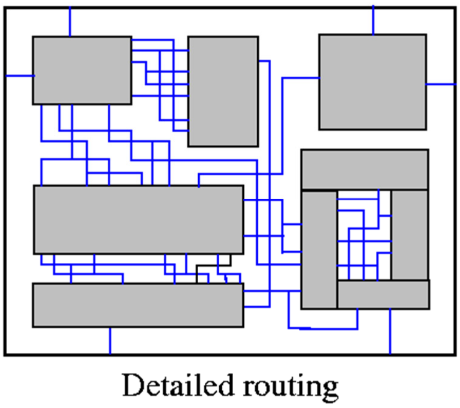
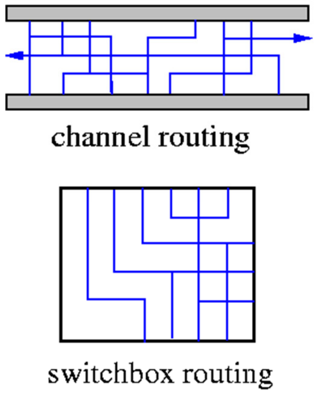
Unit 5E: Channel, Clock, and Power/Ground Routing

- Course contents
  - Channel routing
  - Clock routing
  - Power/ground routing
- Readings
  - Chapters 9.3 and 9.4





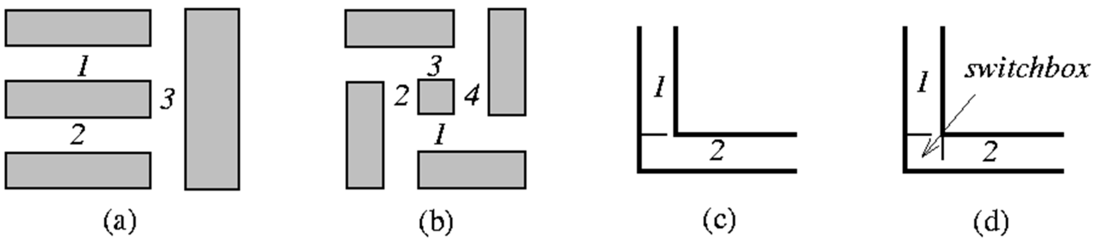
Order of Routing Regions and L-Channels

(a) No conflicts in case of routing in the order of 1, 2, and 3.

(b) No ordering is possible to avoid conflicts.

(c) The situation of (b) can be resolved by using L-channels.

(d) An L-channel can be decomposed into a channel and a switchbox.

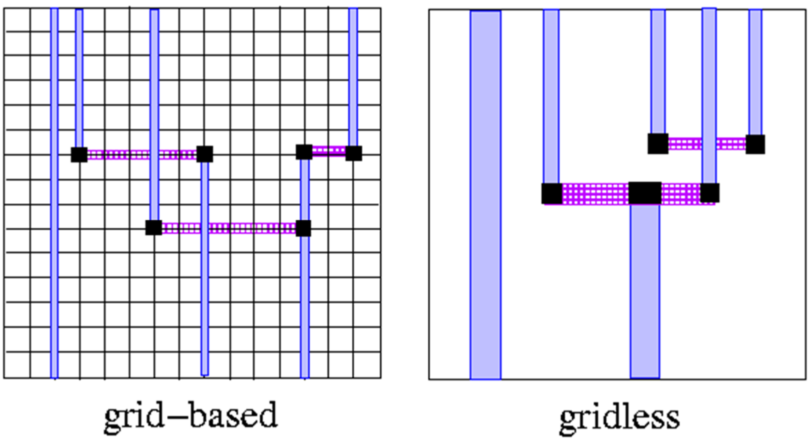


Routing Considerations

- Number of terminals (two-terminal vs. multi-terminal nets)
- Net widths (power and ground vs. signal nets)
- Via restrictions (stacked vs. conventional vias)
- Boundary types (regular vs. irregular)
- Number of layers (two vs. three, more layers?)
- Net types (critical vs. non-critical nets)

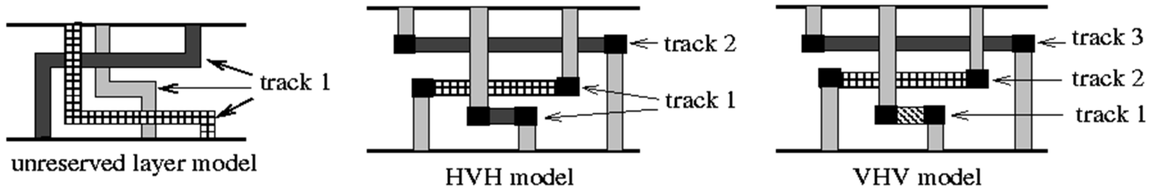
Routing Models

- **Grid-based model:**
  - A grid is super-imposed on the routing region.
  - Wires follow paths along the grid lines.
  - Pitch: distance between two grid lines.
- **Gridless model:**
  - Any model that does not follow this “gridded” approach.



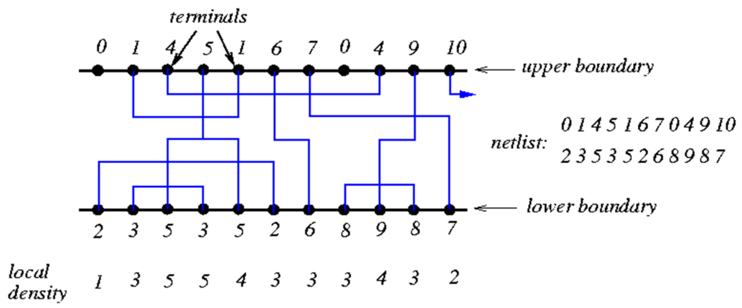
Models for Multi-Layer Routing

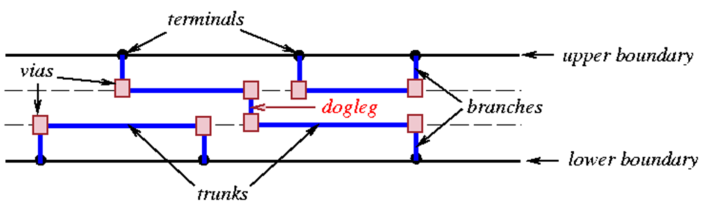
- **Unreserved layer model:** Any net segment is allowed to be placed in any layer.
- **Reserved layer model:** Certain type of segments are restricted to particular layer(s).
  - Two-layer: HV (horizontal-Vertical), VH
  - Three-layer: HVH, VHV



