    - Capas (*layers*)
      - Chip: mínimo 2, rutado H y rutado V
      - PCB: normalmente sólo 1
    - Direcciones
      - Normalmente H y V. También 45º
    - Anchura: distinta para cada material/capa
    - Vías: conexiones entre capas
      - Tamaño, separación, localización, verticalidad
  - Requerimientos de tiempos (redes críticas)
    - Retardo RC por unidad longitud: capa metal, vía, etc.

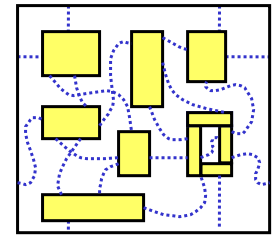
Anatomía del rutado



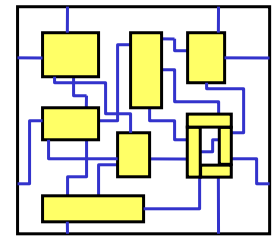
[©Bazargan] 7

Clasificación Rutado: Global vs. Detallado

- Rutado Global
  - Input: placement detallado, con localización exacta de terminales
  - Buscar el "canal" (región de rutado) para cada red
  - Objetivo: minimizar área (congestión), y temporización (aproximado)



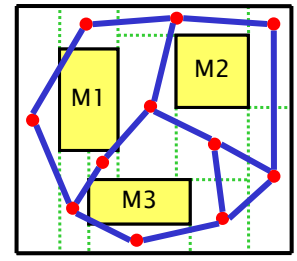
- Rutado Detallado
  - Input: canales y rutado aproximado de la etapa anterior (rutado global)
  - Buscar la ruta exacta para cada red, con las capas necesarias
  - Objetivo: rutado válido, minimizar área (congestión), cumplir restricciones de temporización
  - Objetivos adicionales: min via, potencia



Figs. [©Sherwani]

Clasificación (I)

[©Sarrafzadeh]



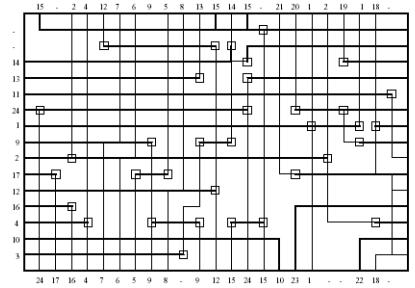
- Global:
  - Divide superficie rutable en regiones
    - Divide y vencerás
    - Regiones poco definidas (coarse)
    - Tipos: canal, switch-box, etc
  - Asigna cada red a una región
    - No especifica rutado detallado de red
    - Deja que un router detallado resuelva geometría final de redes, en cada región
  - Tipos: laberinto, steiner tree, mejora iterativa, ILP

Clasificación (II)

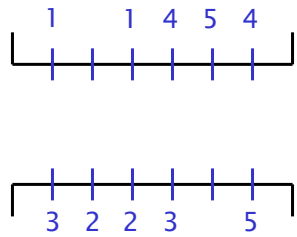
- Detallado
  - Halla geometría exacta del trazado de red
  - Tipos
    - En toda el área del trazado
      - Computacionalmente intratable. Sólo aplicable en casos especiales (área pequeña)
      - Algoritmo del laberinto
    - En área restringida
      - Canal: área rectangular rodeada por bloques en lados opuestos
        - » Algoritmo borde izqdo., algoritmo YACR2
      - *Switch box*: área rectangular rodeada por bloques en todos los lados
        - » Modela esquinas de canales en L
      - Otros
- Especiales: árbol de reloj, alimentación, masa,...

Entornos de rutado

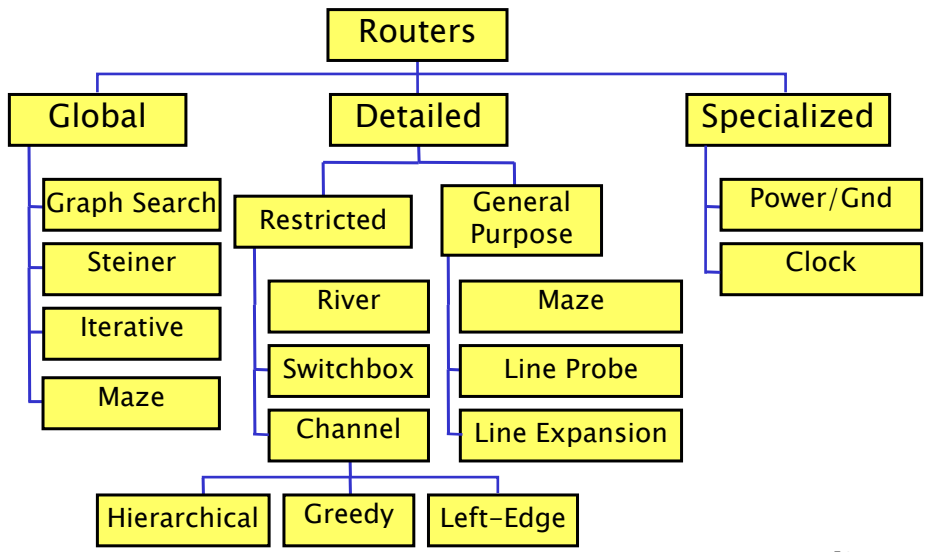
- Switch box



- Canal



Taxonomía de VLSI Routers



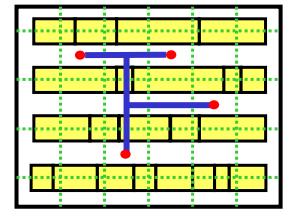
[©Keutzer]

Algoritmo de laberinto (Maze)

