Routing

- Generates a “loose” route for each net.
- Assigns a list of routing regions to each net without specifying the actual layout of wires.
- Finds the actual geometric layout of each net within the assigned routing regions.

Placement → Global Routing → Detailed Routing → Compaction

Global routing

Detailed routing

Compaction
Routing Constraints

- 100% routing completion + area minimization, under a set of constraints:
  - Placement constraints: usually based on fixed placement
  - Number of routing layers
  - Geometrical constraints: must satisfy design rules
  - Timing constraints (performance-driven routing): must satisfy delay constraints
  - Others

![Two-layer routing](image1.png)  
![Geometrical constraint](image2.png)
Maze Router: Lee Algorithm

- Discussion mainly on single-layer routing
- Strengths
  - Guarantee to find connection between 2 terminals if it exists.
  - Guarantee minimum path.
- Weaknesses
  - Requires large memory for dense layout
  - Slow
- Applications: global routing, detailed routing
Lee Algorithm

- Find a path from \( S \) to \( T \).

- Time & space complexity for an \( M \times N \) grid: \( O(MN) \) (huge!)
Reducing Memory Requirement

• Akers’s Observation (1967)
  - Adjacent labels for \( k \) are either \( k-1 \) or \( k+1 \).
  - Want a labeling scheme such that each label has its preceding label different from its succeeding label.

• **Way 1:** coding sequence 1,2,3,1,2,3,…; states: 1, 2, 3, empty, \textit{blocked} (3 bits required)

• **Way 2:** coding sequence 1,1,2,2,1,1,2,2,…; states: 1, 2, empty, \textit{blocked} (need only 2 bits)
Sequence: 1, 2, 3, 1, 2, 3, ...

Sequence: 1, 1, 2, 2, 1, 1, 2, 2, ...

6
Reducing Running Time

- **Starting point selection:** Choose the point farthest from the center of the grid as the starting point.
- **Double fan-out:** Propagate waves from both the source and the target cells.
- **Framing:** Search inside a rectangle area 10-20% larger than the bounding box containing the source and target.
  - Need to enlarge the rectangle and redo if the search fails.
Connecting Multi-Terminal Nets

• Step 1: Propagate wave from the source $s$ to the closest target.
• Step 2: Mark ALL cells on the path as $s$.
• Step 3: Propagate wave from ALL $s$ cells to the other cells.
• Step 4: Continue until all cells are reached.
• Step 5: Apply heuristics to further reduce the tree cost.
Routing on a Weighted Grid

- Motivation: finding more desirable paths
- \( \text{weight(grid cell)} = \# \text{ of unblocked grid cell segments} - 1 \)
A Routing Example on a Weighted Grid

initialize cell weights

wave propagation

first wave reaches the target

finding other paths

min–cost path found
Hadlock’s Algorithm


- Uses detour number (instead of labeling wavefront in Lee’s router)
  - Detour number, \( d(P) \): # of grid cells directed *away from* its target on path \( P \).
  - \( MD(S,T) \): the Manhattan distance between \( S \) and \( T \).
  - Path length of \( P \), \( l(P) \): \( l(P) = MD(S,T) + 2d(P) \).
  - \( MD(S,T) \) fixed! \( \Rightarrow \) Minimize \( d(P) \) to find the shortest path.
  - For any cell labeled \( i \), label its adjacent unblocked cells *away from* \( Ti + 1 \); label \( i \) otherwise.

- Time and space complexities: \( O(MN) \), but substantially reduces the # of searched cells.

- Finds the shortest path between \( S \) and \( T \).
Hadlock’s Algorithm (cont’d)

![Diagram of Hadlock’s Algorithm](image)
Mikami-Tabuchi’s Algorithm

- Every grid point is an escape point.
Hightower’s Algorithm

- A single escape point on each line segment.
- If a line parallels to the blocked cells, the escape point is placed just past the endpoint of the segment.
Net Ordering

- Net ordering greatly affects routing solutions.
- In the example, we should route net $b$ before net $a$.

route net $a$ before net $b$

route net $b$ before net $a$
Net Ordering (cont’d)

- Order the nets in the ascending order of the # of pins within their bounding boxes.

