

Design Rules

- The fabrication process will suffer from **tolerances**
- Chip features will have a practical **minimum size** to allow them to be fabricated reliably enough (with high enough **yield**)
- This is captured into a set of precise **Design Rules**
- Modern processes have terribly complex set of design rules as a compromise between **flexibility** and **manufacturability**

http://en.wikipedia.org/wiki/Design_rule_checking

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 2

Design Rules

Design rules: restrictions on the mask patterns to increase the probability of successful fabrication.

Compromise between

- Density
- Yield
- Ease of use

Patterns and design rules are expressed in either nanometers or integer multiples of 'gridsize' λ . The types of the most common design rules:

- minimum-width rules (a)
- minimum-separation rules (b, c, d)
- minimum-overlap rules (e)

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 3

λ -based Transistor Rules

- Illustration only

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 4

More DR Examples (Intra-layer)

Same potential Different potential

Intra-layer design rules: minimum dimensions and spacings

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 5

Vias & Contacts.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 6

Symbolic Layout

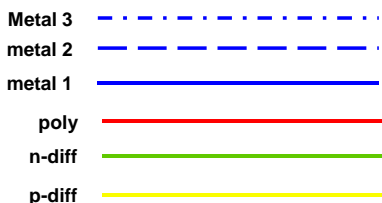
A layout is **symbolic** when not all mask patterns have full specification:

- Single symbols are used to represent elements located in several layers, e.g. transistors, contact cuts.
- The **length, width** or **layer** of a wire or other layout element might be left unspecified.
- Mask layers not directly related to the functionality of the circuit do not need to be specified, e.g. n-well, p-well.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 7

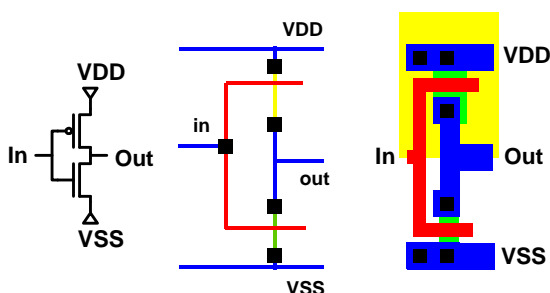
Symbolic Layout / Stick diagrams

- A stick diagram is a **cartoon** of a layout.
- Does show components/vias but only **relative placement**.
- Does **not** show **exact placement**, transistor sizes, wire lengths, wire widths, tub boundaries, some special components.



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 8

Inverter Schematic, Stick Diagram, Layout.



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 9

Compaction and its Applications

A **compaction program** or **compactor** generates layout at the mask level. It attempts to make the layout **as dense as possible**.

Applications of compaction:

- **Area minimization:** removing redundant space in layout at the mask level.
- **Layout compilation:** generation of mask-level layout from symbolic layout.
- **Redesign:** automatic removal of design-rule violations.
- **Rescaling:** converting mask-level layout from one technology to another.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 10

Aspects of Compaction

Dimension:

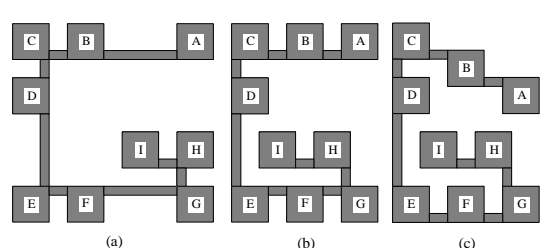
- **1-dimensional (1D):** layout elements only move or shrink in one dimension (x or y). Often sequentially performed first in the x-dimension and then in the y-dimension (or vice versa).
- **2-dimensional (2D):** layout elements move and shrink simultaneously in two dimensions.

Complexity:

- **1D-compaction** can be done **efficiently**;
- **2D-compaction** is **NP-hard**.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 11

1D Compaction: X Followed by Y



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 12

1D Compaction: X Followed by Y

(a) (d) (e)

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 13

2D Compaction

(a) (f)

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 14

Comparison

(a) (c) (e) (f)

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 15

Inequalities for Distance Constraints

Minimum-distance design rules can be expressed as inequalities.

For example:

$$x_2 - x_1 \geq a$$

$$x_3 - x_2 \geq b$$

$$x_3 - x_6 \geq b$$

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 16

The Constraint Graph

The inequalities can be used to construct a **constraint graph** $G(V, E)$:

- There is a **vertex** v_i for each **variable** x_i .
- For each inequality of the form $x_j - x_i \geq d_{ij}$, there is an **edge** (v_i, v_j) with **weight** d_{ij} .
- There is an extra **source vertex**, v_0 ; it is located at $x = 0$; all other vertices are at its right.
- If all the inequalities express minimum-distance constraints, the graph is acyclic. It is a **DAG**, a **directed acyclic graph**.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 17

The Constraint Graph

Example rules

- All widths $\geq a$
- All spacings $\geq b$

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 18

Maximum-Distance Constraints

Sometimes the distance of layout elements is bounded by a maximum, e.g. when the user wants a maximum wire width.