- Válido para rutado detallado (o rutado global si el área es pequeña)
- Modo de operación
  - Rutado de una red en cada paso
    - Conecta completamente todos los pines de una red
    - Usando tanta área como se requiera (no restrinjo el rutado a regiones)
  - Problema:
    - Dependencia secuencial: red actual bloquea a siguientes
  - Solución:
    - Rutar primero las redes críticas

Modelo inicial Maze

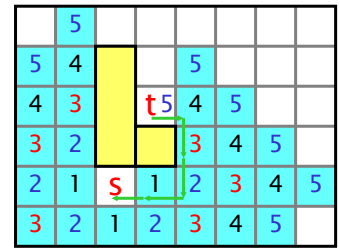
- Rejilla: cada cuadrícula representa
  - Zona de tránsito H o V de un cable
  - Zona de cambio de dirección de un cable



- Consideramos redes punto a punto
  - Source (S) y Target (T)
- Encontrar el camino más corto que conecta S con T
  - Tránsito entre células: coste unidad

Idea del laberinto

- Basado en breadth-first search
  - Algoritmo muy simple
  - Trabaja con un grafo de la rejilla
  - Complejidad: tamaño de rejilla (NxN)
- Algoritmo
  - Propagar una "onda" desde origen (source) a destino (target)
  - Volver camino más corto
- Garantiza encontrar óptimo
  - Habitualmente hay varias soluciones óptimas
- ¿Redes multi-punto?
  - Para el resto de terminales utilizar el camino entre los dos primeros como origen de la onda



Mecanismo de operación (I)

Inicialización:
- Etiquetar S como célula activa.
- Inicializar variable Distancia D=1

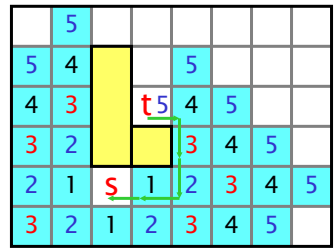
Expansión
- Encontrar células vecinas a las células activas
  - A distancia D de S (coste de camino a S igual a D)
  - Desactivar actuales células activas y marcar células alcanzadas como activas
  - D=D+1
  - Repetir punto 2 hasta llegar a T

Back-tracking
- Desandar (backtracking) hasta llegar a S
  - Seguir números decrecientes
  - Intentar minimizar núm. de cambios dirección

Limpieza
- Limpiar rejilla, marcando el camino creado como obstáculo para futuros hilos
- Rutar nuevos hilos

Mecanismo de operación (II)

- Basado en breadth-first-search
  - Garantiza encontar el camino entre S y T, si existe, y será óptimo (el más corto)
  - Pero el rutado de una red dificulta el siguiente
    - El orden de selección importa ⇒ heurístico



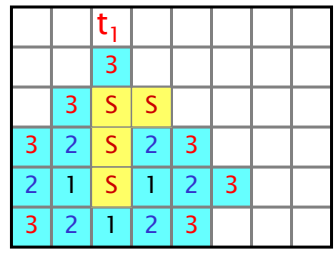
Extensiones al modelo

- Redes multipunto
- Tecnología: varias capas
- Eficiencia/complejidad
  - Almacenamiento:
    - ¿almacenar el coste del camino en todas las células alcanzadas? bits/célula ↑ ↑
    - Avance por frente de onda
  - Velocidad de búsqueda:
    - Avance por frente de onda
    - Esquemas predictivos (búsqueda no simétrica)

Redes multipunto (I)

- Pasos:
  - Etiquetar un pin como S, y el resto como T
  - Buscar camino más corto S-T
    - A priori desconocemos qué T se alcanzará primero
  - Tras alcanzar el primer T, etiquetar todas las células del camino como S
  - Continuar hasta alcanzar todos los T
    - Avanzando desde todos los S
- Dependencia respecto al pin inicialmente elegido
  - Análogo a árbol de Steiner

Redes multipunto (II)



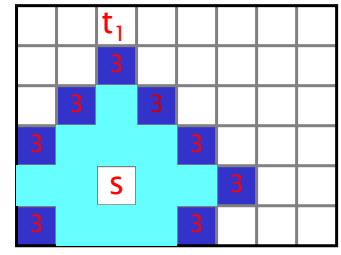
Múltiples capas de conexionado

- Mantener en paralelo una rejilla por cada capa
  - Etiquetar células donde se permitan vías
  - Expandir, pudiendo pasar de un grid a otro

- Refinamiento mediante costes ponderados
  - Coste de cada célula puede diferir de la unidad
    - Coste asociado a vías
    - Costes distintos en capas distintas
    - Costes distintos en regiones distintas de misma capa
  - Expandir parcialmente células activas a aquellas vecinas tal que coste camino a S sea C, C+1, C+2,...
    - Una célula puede tener varias expansiones hasta dejar de ser activa
  - Encuentra el camino de mínimo coste
    - Distinto de camino de mínima longitud

Eficiencia/Complejidad (I)

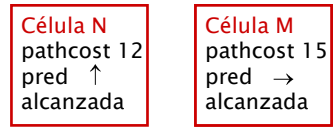
- Almacenamiento
  - No almacenar coste de camino en cada célula alcanzada
  - Concepto Frente de Onda



  - Estructuras de datos
    - Rejilla (array dinámico o estático)
    - Frente de onda (F.O.): Lista
      - Celdas activas para expandirse
  - Datos a almacenar
    - F.O.: coste del camino a la célula
    - Rejilla:
      - Coste individual de celda (rejilla ponderada)
      - 1 bit: célula alcanzada/no alcanzada
      - 2 bits: predecesor (N, S, E, O)

Eficiencia/Complejidad (II)