3 types of 3−layer models

Terminology for Channel Routing





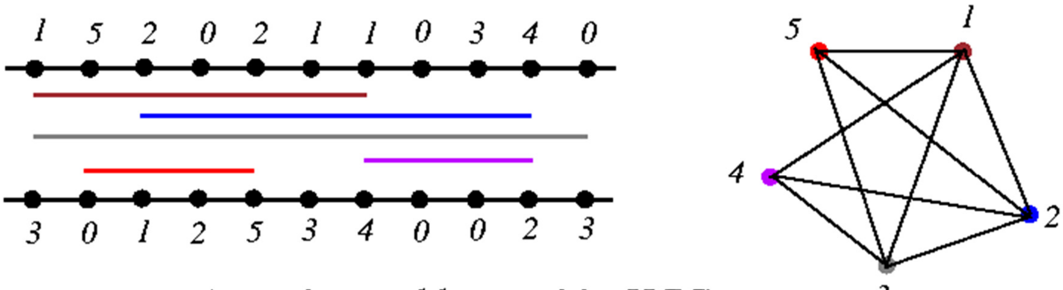
- Local density at column *i, d(i)*: total # of nets that crosses column *i*.
- **Channel density:** maximum local density
  - # of horizontal tracks required ≥ channel density.

Channel Routing Problem

- **Assignments of horizontal segments of nets to tracks.**
- **Assignments of vertical segments to connect.**
  - horizontal segments of the same net in different tracks, and
  - the terminals of the net to horizontal segments of the net.
- **Horizontal and vertical constraints must not be violated.**
  - Horizontal constraints between two nets: the horizontal span of two nets overlaps each other.
  - Vertical constraints between two nets: there exists a column such that the terminal on top of the column belongs to one net and the terminal on bottom of the column belongs to another net.
- **Objective: Channel height is minimized** (i.e., channel area is minimized).

Horizontal Constraint Graph (HCG)

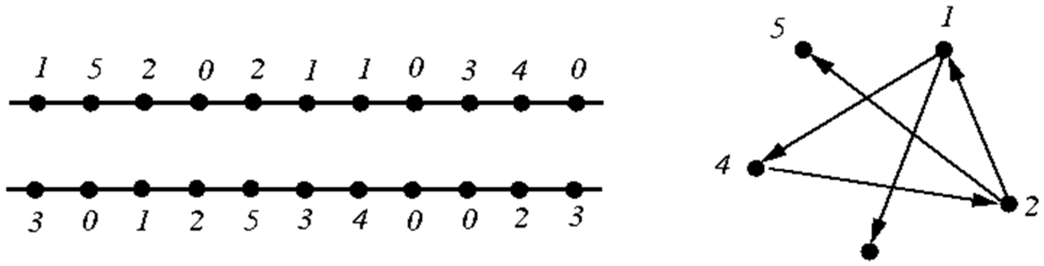
- HCG $G = (V, E)$ is **undirected** graph where
  - $V = \{ v_i \mid v_i$ represents a net $n_i\}$
  - $E = \{(v_i, v_j) \mid$ a horizontal constraint exists between n_i and $n_j\}$.
- For graph G: vertices $\Leftrightarrow$ nets; edge $(i, j) \Leftrightarrow$ net i overlaps net j.



A routing problem and its HCG.

Vertical Constraint Graph (VCG)

- VCG $G = (V, E)$ is **directed** graph where
  - $V = \{ v_i \mid v_i$ represents a net $n_i\}$
  - $E = \{(v_i, v_j) \mid$ a vertical constraint exists between n_i and $n_j\}$.
- For graph G: vertices $\Leftrightarrow$ nets; edge $i \rightarrow j \Leftrightarrow$ net i must be above net j.



A routing problem and its VCG.

2-L Channel Routing: Basic Left-Edge Algorithm

- Hashimoto & Stevens, "Wire routing by optimizing channel assignment within large apertures," DAC-71.
- **No vertical constraint.**
- HV-layer model is used.
- **Doglegs are not allowed.**
- Treat each net as an interval.
- Intervals are sorted according to their left-end x-coordinates.
- Intervals (nets) are routed one-by-one according to the order.
- For a net, tracks are scanned from top to bottom, and the first track that can accommodate the net is assigned to the net.
- Optimality: produces a routing solution with the minimum # of tracks (if no vertical constraint).

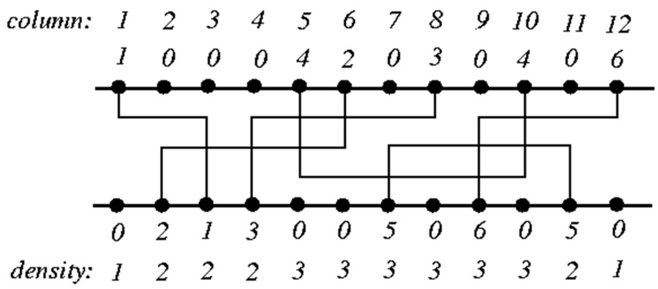
Basic Left-Edge Algorithm

Algorithm: Basic_Left-Edge(U, $track[j]$)
U: set of unassigned intervals (nets) $I_1, \ldots, I_n$;
$I_j=[s_j, e_j]$: interval j with left-end x-coordinate s_j and right-end e_j;
$track[j]$: track to which net j is assigned.

```
begin
U ← {I1, I2 , …, In};
t ← 0;
while (U ≠ ∅ ) do
    t ← t + 1;
    watermark ← 0;
    while (there is an Ij ∈ U s.t. sj > watermark) do
        Pick the interval Ij ∈ U with sj > watermark,
        nearest watermark;
        track[j] ← t;
     watermark ← ej;
     U ← U - {Ij};
end
```