■ A maximum distance constraint gives an inequality of the form:

$$x_j - x_i \leq c_{ij}$$

$$\Leftrightarrow$$

$$x_i - x_j \geq -c_{ij}$$

■ Consequence for the constraint graph: **backward edge**

(v_i, v_j) with weight $d_{ij} = -c_{ij}$.
The graph is not acyclic anymore.

$\rightarrow a \leftarrow$ min. distance
 $\rightarrow b \leftarrow$ max. distance

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 19

Longest Path Problem

■ Given graph with weighted edges, assign positions to nodes that is consistent with all the edges

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 20

Longest-Path Algorithm for DAGs

```

longest-path(G)
{
  for (i ← 1; i ≤ n; i ← i + 1)
    p_i ← "in-degree of v_i";
  Q ← {v_0};
  while (Q ≠ ∅) {
    v_i ← "any element from Q";
    Q ← Q \ {v_i};
    for each v_j "such that" (v_i, v_j) ∈ E { // for each outgoing edge
      x_j ← max(x_j, x_i + d_ij); // set max distance for x_j
      p_j ← p_j - 1;
      if (p_j ≤ 0) // if all incoming edges have been processed
        Q ← Q ∪ {v_j}; // can process its outgoing edges
    }
  }
}
            
```

```

main ()
{
  for (i ← 0; i ≤ n; i ← i + 1)
    x_i ← 0;
  longest-path(G);
}
            
```

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 21

```

while (Q ≠ ∅) {
  v_i ← "any element from Q";
  Q ← Q \ {v_i};
  for each v_j "such that" (v_i, v_j) ∈ E {
    x_j ← max(x_j, x_i + d_ij);
    p_j ← p_j - 1;
    if (p_j ≤ 0)
      Q ← Q ∪ {v_j};
  }
}
            
```

Q	p ₁	p ₂	p ₃	p ₄	p ₅	x ₁	x ₂	x ₃	x ₄	x ₅
"not initialized"	1	2	1	2	1	0	0	0	0	0
{v ₀ }	0	1	1	2	1	1	5	0	0	0
{v ₁ }	0	0	1	2	0	1	5	0	0	3
{v ₂ , v ₅ }	0	0	0	1	0	1	5	6	6	3
{v ₃ , v ₅ }	0	0	0	1	0	1	5	6	6	3
{v ₅ }	0	0	0	0	0	1	5	6	7	3
{v ₄ }	0	0	0	0	0	1	5	6	7	3

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 22

Longest-Path Algorithm for DAGs

```

longest-path(G)
{
  for (i ← 1; i ≤ n; i ← i + 1)
    p_i ← "in-degree of v_i";
  Q ← {v_0};
  while (Q ≠ ∅) {
    v_i ← "any element from Q";
    Q ← Q \ {v_i};
    for each v_j "such that" (v_i, v_j) ∈ E {
      x_j ← max(x_j, x_i + d_ij);
      p_j ← p_j - 1;
      if (p_j ≤ 0)
        Q ← Q ∪ {v_j};
    }
  }
}
            
```

```

main ()
{
  for (i ← 0; i ≤ n; i ← i + 1)
    x_i ← 0;
  longest-path(G);
}
            
```

Q: What is the worst-case time complexity of this algorithm?
A: $O(|E|)$

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 23

Longest-Paths in Cyclic Graphs

Constraint-graph compaction with maximum-distance constraints requires solving the longest-path problem in cyclic graphs.

Two cases are distinguished:

- There are **positive cycles** ⇒ the problem is **NP-hard**; however, a positive cycle corresponds to a set of conflicting constraints. The best to be done is to detect the cycles.
- All cycles are **negative**: polynomial-time algorithms exist.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 24

The Liao-Wong Algorithm (1)

Main ideas:

- Split the edge set E of the constraint graph into **two subsets**:
 - forward edges** E_f : related to minimum-distance constraints,
 - backward edges** E_b : related to maximum-distance constraints.
- The graph $G(V, E)$ is **acyclic**; the minimum distance for each vertex can be computed with the procedure "longest-path".
- Repeat until convergence:
 - update minimum distances by processing the edges from E_b ,
 - call "longest-path" for $G(V, E_f)$.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 25

The Liao-Wong Algorithm (2)

```

count ← 0;
for (i ← 1; i ≤ n; i ← i + 1)
    x_i ← -∞;
x_0 ← 0;

do { flag ← 0;
    longest-path(G_f); // G_f = G(V, E_f)
    for each (v_i, v_j) ∈ E_b
        if (x_j < x_i + d_ij) {
            x_j ← x_i + d_ij; // backward edge reduces distance
            flag ← 1; // not yet converged
        }
    count ← count + 1;
    if (count > |E_b| && flag)
        error("positi ve cycle")
}
while (flag); // while not converged
    
```

E_f = forward edges
E_b = backward edges

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 26

The Liao-Wong Algorithm (3).