- Avance eficiente
  - Expandir completamente la celda del F.O. de menor coste camino
    - Si hay varias, usar heurísticos para desempatar
  - Añadir vecinas recién alcanzadas a la lista del F.O.
  - Eliminar del F.O. la célula expandida
    - Reduzco tamaño lista F.O. cuando puedo



Eficiencia/Complejidad (III)

```
Datos: S, T, pesos rejilla
Lista_F.O.={S}
while (lista_F.O. != vacía) {
   C=célula de menor coste en lista_F.O.;
   if (C is T) {
       Desanda el camino en rejilla; Limpia; exit;
   }
   else {
      for (cada vecino C' de C aún no alcanzado) {
       marcar nueva célula C' como ALCANZADA;
        calcular el coste de alcanzarla;
        apuntar en C' (lista_F.O.) el coste del camino a ella
        apuntar en C' (rejilla) su direc. acceso (predecesor C)
        añadir C' a lista_F.O.
        } /* del for */
        borrar C de lista_F.O.
   } /* del else */
} /* del while */
```

Eficiencia/Complejidad (IV)

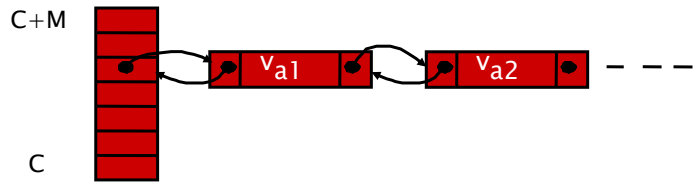
- Expansión completa (EC) vs. Expansión parcial (EP)
  - En E.C. incluyo células en F.O. que pueden tardar en expandirse
    - Por tener alto coste individual
    - Gasto tiempo en guardar en el F.O. células no útiles por ahora
      - Pero acceso a listas ordenadas es rápido: O(log n)
  - En E.P. **no** me preocupo de buscar un coste concreto para el camino a expandir
    - Ahorro cálculos, accesos a estructuras de datos, etc.
  - Comportamiento con E.C. es más regular y predecible que E.P.
    - Mayor independencia respecto a características particulares del problema

Eficiencia/Complejidad (V)

- Tras alcanzar una célula
  - Ningún camino futuro a ella tendrá menor coste
    - Pues expandimos siempre desde células de coste mínimo
  - Válido para funciones de coste consistentes
- Cada célula puede ser alcanzada varias veces
  - Desde varias direcciones
  - Sólo es importante el primer acceso, ya que indica el camino mínimo
    - Guardamos constancia del primer acceso (predecesor)

Estructuras de datos (I)

- Grid: array
- F.O.:
  - a) Lista doblemente enlazada:
    - Sin ordenar: Inserción rápida, búsqueda célula de menor coste es lenta
    - Ordenada: Inserción lenta, búsq. rápida
  - b) Buckets: array posibles valores costes F.O.
    - Búsq. rápida (más que lista ordenada)
    - Inserción rápida (comparable a lista sin ordenar)

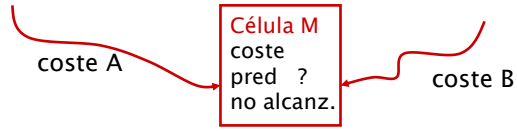


Estructuras de datos (II)

- Buckets
  - Problema: ¿tamaño array? ¿acotado?
  - Claves:
    - Siempre se expande la célula coste mín (C)
    - Coste máx individual por célula acotado (M)
    - Caso peor: expandir célula de C a C+M
      - Requiere array de M + 1 elementos
  - Tamaño array fijo
    - Costes internos variables (array circular)
    - Índices de coste menor y mayor
      - Necesarios porque reutilizamos posiciones (hay que descartar basura)

Consistencia de func. coste (I)

- Función coste consistente:
  - Incremento coste en acceso no depende del camino de acceso a célula
- Ejemplo función coste consistente
  - Coste interno a la célula
- Ejemplo función coste inconsistente
  - Coste en bordes de célula
  - Penalización por cambios dirección



Consistencia de func. coste (II)

- Función coste inconsistente
  - Puede aparecer la misma célula varias veces en F.O. con coste distinto
  - Seguir expandiendo la de menor coste primero
  - PERO:
    - Detener búsqueda sólo cuando cada célula del F.O. tenga coste mayor que T
    - Abandonar sólo cuando ya no es posible alcanzar T con un coste menor que el mínimo del F.O.

Esquemas inteligentes

- Esquema de Rubin
  - Término de predicción en la función de coste
  - Búsqueda 1º en profundidad hacia T
- Función de coste

$$cost[cell_i] = path\_cost[S \rightarrow cell_i] + predict(path\_cost[cell_i \rightarrow T])$$

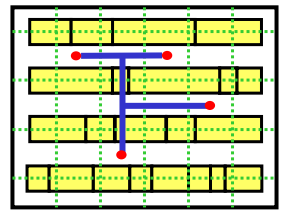
  - Término predicción:
    - Cota inferior coste desde célula actual a T
- Depth-first search
  - Entre células de mismo coste, las más próximas al target se expanden primero

Rutado de canal

- Conceptos previos: revisión de teoría de grafos
  - Subgrafo inducido
  - Grafo completo
  - Grafo complementario
  - Clique
  - Clique máximo

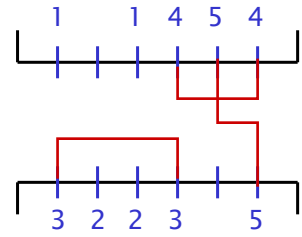
Definición del problema

- Rutado de canal:
  - Conectar pines de una red a ambos lados del canal
    - Segmentos H y V
      - Dos capas de metal
      - Conexión mediante vías
    - Conexionado Manhattan