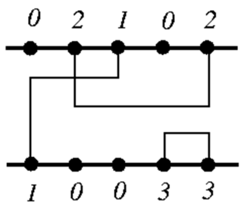

Basic Left-Edge Example

- $U = \{I_1, I_2, \ldots, I_6\}$; $I_1 = [1, 3]$, $I_2 = [2, 6]$, $I_3 = [4, 8]$, $I_4 = [5, 10]$, $I_5 = [7, 11]$, $I_6 = [9, 12]$.
- t =1:
  - Route I_1: *watermark* = 3;
  - Route I_3 : *watermark* = 8;
  - Route I_6: *watermark* = 12;
- t = 2:
  - Route I_2 : *watermark* = 6;
  - Route I_5 : *watermark* = 11;
- t = 3: Route I_4

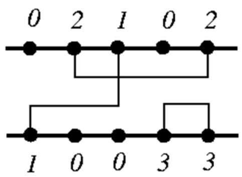


Basic Left-Edge Algorithm

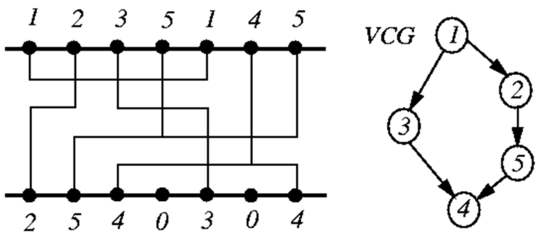
- If there is no vertical constraint, the basic left-edge algorithm is optimal.
- If there is any vertical constraint, the algorithm no longer guarantees optimal solution.



result from basic left−edge algorithm 3 tracks



optimal routing: 2 tracks



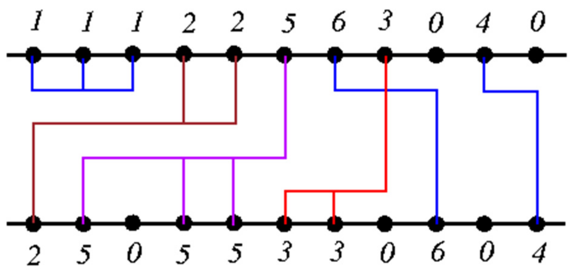
Constrained Left-Edge Algorithm

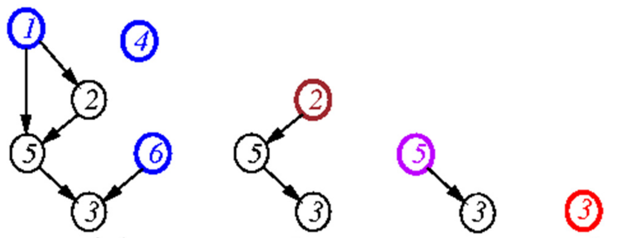
Algorithm: Constrained_Left-Edge(*U*, *track*[*j*])
U: set of unassigned intervals (nets) $I_1, \ldots, I_n$;
$I_j=[s_j, e_j]$: interval *j* with left-end *x*-coordinate s_j and right-end e_j;
track[*j*]: track to which net *j* is assigned.

```
begin
U ← { I1, I2, …, In};
t ← 0;
while (U ≠ ∅) do
    t ← t + 1;
    watermark ← 0;
    while (there is an unconstrained Ij ∈ U s.t. sj >
    watermark) do
    Pick the interval Ij ∈ U that is unconstrained,
      with sj > watermark, nearest watermark;
      track[j] ← t;
      watermark ← ej;
      U ← U - {Ij};
end
```

Constrained Left-Edge Example

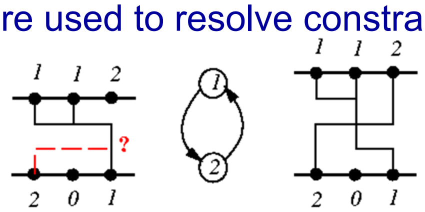
- I_1 = [1, 3], I_2 = [1, 5], I_3 = [6, 8], I_4 = [10, 11], I_5= [2, 6], I_6 = [7, 9].
- Track 1: Route I_1 (cannot route I_3); Route I_6; Route I_4.
- Track 2: Route I_2; cannot route I_3.
- Track 3: Route I_5.
- Track 4: Route I_3.



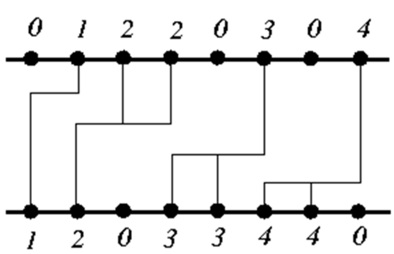


Dogleg Channel Router

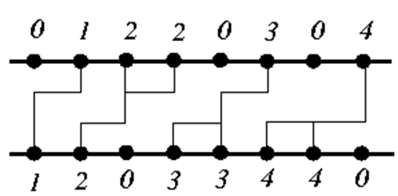
- Deutch, "A dogleg channel router," 13rd DAC, 1976.
- **Drawback of Left-Edge: cannot handle the cases with constraint cycles.**
  - **Doglegs** are used to resolve constraint cycle.



- **Drawback of Left-Edge: the entire net is on a single track.**
  - **Doglegs** are used to place parts of a net on different tracks to minimize channel height.
  - Might incur penalty for additional vias.



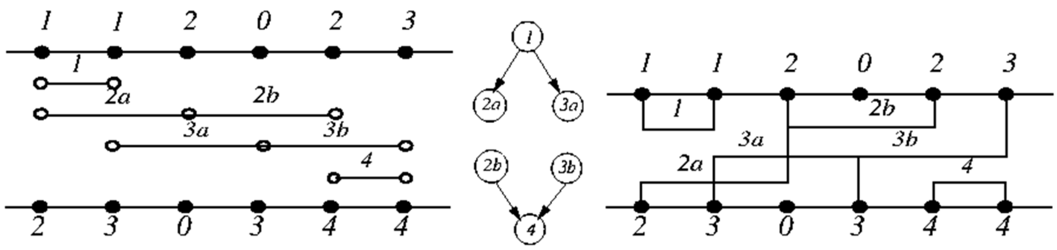
no dogleg



with dogleg

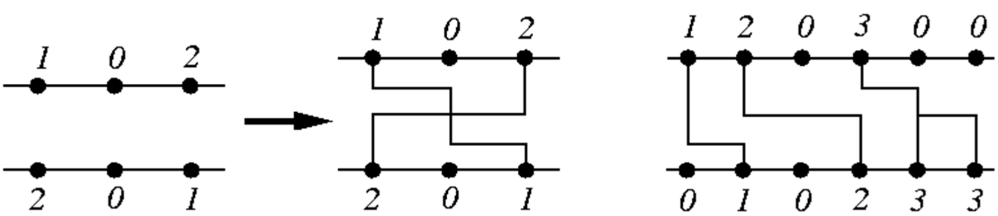
Dogleg Channel Router

- Each multi-terminal net is broken into a set of 2-terminal nets.
- Two parameters are used to control routing:
  - Range: Determine the # of consecutive 2-terminal subnets of the same net that can be placed on the same track.
  - Routing sequence: Specifies the starting position and the direction of routing along the channel.
- Modified Left-Edge Algorithm is applied to each subnet.