- Order the nets in the ascending (or descending??) order of their length.
- Order the nets based on their timing criticality.
- A mutually intervening case:

  routing ordering: \( a (0) \rightarrow b (1) \rightarrow d (2) \rightarrow c (6) \)
Rip-Up and Re-routing

- Rip-up and re-routing is required if a global or detailed router fails in routing all nets.
- Two steps in rip-up and re-routing
  1. Identify bottleneck regions, rip off some already routed nets.
  2. Route the blocked connections, and re-route the ripped-up connections.
- Repeat the above steps until all connections are routed or a time limit is exceeded.
Graph Models for Global Routing: Grid Graph

- Each grid cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding grid cells are adjacent to each other.
- The occupied grid cells are represented as filled circles, whereas the others are as clear circles.
Graph Model: Channel Intersection Graph

- Channels are represented as edges.
- Channel intersections are represented as vertices.
- Edge weight represents channel capacity.
- Extended channel intersection graph: terminals are also represented as vertices.
Global Routing Problem

- Given a netlist $N = \{N_1, N_2, \ldots, N_n\}$, a routing graph $G = (V, E)$, find a Steiner tree $T_i$ for each net $N_i$, $1 \leq i \leq n$, such that $U(e_j) \leq c(e_j)$, $\forall e_j \in E$ and $\sum_{i=1}^{n} L(T_i)$ is minimized, where
  - $c(e_j)$: capacity of edge $e_j$;
  - $x_{ij} = 1$ if $e_j$ is in $T_i$; $x_{ij} = 0$ otherwise;
  - $U(e_j) = \sum_{i=1}^{n} x_{ij}$: # of wires that pass through the channel corresponding to edge $e_j$;
  - $L(T_i)$: total wirelength of Steiner tree $T_i$.

- For high-performance, the maximum wirelength $(\max_{i=1}^{n} L(T_i))$ is minimized (or the longest path between two points in $T_i$ is minimized).
Global Routing in Different Design Styles

- **full custom**
  - flexible channels
  - most general problem

- **standard cell**
  - flexible channels
  - fixed feedthroughs

- **gate array**
  - fixed channels

- **FPGA**
  - fixed routing tracks
  - switchbox constraints
Global Routing in Standard Cell

- **Objective**
  - Minimize total channel height.
  - Assignment of feedthroughs.

- For high performance,
  - Minimize the maximum wire length.
  - Minimize the maximum path length.
Global Routing in Gate Array

• Objective
  - Guarantee 100% routability.
• For high performance,
  - Minimize the maximum wire length.
  - Minimize the maximum path length.

Each channel has a capacity of 2 tracks.
Global Routing in FPGA

- Objective
  - Guarantee 100% routability.
  - Consider switch-module architectural constraints.
- For performance-driven routing,
  - Minimize # of switches used.
  - Minimize the maximum wire length.
  - Minimize the maximum path length.

Each channel has a capacity of 2 tracks.
Classification of Global Routing Algorithm

- **Sequential approach**: Assigns priority to nets; routes one net at a time based on its priority (net ordering?).
- **Concurrent approach**: All nets are considered at the same time (complexity?)
Global Routing: Maze Routing

- Routing channels may be modeled by a weighted undirected graph called channel connectivity graph.
- Node $\leftrightarrow$ channel; edge $\leftrightarrow$ two adjacent channels; capacity: $(\text{width}, \text{length})$
Global Routing by Integer Programming

- Suppose that for each net $i$, there are $n_i$ possible trees $t_{i1}, t_{i2}, \ldots, t_{in_i}$ to route the net.
- Constraint I: For each net $i$, only one tree $t_{ij}$ will be selected.
- Constraint II: The capacity of each cell boundary $c_i$ is not exceeded.
- Minimize the total tree cost.
- **Question:** Feasible for practical problem sizes?
  - **Key:** hierarchical approach!

---

**Diagram:**

**an routing instance**

**grid graph**

**a feasible routing**

**trees of net 1**

**trees of net 2**

**trees of net 3**
An Integer-Programming Example

<table>
<thead>
<tr>
<th>Boundary</th>
<th>$t_1^1$</th>
<th>$t_1^2$</th>
<th>$t_2^1$</th>
<th>$t_1^2$</th>
<th>$t_2^2$</th>
<th>$t_2^2$</th>
<th>$t_3^1$</th>
<th>$t_3^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- $g_{ij}$: cost of tree $t_{j}^{i} \Rightarrow g_{1,1} = 2, g_{1,2} = 3, g_{1,3} = 3, g_{2,1} = 2, g_{2,2} = 3, g_{2,3} = 3, g_{3,1} = 2, g_{3,2} = 2.$