Step	x ₁	x ₂	x ₃	x ₄	x ₅
Initialize	-∞	-∞	-∞	-∞	-∞
Forward1	1	5	6	7	3
Backward1	2	5	6	7	3
Forward2	2	5	6	8	4
Backward2	2	5	7	8	4
Forward3	2	5	7	8	4
Backward3	2	5	7	8	4

- After first forward iteration, the max-3 constraint between v_2 and v_3 is violated
- Corrected after first backward iteration
- But then v_4 and v_5 are too close to v_1
- Etc.

Q: What is the worst-case time complexity of this algorithm?

A: $O(|E_b| \times |E|)$.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 27

The Bellman-Ford Algorithm (1)

```

for (i ← 1; i ≤ n; i ← i + 1)
    x_i ← -∞;
x_0 ← 0;
count ← 0;
S_1 ← {v_0}; // current wavefront
S_2 ← ∅; // wavefront for next iter
while (count ≤ n && S_1 ≠ ∅) {
    for each v_j ∈ S_1
        for each v_j "such that" (v_i, v_j) ∈ E
            if (x_j < x_i + d_ij) {
                x_j ← x_i + d_ij;
                S_2 ← S_2 ∪ {v_j}
            }
    S_1 ← S_2; // swap wavefront
    S_2 ← ∅;
    count ← count + 1; // counter to detect positive loops
}
if (count > n)
    error("positi ve cycle");
    
```

<http://en.wikipedia.org/wiki/Bellman-ford>
Repeated wavefront propagation

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 28

```

while (count ≤ n && S_1 ≠ ∅) {
    for each v_i ∈ S_1
        for each v_j "such that" (v_i, v_j) ∈ E
            if (x_j < x_i + d_ij) {
                x_j ← x_i + d_ij;
                S_2 ← S_2 ∪ {v_j}
            }
    S_1 ← S_2;
    S_2 ← ∅;
    count ← count + 1;
}
    
```

S ₁	x ₁	x ₂	x ₃	x ₄	x ₅
"notinitialized"	-∞	-∞	-∞	-∞	-∞
{v ₀ }	1	5	-∞	-∞	-∞
{v ₁ , v ₂ }	2	5	6	6	3
{v ₁ , v ₃ , v ₄ , v ₅ }	2	5	6	7	4
{v ₄ , v ₅ }	2	5	6	8	4
{v ₄ }	2	5	7	8	4
{v ₃ }	2	5	7	8	4

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 29

Longest Path vs Bellman-Ford

Kernel of algorithms

```

while (Q ≠ ∅) {
    v_i ← "any element from Q";
    Q ← Q \ {v_i};
    for each v_j "such that" (v_i, v_j) ∈ E {
        x_j ← max(x_j, x_i + d_ij);
        p_j ← p_j - 1;
        if (p_j ≤ 0)
            Q ← Q ∪ {v_j};
    }
}
                
```

Longest Path

Worst case: $O(|E_b| \times |E|)$.

```

while (count ≤ n && S_1 ≠ ∅) {
    for each v_i ∈ S_1
        for each v_j "such that" (v_i, v_j) ∈ E
            if (x_j < x_i + d_ij) {
                x_j ← x_i + d_ij;
                S_2 ← S_2 ∪ {v_j}
            }
    S_1 ← S_2;
    S_2 ← ∅;
    count ← count + 1;
}
                
```

Bellman-Ford

Worst-case: $O(n \times |E|) = O(n^3)$.

Average-case: $O(n^{1.5})$. (Schiele)

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 30

Longest and Shortest Paths

- Longest paths become shortest paths and vice versa when edge weights are multiplied by -1 .
- Situation in DAGs: both the longest and shortest path problems can be solved in linear time.
- Situation in cyclic directed graphs:
 - + All weights are positive: shortest-path problem in P (Dijkstra), longest-path problem is NP-complete.
 - + All weights are negative: longest-path problem in P (Dijkstra), shortest-path problem is NP-complete.
 - + No positive cycles: longest-path problem is in P
 - + No negative cycles: shortest-path problem is in P.
 - + Otherwise: problem is NP-complete.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 31

Remarks Constraint-Graph Compaction

- The algorithms mentioned only compute the left-most position for each layout element. All elements outside the **critical paths** also have a **right-most position**. It is interesting to find the best position within this interval with respect to some cost function.
- The quality of the layout can further be improved by **automatic jog insertion**.
- A method to reduce complexity is **hierarchical compaction**.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 32

Constraint Generation

- The constraint graph is not directly available after layout design. It must be computed.
- The set of constraints should be **irredundant (why?)** and generated efficiently.

- Doenhardt and Lengauer have proposed a method for irredundant constraint generation with complexity $O(n \log n)$.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 33

Virtual Grid Compaction

Features:

- 1D method.
- All layout elements are associated to horizontal and vertical grid lines.
- Initially the distance between grid lines is unspecified.
- The distance between two subsequent grid lines is computed by taking the maximum separation imposed by pairs of elements on different grid lines.
- Disadvantage: rigid as elements initially located on one line always remain aligned.

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 34