Restricted vs. Unrestricted Doglegging

- **Unrestricted doglegging:** Allow a dogleg even at a position where there is no pin.
- **Restricted doglegging:** Allow a dogleg only at a position where there is a pin belonging to that net.
- The dogleg channel router does not allow unrestricted doglegging.



dogleg channel router will fail! *Solution exists!* *restricted doglegging*

dogleg splits a net into subnets.

Robust Channel Router

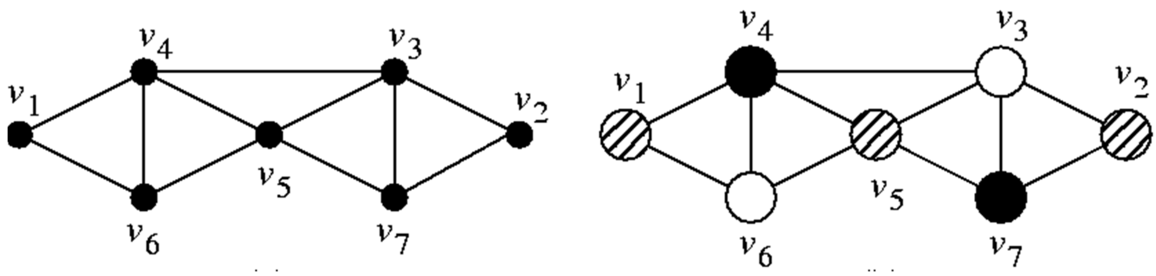
- Yoeli, "A robust channel router," IEEE TCAD, 1991.
- Alternates between top and bottom tracks until the center is reached.
- The working side is called the *current side*.
- Net weights are used to guide the assignment of segments in a track, which
  - favor nets that contribute to the channel density;
  - favor nets with terminals at the current side;
  - penalize nets whose routing at the current side would cause vertical constraint violations.
- Allows unrestricted doglegs by rip-up and re-route.

Robust Channel Router

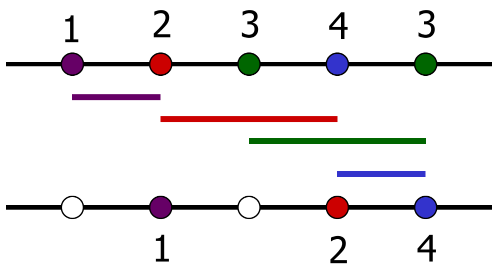
- Select the set of nets for the current side by solving the maximum weighted independent set problem for interval graphs.
  - NP-complete for general graphs, but can be solved efficiently for interval graphs using dynamic programming.
- Main ideas:
  - The interval for net i is denoted by $[x_{i_{min}}, x_{i_{max}}]$; its weight is w_i.
  - Process channel from left to right column; the optimal cost for position c is denoted by total[c];
  - A net n with a rightmost terminal at position c is taken into the solution if $\text{total}[c-1] < w_n + \text{total}[x_{n_{min}} - 1]$.
- Can apply maze routers to fix local congestion or to post-process the results. (Why not apply maze routers to channel routing directly??)

Interval Graphs

- There is a vertex for each interval.
- Vertices corresponding to overlapping intervals are connected by an edge.
- Solving the track assignment problem is equivalent to finding a **minimal vertex coloring** of the graph.



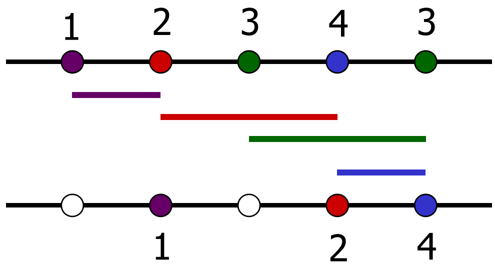
Weight Computation



d(1) = 1
d(2) = 2
d(3) = 2
d(4) = 3 (nets 2, 3, 4)
d(5) = 2

- Computation of the weight w_i for net i:
  1. favor nets that contribute to the channel density: add a large B to w_i.
  2. favor nets with current side terminals at column x: add d(x) to w_i.
  3. penalize nets whose routing at the current side would cause vertical constraint violations: subtract Kd(x) from w_i, K = 5 ~ 10.
  - Assume B = 1000 and K = 5 in the 1st iteration (top side):
    - $w_1 = (0) + (1) + (-5 * 2) = -9$
    - Net 1 does not contribute to the channel density
    - One net 1 terminal on the top
    - Routing net 1 causes a vertical constraint from net 2 at column 2 whose density is 2

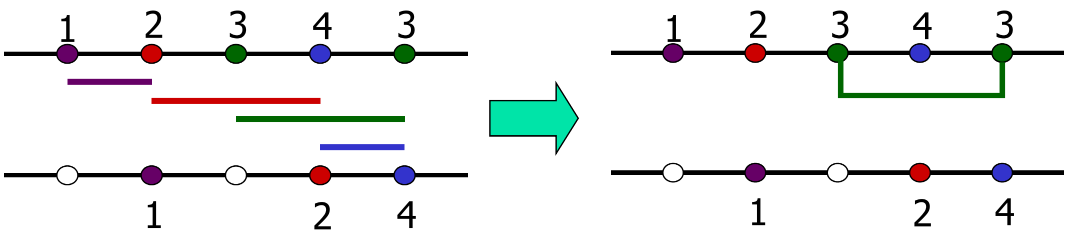
Weight Computation (cont'd)



d(1) = 1
d(2) = 2
d(3) = 2
d(4) = 3 (nets 2, 3, 4)
d(5) = 2

- Computation of the weight w_i for net *i*:
  1. favor nets that contribute to the channel density: add a large *B* to w_i.
  2. favor nets with current side terminals at column *x*: add d(*x*) to w_i.
  3. penalize nets whose routing at the current side would cause vertical constraint violations: subtract *K*d(*x*) from w_i, $K = 5 \sim 10$.
  - Assume *B* = 1000 and *K* = 5 in the 1st iteration (top side):
    - $w_1 = (0) + (1) + (-5 * 2) = -9$
    - $w_2 = (1000) + (2) + (-5 * 3) = 987$
    - $w_3 = (1000) + (2+2) + (0) = 1004$
    - $w_4 = (1000) + (3) + (-5 * 2) = 993$