Modelo de representación

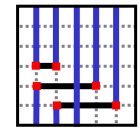
- ✓ Terminales en columnas equiespaciadas
- ✓ Dos terminales por columnas (top, bottom)
- ✓ Una red ocupa un segmento horizontal (pista) y segmentos verticales



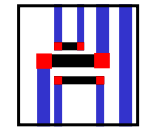
- Evitar:
  - *Dogleg*: red que ocupa varias pistas del canal, sin retroceso
  - *Jog*: red que ocupa varias pistas del canal, con retroceso
- Definiciones:
  - Capacidad de canal: espacio entre 2 hileras de células estándar
  - Densidad local de una columna j: nº redes que cruzan j
  - Densidad de canal (k): máxima densidad local

Elementos del problema

- Input
  - Posición fija de terminales arriba y abajo
  - Posible restricción de capacidad de canal (capacidad = max # de líneas horizontales de arriba a abajo del canal)
  - Con o sin rejilla



Con rejilla



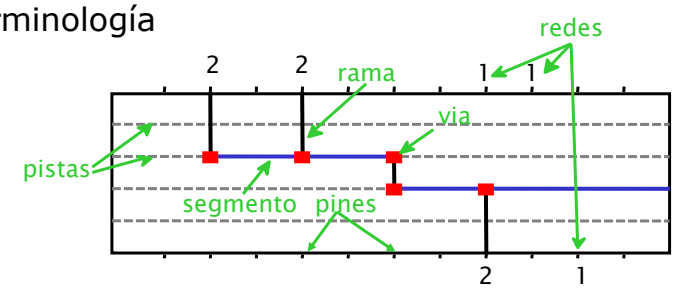
Sin rejilla

- En los algoritmos consideramos
  - Restricciones:
    - Estructura de la rejilla
    - Dos capas para rutar (una H, otra V)
  - Minimizar
    - # pistas (altura del canal)
    - Longitud total de conexionado
    - # vias

[©Sherwani]
[©Keutzer]

Terminología y algoritmos

- El caso general es NP-Completo
- Algoritmos
  - Caso sencillo: left-edge algorithm (P)
  - Caso general: NP ➔ heurísticos
    - Problema definido mediante grafo de intersección de canal y grafo de restricciones verticales
    - Se utilizan tanto algoritmos de grafos como algoritmos greedy
- Terminología



[©Keutzer]

Objetivos

- Minimizar tamaño routing (área canales)
- Minimizar longitud conexionado
  - Minimizar área
  - Minimizar tiempos propagación
- Equivalen a minimizar número de pistas:
  - $L_{HORIZ.}$ es constante para una red
  - Pines conectados a mismo lado del canal:

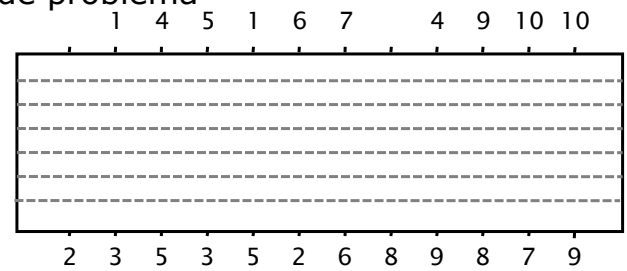
$$Long.\ red = L_{HORIZ.} + N^{o}\ pines \times L_{VERTIC.}$$

  - Pines conectados a distinto lado del canal:

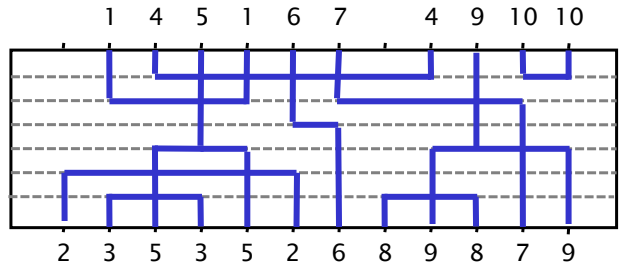
$$Long.\ red = L_{HORIZ.} + Anchura\_canal \times \min\left\{\#\ pines_{top}, \#\ pines_{bottom}\right\} + L_{VERTIC.}\left(\#\ pines_{lado\ con\ más\ pines} - \min\left\{\#\ pines_{top}, \#\ pines_{bottom}\right\}\right)$$

Ejemplo de rutado de canal

Ejemplo de problema



Solución:



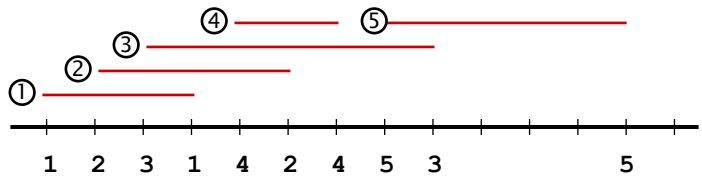
[©Keutzer]

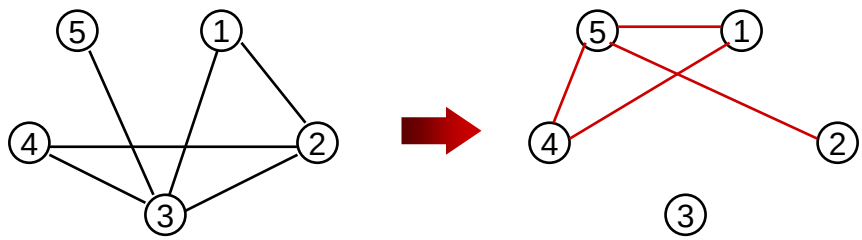
Grafo de restricciones horizontales (HCG)