Minimize $2x_{1,1} + 3x_{1,2} + 3x_{1,3} + 2x_{2,1} + 3x_{2,2} + 3x_{2,3} + 2x_{3,1} + 2x_{3,2}$

subject to

$$
\begin{align*}
  x_{1,1} + x_{1,2} + x_{1,3} & = 1 \quad (Constraint \ I : t_1^1) \\
  x_{2,1} + x_{2,2} + x_{2,3} & = 1 \quad (Constraint \ I : t_2^1) \\
  x_{3,1} + x_{3,2} & = 1 \quad (Constraint \ I : t_3^1) \\
  x_{1,2} + x_{1,3} + x_{2,1} + x_{2,3} + x_{3,1} & \leq 2 \quad (Constraint \ II : B1) \\
  x_{1,1} + x_{1,3} + x_{2,2} + x_{2,3} + x_{3,1} & \leq 2 \quad (Constraint \ II : B2) \\
  x_{1,2} + x_{1,3} + x_{2,1} + x_{2,2} + x_{3,2} & \leq 2 \quad (Constraint \ II : B3) \\
  x_{1,1} + x_{1,2} + x_{2,2} + x_{2,3} + x_{3,2} & \leq 2 \quad (Constraint \ II : B4) \\
  x_{i,j} & = 0, 1, 1 \leq i, j \leq 3
\end{align*}
$$
Detailed Routing

Detailed routing

channel routing

switchbox routing
Order of Routing Regions

- (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.

• Gridless model:
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 $i$: total # of nets that crosses column $i$.
- Channel density: maximum local density; # of horizontal tracks required $\geq$ channel density.
Channel Routing

• 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 the other net.
• Objective: Channel height is minimized (i.e., channel area is minimized).
Horizontal Constraint Graph (HCG)

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

\[
\begin{array}{cccccccc}
1 & 5 & 2 & 0 & 2 & 1 & 1 & 0 & 3 & 4 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
3 & 0 & 1 & 2 & 5 & 3 & 4 & 0 & 0 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
5 & 4 & 1 & 2 & 3 & \\
\end{array}
\]

*A routing problem and its VCG.*
2-L Channel Routing: Basic Left-Edge Algorithm

- 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.

1 begin
2 $U \leftarrow \{I_1, I_2, \ldots, I_n\}$;
3 $t \leftarrow 0$;
4 while ($U \neq \emptyset$) do
5 \hspace{1em} $t \leftarrow t + 1$;
6 \hspace{1em} $watermark \leftarrow 0$;
7 \hspace{1em} while (there is an $I_j \in U$ s.t. $s_j > watermark$) do
8 \hspace{2em} Pick the interval $I_j \in U$ with $s_j > watermark$, nearest $watermark$;
9 \hspace{2em} $track[j] \leftarrow t$;
10 \hspace{2em} $watermark \leftarrow e_j$;
11 \hspace{2em} $U \leftarrow U - \{I_j\}$;
12 end
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.
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.

1 begin
2 $U \leftarrow \{I_1, I_2, \ldots, I_n\}$;
3 $t \leftarrow 0$;
4 while ($U \neq \emptyset$) do
5 \hspace{1em} $t \leftarrow t + 1$;
6 \hspace{1em} $watermark \leftarrow 0$;
7 \hspace{1em} while (there is an unconstrained $I_j \in U$ s.t. $s_j > watermark$) do
8 \hspace{2em} Pick the interval $I_j \in U$ that is unconstrained,
9 \hspace{2em} with $s_j > watermark$, nearest $watermark$;
10 \hspace{2em} $track[j] \leftarrow t$;
11 \hspace{2em} $watermark \leftarrow e_j$;
12 \hspace{1em} $U \leftarrow U - \{I_j\}$;
13 end
Constrained Left-Edge Example

- 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

- 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.
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.
Over-the-Cell Routing

- Routing over the cell rows is possible due to the limited use of the 2nd (M2) metal layers within the cells.
- Divide the over-the-cell routing problem into 3 steps: (1) routing over the cell, (2) choosing the net segments, and (3) routing within the channel.
Clock Routing

- **Clock skew** is defined as the difference in the minimum and the maximum arrival time of the clock.

  \[
  \text{clock skew} = 20 - 9 = 11
  \]

- Route clock nets such that
  - Clock signals arrive simultaneously
  - Clock delay is minimized
    * Other issues: total wirelength, power consumption.
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 \( \max_{i,j \in P} |t(0,i) - t(0,j)| \) (skew) and \( \max_{i \in P} t(0,i) \) (delay) are both minimized.

- Pathlength-based approaches

- RC-delay based approaches:
H-Tree Based Algorithm


\[ \text{H-tree over 4 points} \]

\[ \text{H-tree over 16 points} \]
The MMM Algorithm

- Each clock pin is represented as a point in the region, $S$.
- The region is partitioned into two subregions, $S_L$ and $S_R$.
- The center of mass is computed for each subregion.
- The center of mass of the region $S$ is connected to each of the centers of mass of subregion $S_L$ and $S_R$.
- The subregions $S_L$ and $S_R$ are then recursively split in $Y$-direction.
- The above steps are repeated with alternate splitting in $X$- and $Y$-direction.
- Time complexity: $O(n \log n)$.