Top-Row Net Selection

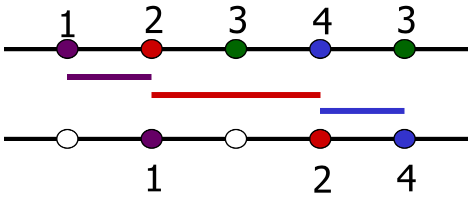


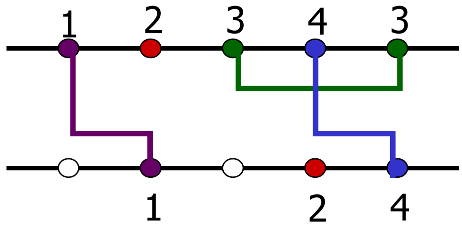
- $w_1 = -9$, $w_2 = 987$, $w_3 = 1004$, $w_4 = 993$.
- A net *n* with a rightmost terminal at position *c* is taken into the solution if: $\text{total}[c-1] < w_n + \text{total}[x_{n_{min}} - 1]$.

total[1] = 0	selected_net[1] = 0
total[2] = max(0, 0-9) = 0	selected_net[2] = 0
total[3] = 0	selected_net[3] = 0
total[4] = max(0, w_2+total[1]) = 987	selected_net[4] = 2
total[5] = max(987, 0+1004, 0+993) = 1004	selected_net[5] = 3

- Select nets backwards from right to left and with no horizontal constraints: Only net 3 is selected for the top row. (Net 2 is not selected since it overlaps with net 3.)

Bottom-Row Net Selection



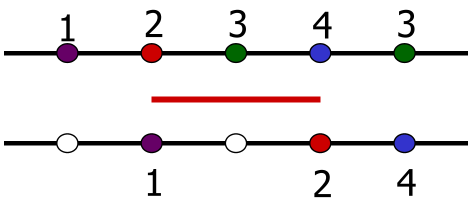


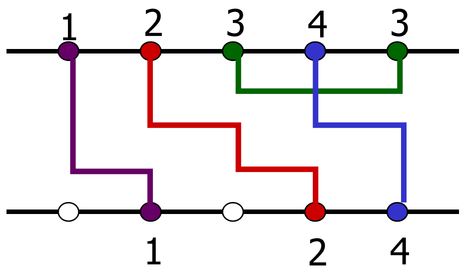
- 2^{nd} iteration: bottom-row selection
  - w_1 = (1000) + (2) + (0) = 1002
  - w_2 = (1000) + (2) + (-5 * 2) = 992
  - w_4 = (1000) + (1) + (-5 * 2) = 991

total[1] = 0	selected_net[1] = 0
total[2] = max(0, 0+1002) = 1002	selected_net[2] = 1
total[3] = 1002	selected_net[3] = 0
total[4] = max(1002, 0+992) = 1002	selected_net[4] = 0
total[5] = max(1002, 1002+991) = 1993	selected_net[5] = 4

- Nets 4 and 1are selected for the bottom row.

Maze Routing + Rip-up & Re-route





- 3^{rd} iteration
  - Routing net 2 in the middle row leads to an infeasible solution.
  - Apply maze routing and rip-up and re-route nets 2 and 4 to fix the solution.

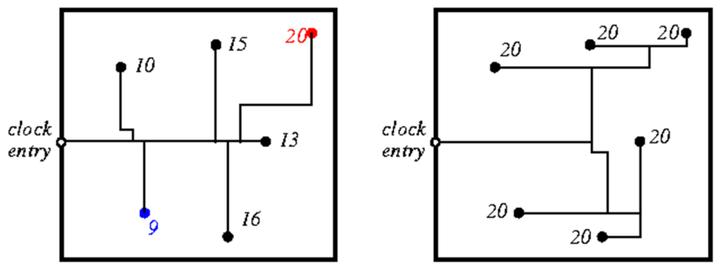
Robust Channel Router

```
robust_router (struct netlist N)
{
  set of int row;
  struct solution S;
  int total[channel_width + 1], selected_net[channel_width ·
  int top, height, c, r, i;

  top ← 1;
  height ← density(N);
  for (r ← 1; r ≤ height; r ← r + 1) {
    for all "nets i in netlist N"
      w_i ← compute_weight(N, top);
    total[0] ← 0;
    for (c ← 1; c ≤ channel_width; c ← c + 1) {
      selected_net[c] ← 0;
      total[c] ← total[c − 1];
      if ("some net n has a top terminal at position c")
        if (w_n + total[x_{n_min} − 1]) > total[c]) {
          total[c] ← w_n + total[x_{n_min} − 1]);
          selected_net[c] ← n;
        }
      if ("some net n has a bottom terminal at position c")
        if (w_n + total[x_{n_min} − 1]) > total[c]) {
          total[c] ← w_n + total[x_{n_min} − 1]);
          selected_net[c] ← n;
        }/* if */
    }/* for */
    row ← ∅;
    c ← channel_width;
    while (c > 0)
      if (selected_net[c]) {
        n ← selected_net[c];
        row ← row ∪ {n};
        c ← x_{n_min} − 1;
      }
      else
        c ← c − 1;
    solution ← solution ∪ {row};
    top ← !top;
    N ← "N without the nets selected in row"
  }/* for */
  "apply maze routing to eliminate possible vertical constraint violations"
}
```


The Clock Routing Problem (CRP)

- Digital systems
  - **Synchronous systems:** Highly precised clock achieves communication and timing.
  - **Asynchronous systems:** Handshake protocol achieves the timing requirements of the system.
- **Clock skew** is defined as the difference in the minimum and the maximum arrival time of the clock.