Restricción horizontal

- Entre 2 segmentos H con columna común
- No pueden compartir la misma pista
- Grafo restricciones horizontales (intersección de canal)
  - Grafo no dirigido
  - Nodo: representa red
  - Arco: une redes que se intersectan
- Grafo de compatibilidad pistas
  - Complementario a grafo restricciones

Ejemplo



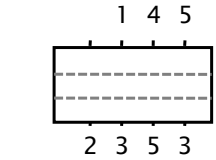


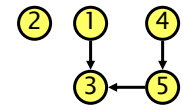
Grafo de Intersección de Canal

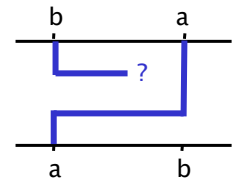
Grafo de Compatibilidad de Pistas

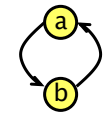
Grafo de restricciones verticales (VCG)

- Restricción vertical
  - Dos redes tienen un terminal en la misma columna
  - Sólo puede haber 2 redes con R.V. en misma columna
  - Grafo restricciones verticales
    - Grafo dirigido
    - Nodo: representa red
    - Arco de i a j: orientado ⇒ red i debe rutarse por encima de red j
  - R.V. cíclica:
    - Dadas 2 redes, ∃ R.V. contrarias en 2 pares de pines
    - Se deben rutar mediante *doglegging*



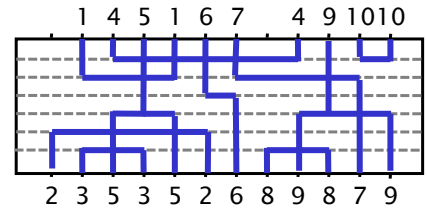


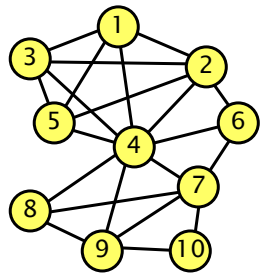




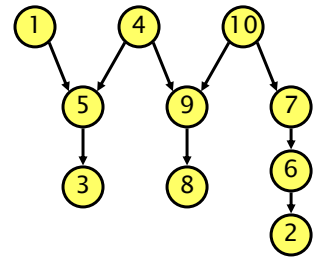
Grafos restricciones Horizontal/Vertical

- El problema de rutado de canal se puede caracterizar completamente mediante HCG y VCG





Grafo de restricciones horizontales (HCG)

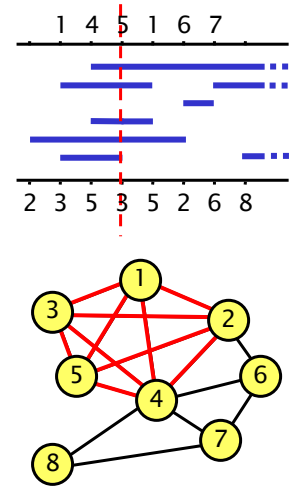


Grafo de restricciones verticales (VCG)

[©Keutzer]

Densidad de Canal

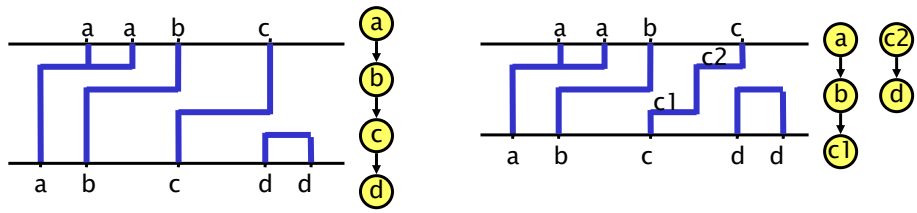
- Una red se extiende desde su terminal más a la izq. hasta su terminal más a la drcha
- Densidad local en columna C
  - ld (C) = # red dividida por la col. C
- Densidad de canal
  - d = max ld(c) de todo en C
- ¿Relación con HCG?
  - Densidad local ⇔ clique en HCG
  - d ⇔ tamaño del clique máximo en HCG
- Cota inferior:
  - # pistas ≥ d



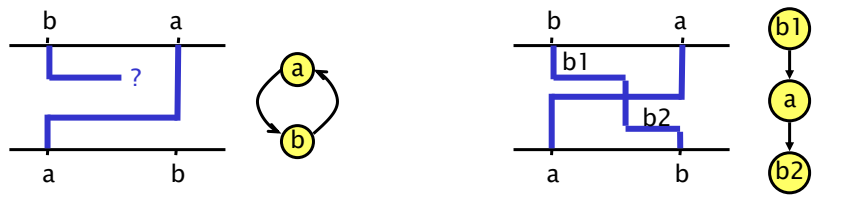
[©Keutzer]

Doglegs

Doglegs pueden reducir el camino más largo en VCG



Doglegs rompen ciclos en VCG



[©Keutzer]

Algoritmos básicos

- Asignan redes sin violar R.H.
- No tienen en cuenta R.V.
  - Pueden violar R.V.
- Son heurísticos
  - No garantizan el óptimo
- Tipos
  - Algoritmo de borde izquierdo (LEA)
  - Basado en cliques

Left Edge Algorithm (LEA)