![Diagram of the MMM Algorithm](image)
The Geometric Matching Algorithm

- 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 the same.
- The above steps are repeated for each segment.
- Apply $H$-flipping to further reduce clock skew (and to handle edges intersection).
Delay Calculation

- Need to consider a more accurate delay model for general circuits.
- **RC Delay:** The delay caused by wires is due to their capacitance and resistance. \( R \propto \frac{l}{wh}; \ C \propto wl \)
- Lumped circuit approximations for distributed RC lines: \( \pi \)-model, \( T \)-model, \( L \)-model.

\[
\begin{align*}
\text{a lumped wire} & \quad \text{\( \Pi \)-model} & \quad \text{\( T \)-model} & \quad \text{\( L \)-model} \\
\end{align*}
\]

- RC delay: A to B: \( D_{AB} = R_1 \left( \frac{C_1}{2} + C_2 + C_3 \right) \); B to C: \( D_{BC} = R_2 \left( \frac{C_2}{2} \right) \); B to D: \( D_{BD} = R_3 \left( \frac{C_3}{2} \right) \)
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 $t_{ij} = \sum_{j \in \delta(i,j)} r_j C_j$.
- $t_{03} = r_0 (c_1 + c_2 + c_3 + c_4 + c_1^s + c_2^s + c_3^s) + r_2 (\frac{c_2}{2} + c_3 + c_4 + c_2^s + c_3^s) + r_4 (\frac{c_4}{2} + c_3^s)$.

![Clock Tree Diagram](image)

![Delay Model Diagram](image)
Exact Zero Skew Algorithm

- To ensure the delay from the tapping point to leaf nodes of subtrees $T_1$ and $T_2$ being equal, it requires that
  \[ r_1\left(\frac{C_1}{2} + C_1\right) + t_1 = r_2\left(\frac{C_2}{2} + C_2\right) + t_2 \]
- Solving the above equation, we have
  \[ x = \frac{(t_2 - t_1) + \alpha l(C_2 + \frac{\beta l}{2})}{\alpha l(\beta l + C_1 + C_2)} \]
- Where $\alpha$ and $\beta$ are the per unit values of resistance and capacitance, $l$ the length of the interconnecting wire, $r_1 = \alpha x l$, $c_1 = \beta x l$, $r_2 = \alpha (1-x)l$, $c_2 = \beta (1-x)l$. 

\[ x = \frac{(t_2 - t_1) + \alpha l(C_2 + \frac{\beta l}{2})}{\alpha l(\beta l + C_1 + C_2)} \]
Zero-Skew Computation

- **Balance delays:** 
  \[ r_1 \left( \frac{c_1}{2} + C_1 \right) + t_1 = r_2 \left( \frac{c_2}{2} + C_2 \right) + t_2. \]
  \[ (t_2 - t_1) + \alpha l (C_2 + \frac{\beta l}{2}) \]

- **Compute tapping points:** 
  \[ x = \frac{\alpha l (\beta l + c_1 + c_2)}{\alpha l (\beta l + c_1 + c_2)} \], \( \alpha (\beta) \): per unit values of resistance (capacitance); \( l \): length of the wire; \( r_1 = \alpha xl \), \( c_1 = \beta xl \); \( r_2 = \alpha (1-x)l \), \( c_2 = \beta (1-x)l \).

- If \( x \notin [0, 1] \), we need **snaking** to find the tapping point.
- **Exp:** \( \alpha = 0.1 \Omega/\text{unit}, \ \beta = 0.2 F/\text{unit} \). (Find tapping points \( E \) for \( A \) and \( B \), \( F \) for \( C \) and \( D \), and \( G \) for \( E \) and \( F \).

\[ \begin{align*}
\text{D} &= (5, 15), \ C_D = 2F \\
\text{C} &= (0, 10), \ C_C = 1F \\
\text{F} &= (5, 11), \ C_F = 2 + 1 + 0.2 \times 10 = 5F \\
\text{E} &= (10, 6), \ C_E = 16 + 10 + 0.2 \times 20 = 30F \\
\text{G} &= (14, 6), \ C_G = 16 + 20 = 36F \\
\text{B} &= (22, 6), \ C_B = 10F \\
\text{A} &= (8, 0), \ C_A = 16F
\end{align*} \]
Delay Computation for Buffered Wires

- Wire: $\alpha = 0.1 \Omega/\text{unit size}$, $\beta = 0.2 F/\text{unit size}$; buffer: $\alpha' = 0.2 \Omega/\text{unit size}$, $\beta' = 1 F/\text{unit size}$; unit-sized wire and buffer.