clock skew = 20 − 9 = 11 *clock skew* = 0

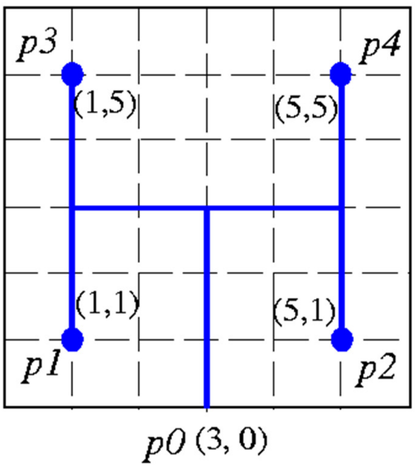
- **CRP:** Routing clock nets such that
  1. clock signals arrive simultaneously
  2. clock delay is minimized
     - Other issues: total wirelength, power consumption, etc

Clock Routing Problem

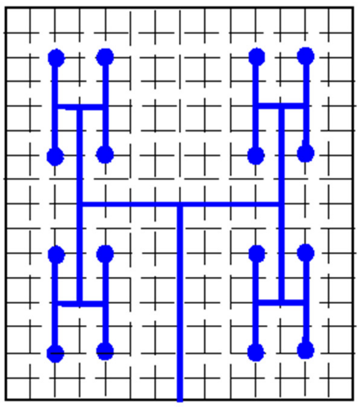
- Given the routing plane and a set of points $P = \{p_1, p_2, \ldots, p_n\}$ within the plane and clock entry point p_0 on the boundary of the plane, the **Clock Routing Problem (CRP)** is to interconnect each $p_i \in P$ such that $\max_{i, j \in P}|t(0, i) - t(0, j)|$ and $\max_{i \in P} t(0, i)$ are both minimized.
- Pathlength-based approaches
  1. *H*-tree: Dhar, Franklin, Wang, ICCD-84; Fisher & Kung, 1982. Geometric matching: Cong, Kahng, Robins, DAC-91.
- RC-delay based approaches:
  1. Exact zero skew: Tasy, ICCAD-91.
  2. Lagrangian relaxation: Chen, Chang, Wong, DAC-96.

H-Tree Based Algorithm

- *H*-tree: Dhar, Franklin, Wang, “Reduction of clock delays in VLSI structure,” ICCD-84.



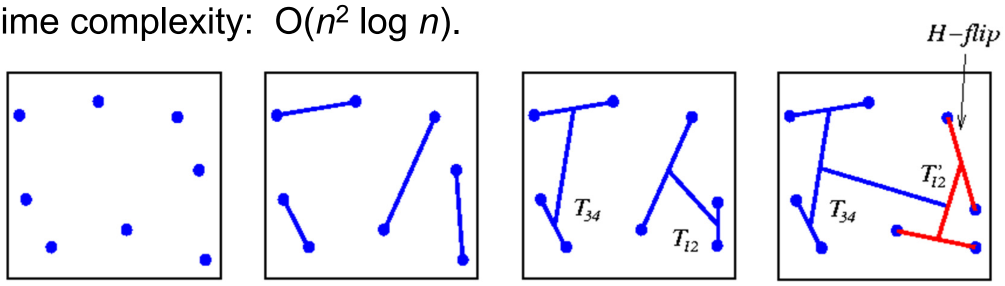
H–tree over 4 points



H–tree over 16 points

The Geometric Matching Algorithm

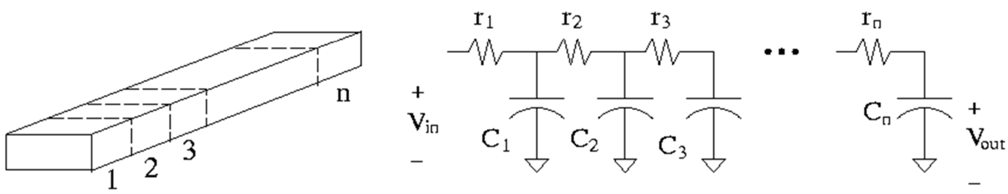
- Cong, Kahng, Robins, "Matching based models for high-performance clock routing," IEEE TCAD, 1993.
- Clock pins are represented as n nodes in the clock tree ($n = 2^k$).
- Each node is a tree itself with clock entry point being node itself.
- The minimum cost matching on n points yields $n/2$ segments.
- The clock entry point in each subtree of two nodes is the point on the segment such that length of both sides is same.
- Above steps are repeated for each segment.
- Apply H-flipping to further reduce clock skew (and to handle edges intersection).
- Time complexity: $O(n^2 \log n)$.



Elmore Delay: Nonlinear Delay Model

- Parasitic resistance and capacitance start to dominate delay in deep submicron wires.
- Resistor r_i must charge all downstream capacitors.
- **Elmore delay:** Delay can be approximated as sum of sections: resistance × downstream capacitance.

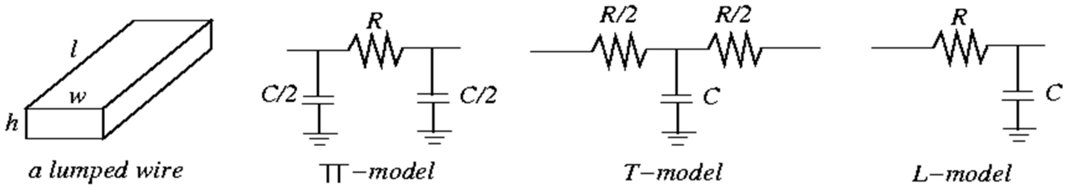
$$\delta = \sum_{i=1}^{n}\left(r_i \sum_{k=i}^{n} c_k\right) = \sum_{i=1}^{n} r(n-i+1)c = \frac{n(n+1)}{2} rc.$$



- Delay grows as **square** of wire length.

Wire Models

- Lumped circuit approximations for distributed RC lines: **π-model** (most popular), *T*-model, *L*-model.