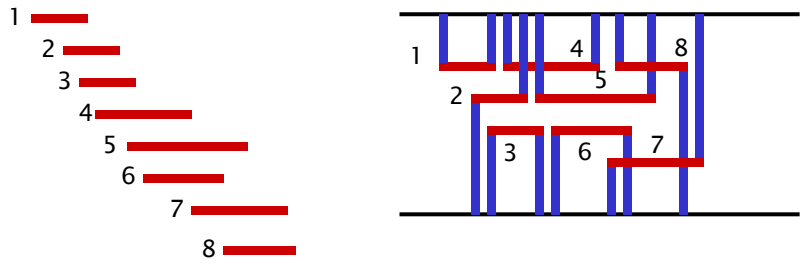
- Válido cuando no hay arcos VCG
- Encuentra solución óptima (# pistas = d)
- Las redes se ordenan en función de su borde izquierdo

```
pista=1

1. Ordenar segmentos de menor a mayor borde izqdo.
2. Elegir el primer segmento de la lista
3. Asignar a pista y borrar segmento de la lista
4. Buscar siguiente segmento que quepa en pista, asignarlo
   a pista, borrarlo de la lista.
5. Repetir 4 hasta llegar al final de la lista
6. pista++
7. Volver a 2 hasta que lista esté vacía
```

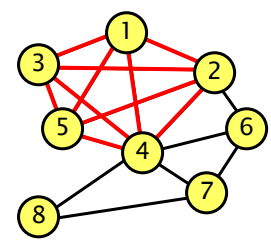
- Complejidad: O(n log n)

Ejemplo LEA



Basado en cliques (I)

- Concepto similar al LEA
- Construir grafo de compatibilidad y buscar cliques en él
- Clique ⇒ redes compatibles en la misma pista
  - Cuantas más redes por pista, menos pistas serán necesarias
  - Interesa hallar cliques máximos



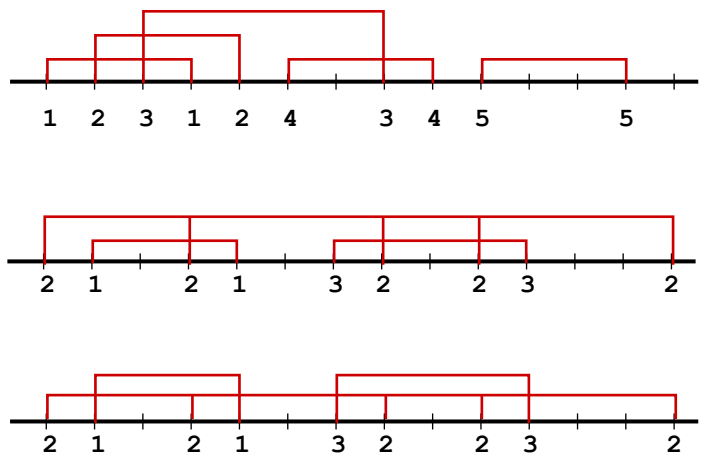
Basado en cliques (II)

- Redes a un solo lado del canal; redes punto a punto

```
pista = 1;
construir grafo complementario G_comp;

while (queden cliques en el grafo G_comp) {
   buscar el más grande;
   conectar redes del clique en la "pista";
   borrar nodos y arcos de esas redes en el grafo;
   pista++;
};
```


Ejemplo



Basado en cliques (II)

- Redes a un solo lado del canal; redes multipunto

```
pista = 1;
construir grafo complementario G_comp;

buscar cliques en G_comp;

for (cada clique j) {
```

$$Peso\ de\ clique\ j \sum_{red\ i\ de\ clique\ j} \#\ pines\ de\ red\ i$$

```
}
while (queden cliques en G_comp) {
   buscar clique con más peso;
   colocar clique en "pista";
   eliminar las redes de ese clique (arcos de G_comp);
   pista ++;
}
```


Basado en cliques (III)

- Redes a ambos lados del canal; redes multipunto
  - Elegir criterio colocación: p.e. colocar respecto al lado bottom
    - Las pistas van incrementándose hacia el lado up
    - El peso de cada clique es igual al número de pines bottom menos el número de pines up.
    - Un peso positivo significa que es más conveniente colocar la red en el lado bottom
  - De entre todas las redes con peso positivo ("quieren" conectarse al lado bottom) elijo aquella con mayor peso.

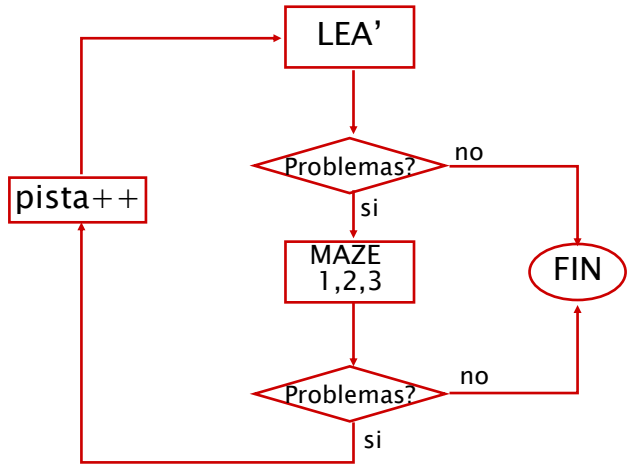
Comparación LEA vs. Clique

- Algoritmo basado en cliques usa modelo más completo
  - LEA coge la primera red que encuentra, sin mirar qué futuras redes existen
  - Produce mejores resultados que LEA
- LEA es más rápido

YACR2: Yet Another Channel Router

- Asigna redes sin violar R.H. ni R.V.
- Aplicable a grafos sin R.V. cíclicas
- Heurístico (no garantiza el óptimo)
- Tres etapas:
  1. Variante del LEA
     - Intenta conseguir máximo empaquetamiento en pistas
     - Garantiza ausencia de Violaciones R.H.
     - Intenta minimizar Violaciones R.V. (no lo garantiza)
     - Empieza por columna máxima densidad
     - Asigna todas las redes de una columna en cada iteración
       - En vez de todas las redes de una pista en cada iteración
  2. Utiliza espacios libres para eliminar V.R.V.
     - Maze particularizado a 3 escenarios típicos
  3. Si ∃ violaciones, #pistas ++ y vuelta a punto 1

YACR2: Diagrama de flujo



1. Variante del LEA

- Tres fases consecutivas
  1) Asignación de redes a pistas en columna de máxima densidad C_k
  2) Asignación de redes a pistas en columnas a la dcha. de C_k
  3) Asignación de redes a pistas en columnas a la izqda. de C_k
- Asignación: criterios heurísticos
  - Elegir 1º redes más conflictivas
    - Redes con mayor número de restricciones
    - Asignarlas cuando existan más grados de libertad
  - Asignar a pistas donde se prevea reducir conflictos
- Datos de partida:
  - Conjunto redes total
  - Nº inicial de pistas a utilizar = densidad canal = k

FASE-1: Seudocódigo

```
/* Inicialización */
Escoger  columna, maxcol,  tal que densidad de maxcol = k;
N={n1, n2, ...} conjunto redes que cruzan maxcol;
S= todas las pistas;

while (N≠Ø) {

   /* Escoger un n ∈ N para asignarlo /*
   n = select (N)

   /* Asignar n a una pista en S */
   assign (n, S);

   /* Actualizar */
   Eliminar n de N
   Eliminar la pista usada de S;
}
```

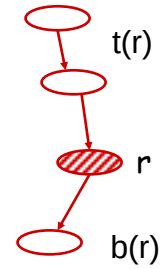

FASE-1: Selección Col. & Red

- Partimos de un tamaño de canal = k
  - No se producirán V.R.H.
- Columna de máxima densidad (*maxcol*):
  - Si hay varias columnas con densidad=k:
    1) Columna con máxima suma de RV de las redes que la atraviesan
    2) Columna con la red que tenga más RV
    3) Columna más centrada en el canal
- Selección siguiente red a asignar (*select*)
  1) Red con mayor nº VRV respecto a redes ya fijadas
  2) Red con mayor nº RV

FASE-1: Asignación pista a red

- Grafo RV
  - t(r) = nº redes por encima de red r
  - b(r) = nº redes por debajo de red r
- Si colocamos r en t(r) o b(r) habrá VRV
- No colocar r en t(r) o b(r) no asegura ausencia de VRV
  - Debido a asignaciones pasadas y futuras
- Fórmula heurística

$$\frac{ideal\ (r) - t(r) - 1}{k - b(r) - ideal\ (r)} = \frac{t(r)}{b(r)}$$



  - nº pistas en canal = k = densidad del canal
  - t(r) = 0 ⇒ ideal(r) = 1
  - b(r) = 0 ⇒ ideal(r) = k
  - t(r) = b(r) ⇒ ideal(r) = (k+1)/2

FASE-2: LEA modificado

```
/* Inicialización */
j = maxcol + 1; N= Ø; S= Ø;

while (j ≤ número columnas) {
   /* Asignar redes */
   if (∃ 1 red (ó 2) cuyo terminal a la izquierda está en j) {
        N=N ∪ redes con terminal a la izqda. en j;
        while (N≠Ø) {
              n = select (N);
              assign (n, S);
              Eliminar n de N;
              Eliminar pista usada de S;}
   }
   /* Ver si quedan nuevas pistas libres */
   if (∃ 1 red (ó 2) cuyo terminal a la derecha está en j) {
        S= S ∪ pista que queda libre;}

   /* Movernos hacia la derecha */
   j++;
}
```

FASE-3: LEA Modificado

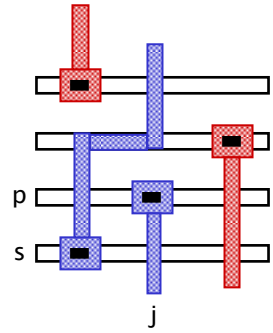
- Análogo a Fase-2
  - Avanzando hacia columnas de la izqda.
- Tamaño canal = k = máx densidad
  - Siempre hay pistas suficientes para no violar R.H. en Fase-2 y Fase-3

Eliminación de VRV

- Posibilidades
  a) *Maze* genérico:
     - Costoso, tiempo de cómputo elevado
  b) Tres algoritmos *maze ad-hoc*
     - Se ajustan a casos típicos
       - Menos esfuerzo computacional
     - Introducen doglegs
     - Pueden introducir pistas y contactos adicionales
     - Casos:
       - *Dogleg* en 1 red sobre columnas contiguas
       - *Dogleg* en 1 red sobre columnas no contiguas
       - *Dogleg* en 2 redes

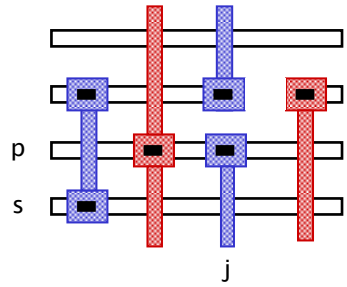
FASE-4 Maze routing para VRV

- Algoritmo 1
  - Busca en las columnas adyacentes
  - No añade ningún contacto



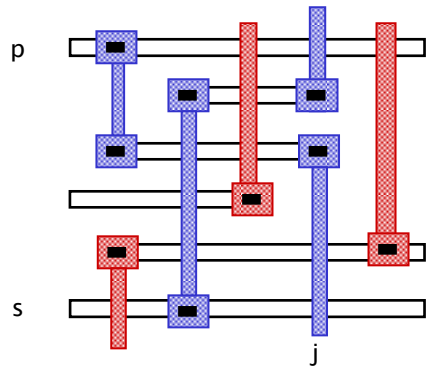
FASE-4 Maze routing para VRV

- Algoritmo 2
  - Dogleg en columnas apartadas
  - Dos contactos extra como máximo