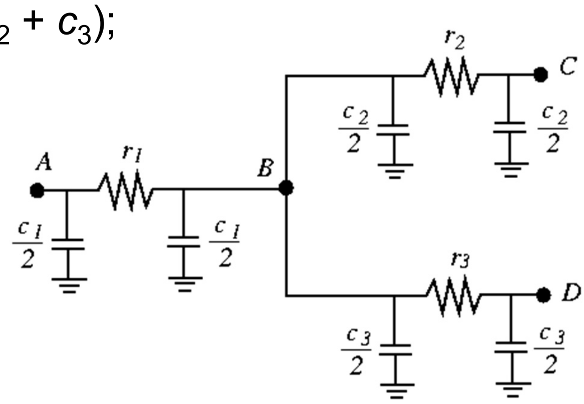


- π-model: If no capacitive loads for C and D,

  A to B: $\delta_{AB} = r_1\,(c_1/2 + c_2 + c_3)$;

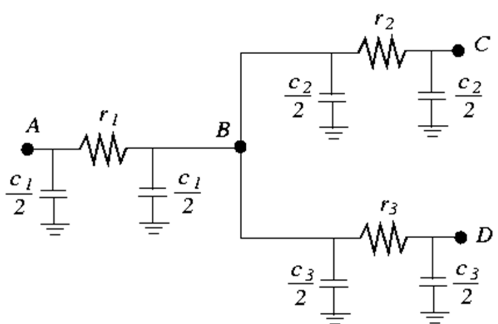
  B to C: $\delta_{BC} = r_2\,(c_2/2)$;

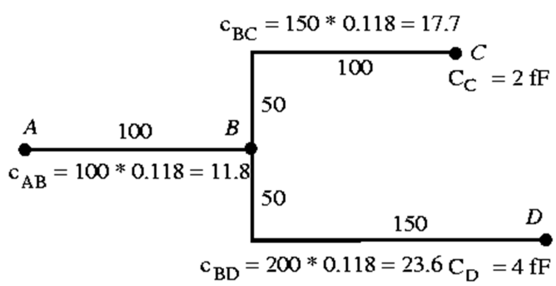
  B to D: $\delta_{BD} = r_3\,(c_3/2)$.



Example Elmore Delay Computation

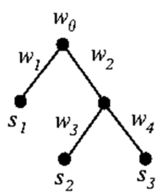
- 0.18 μm technology.: unit resistance $\hat{r}$ = 0.075 Ω / μm; unit capacitance $\hat{c}$ = 0.118 $fF/\mu m$.
  - Assume C_C = 2 fF, C_D = 4 fF.
  - $\delta_{BC} = r_{BC}\,(c_{BC}/2 + C_C) = 0.075 \times 150\,(17.7/2 + 2) = 120$ fs
  - $\delta_{BD} = r_{BD}\,(c_{BD}/2 + C_D) = 0.075 \times 200\,(23.6/2 + 4) = 240$ fs
  - $\delta_{AB} = r_{AB}\,(c_{AB}/2 + C_B) = 0.075 \times 100\,(11.8/2 + 17.7 + 2 + 23.6 + 4) = 400$ fs
  - Critical path delay: $\delta_{AB} + \delta_{BD} = 640$ fs.



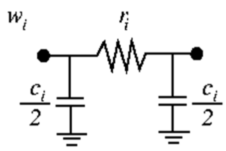


Delay Calculation for a Clock Tree

- Let T be an RC tree with points $P = \{p_1, p_2, \ldots, p_n\}$, c_i the capacitance of p_i, r_i the resistance of the edge between p_i and its immediate predecessor.
- The subtree capacitance at node i is given as $C_i = c_i + \sum_{j \in S_i} C_j$, where S_i is the set of all the immediate successors of p_i.
- Let $\delta(i, j)$ be the path between p_i and p_j, excluding p_i and including p_j.
- The delay between two nodes i and $_j$ is $t_{ij} = \sum_{j \in \delta(i, j)} r_j C_j$,
- $t_{03} = r_0 (c_1+c_2+c_3+c_4+c_1^s+c_2^s+c_3^s) + r_2(c_2/2 + c_3 + c_4 + c_2^s+c_3^s)+r_4(c_4/2+c_3^s)$.

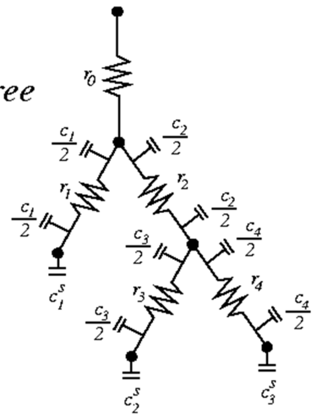


clock tree



delay model

RC tree



Exact Zero Skew Algorithm

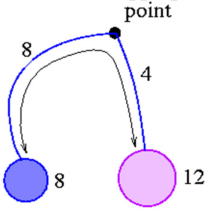
- Tasy, "Exact zero skew algorithm," ICCAD-91.
- To ensure the delay from the **tapping point** to leaf nodes of subtrees T_1 and T_2 being equal, it requires that

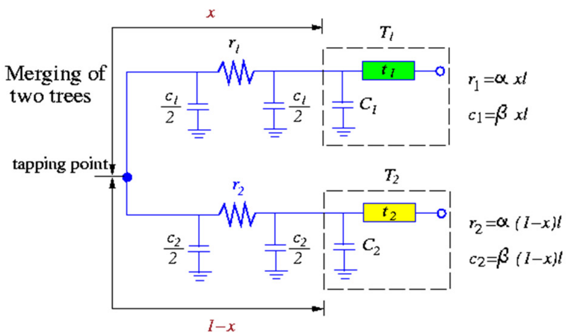
$$r_1 (c_1/2 + C_1) + t_1 = r_2 (c_2/2 + C_2) + t_2.$$

- Solving the above equation, we have

$$x = \frac{(t_2 - t_1) + \alpha l \left(C_2 + \frac{\beta l}{2}\right)}{\alpha l(\beta l + C_1 + C_2)},$$

where α and β are the per unit values of resistance and capacitance, l the length of the interconnecting wire, $r_1 = \alpha x l$, $c_1 = \beta x l$, $r_2 = \alpha(1 - x)\, l$, $c_2 = \beta(1 - x) l$.

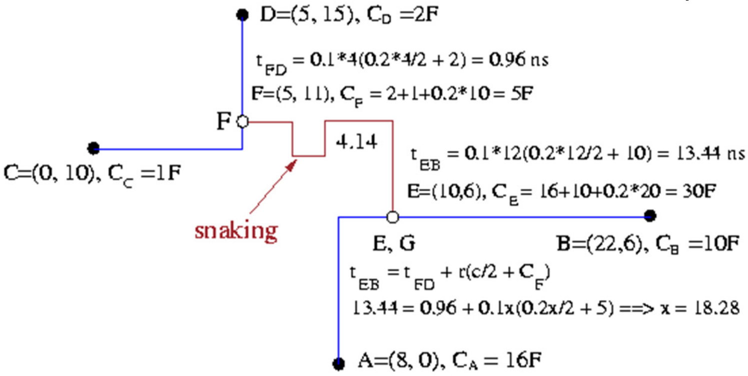




Zero-Skew Computation

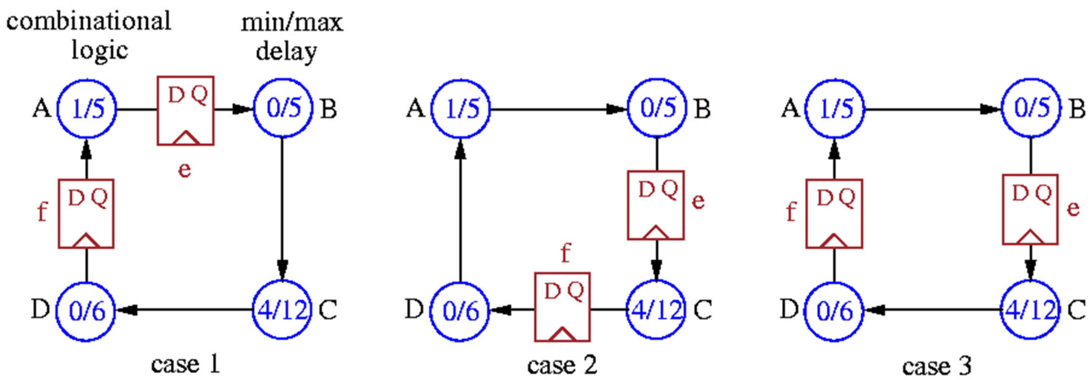
- **Balance delays:** $r_1(c_1/2 + C_1) + t_1 = r_2\,(c_2/2 + C_2) + t_2$.
- **Compute tapping points** $x = \dfrac{(t_2 - t_1) + \alpha l \left(C_2 + \frac{\beta l}{2}\right)}{\alpha l(\beta l + C_1 + C_2)}$, χ (β): per unit values of resistance (capacitance); l: length of the wire;

  $r_1 = \beta\, xl$, $c_1 = \beta x\, l$; $r_2 = \alpha(1 - x)\, l$, $c_2 = \beta\,(1 - x)\, l$.
- If $x \notin [0, 1]$, we need **snaking** to find the tapping point.
- Exp: $\alpha = 0.1\ \Omega$ /unit, $\beta = 0.2\ F$ /unit. (Find tapping points E for A and B, F for C and D, and G for E and F.)



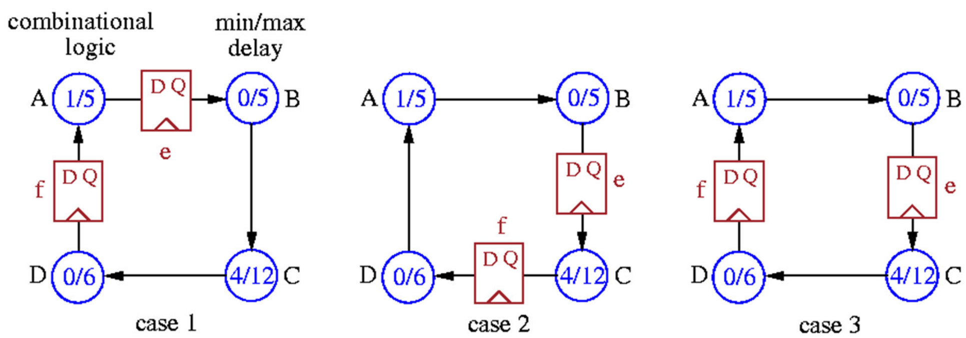
Simultaneous Retiming and Clock Skew Scheduling

- Liu, Papaefthymiou, Friedman: Simultaneous retiming and **useful** clock skew scheduling can further reduce clock period, DAC-99.
- Case 1
  - Zero clock skew: clock period $\phi = 23\tau$.
  - Schedule *e* 4τ earlier than *f*: clock period $\phi = 19\tau$.



Retiming and Clock Skew Scheduling

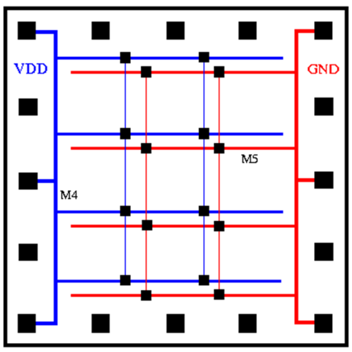
- Case 1
  - Zero clock skew: clock period $\phi = 23\tau$.
  - Schedule *e* 4τ earlier than *f*: clock period $\phi = 19\tau$.
- Case 2
  - Zero clock skew: clock period $\phi = 16\ \tau$.
  - Schedule *f* 1τ earlier than *e*: clock period $\phi = 15\tau$.
- Case 3: optimal effective clock period? No!! Optimal case?
  - Zero clock skew: clock period $\phi = 18\tau$.
  - Schedule *e* 4τ earlier than *f*: clock period $\phi = 14\tau$.



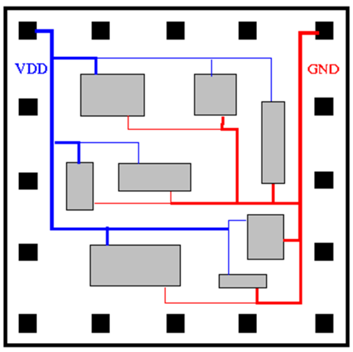
Chang, Huang, Li, Lin, Liu

Power/Ground Routing

- Are usually laid out entirely on metal layers for smaller parasitics.
- Two steps:
  1. **Construction of interconnection topology:** non-crossing power, ground trees.
  2. **Determination of wire widths:** prevent metal migration, keep voltage (IR) drop small, widen wires for more power-consuming modules and higher density current (1.5 mA per μm width for Al). (So area metric?)



grids



interdigitated trees