FASE-4 Maze routing para VRV

- Algoritmo 3
  - Dogleg en las dos redes
  - Añade varios contactos



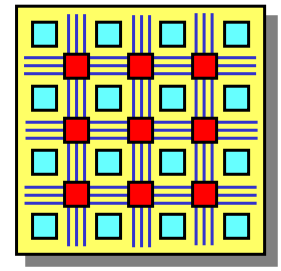
✓Si no funcionan: añadir una pista

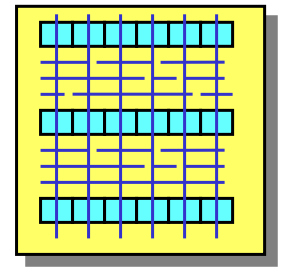
Canales con RV cíclicas

- No se pueden resolver sin *doglegs*
  1. Se rompe el grafo cíclico en uno acíclico
     - Se elimina la RV de la columna con más espacio libre alrededor
  2. Se resuelve el canal acíclico mediante YACR2
  3. Se incorporan las RV y se intenta resolver con algoritmos *maze* vistos (ad-hoc)
  4. Si no se puede, se añade una columna más y se repite el proceso

Arquitectura/Trazado de FPGAs

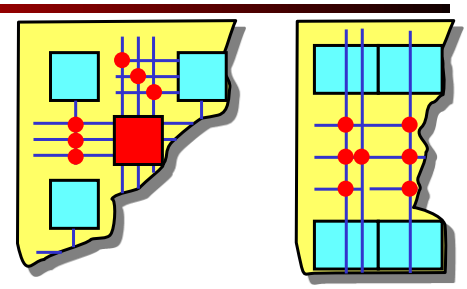
- FPGAs tipo matriz
  - Matriz de unidades funcionale
  - Canales horizontales y verticales de rutado para las unidades funcionales
  - Switch boxes versátiles
  - Ejemplo: Xilinx, Altera
- FPGAs basadas en filas
  - Como en el diseño de células estándar
  - Filas de bloques lógicos
  - Canales de rutado (anchura fija) entre hileras de lógica
  - Ejemplo: Actel



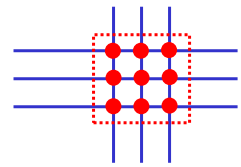


Elementos programables de interconexión

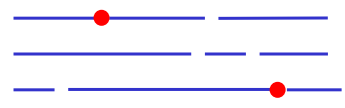
- Utilizados para conectar:
  - La E/S de las unidades funcionales a las conexiones



  - Una conexión horizontal a una conexión vertical

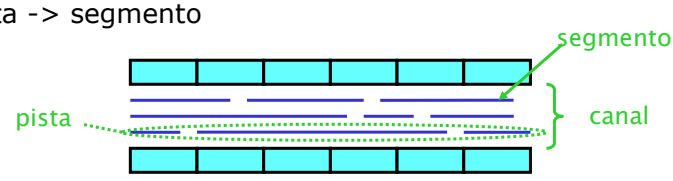


  - Dos segmentos de conexión para hacer una conexión mayor

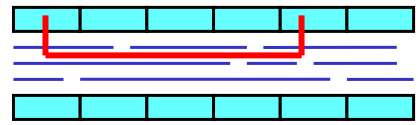


Arquitectura FPGA con canales de rutado

- Anchura de canal fija (pistas)
- Es preciso conocer todas las necesidades de conectividad a priori
- Canal -> pista -> segmento

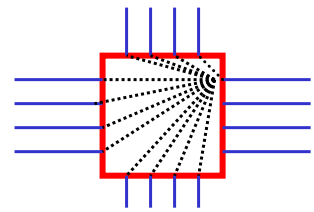


- ¿Qué longitud de segmento?
  - Largos: llevan señales más lejos, con menos cajas de "concatenación", pero se puede desperdiciar área
  - Cortos: conexiones locales, lento para conexiones largas

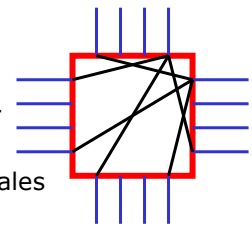


Switches de interconexión

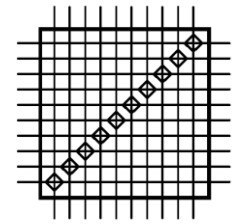
- Idealmente, proporcionan switches para todas las posibles conexiones



- Compromiso:
  - Demasiados switches:
    - Mucha area
    - Complejo de programar
  - Demasiados pocos:
    - No se pueden rutar señales



Solución posible



Xilinx 4000

Rutado de la FPGA

- Recursos de rutado pre-fabricados
  - 100% routability utilizando los canales existentes
  - Si falla el rutado de todas las redes, repartir placement
- Aspectos arquitecturales FPGA
  - Cuidado balance entre el número de bloque lógicos y los recurso de rutado (¿utilización de área llógica al 100%?)
  - Diseñar switchboxes flexibles y canales entra en conflicto con velocidades de reloj elevadas)
- Algoritmos de rutado en FPGAs
  - Algoritmos de búsqueda en grafos
    - Convertir los segmentos de conexión en nodos del grafo y los switch en arcos
  - Heurísticos de bin packing (redes como objetos, pistas como bins)
  - Combinación de maze routing y algoritmos de búsqueda en grafos