BonnCell

Automatic Layout of Leaf Cells

Stefan Hougardy, Tim Nieberg, Jan Schneider

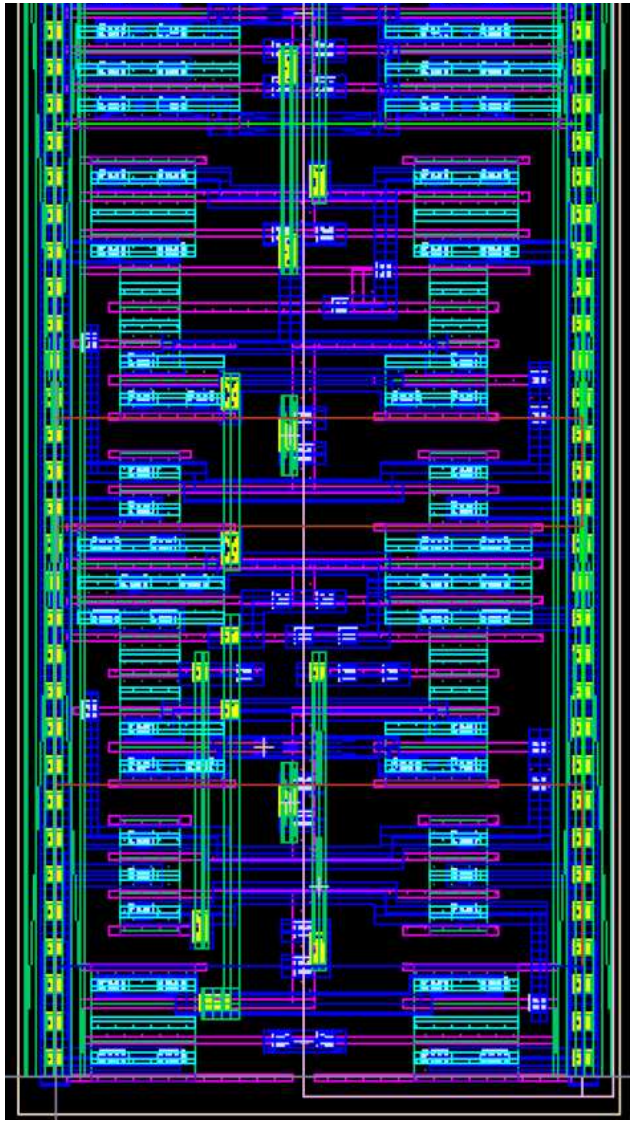
Research Institute for Discrete Mathematics
University of Bonn

January 24, 2013

Outline

- Introduction
- BonnCell Placement
- BonnCell Routing
- Status and Results

The Leaf Cell Layout Problem



Input: A netlist, the cell image

Goal : A "good" solution

- Place the FETs
  - Choose number of fingers
  - Decide to swap FETs
  - Obey all placement rules
  - Guarantee routability
- Routing
  - Find an LVS-clean routing
  - Should be almost DRC-clean
  - Minimize M2 usage

Leaf Cell Layout – Overview

- Leaf cell layout is used in highly optimized arrays
- So far has been done manually
- Example: a next generation Core SRAM
  - Expected time for first pass layout: 3 months or more
  - Actual time needed: 1.5 months with BonnCell
- By now, BonnCell is used worldwide
  - IBM Design Centers in Germany, USA, Israel, India

Main Features of BonnCell

- Generates **1-dimensional and 2-dimensional** cell layouts
- Interleaving of stacks
- Instances can be **non-dual**, non-series-parallel, non-planar
- Unequal number of P and N devices possible
- **Dynamic** placement and **dynamic** folding
  - Folding of multiple FETs
  - First time done by an automatic tool
- We do not enforce and-stack-clustering
- Routing does not rely on structured placement
- DRC rules are taken into account during routing

Previous Work

Bar-Yehuda, Feldman, Pinter, Wimer (1989)

"Depth-First Search and Dynamic Programming Algorithms for Efficient CMOS Cell Generation"

- Similar basic approach as BonnCell (also **branch & bound**)
- Similar target function
- Much more restricted:
  - Strictly 1-dimensional
  - All FETs have exactly **1 finger**
  - Only very **structured routings** are considered
- **No optimum** is found due to simple flipping strategy

Previous Work

Iizuka, Ikeda, Asada (2006)

"Exact Minimum-Width Multi-Row Transistor Placement for Dual and Non-Dual CMOS Cells"

- Different approach: Transformation to CNF, applying a **SAT solver**
- Similar flow: Increase cell size until feasibility is reached
- Supports **2-dimensional** cells
- Much more restricted:
  - Strictly 1-dimensional (i.e. **no interleaving**)
  - All FETs have exactly **1 finger**
  - **Routability** is hardly addressed

Leaf Cell Placement in BonnCell

Leaf Cell Placement Problem

Given an **image** and a netlist, containing

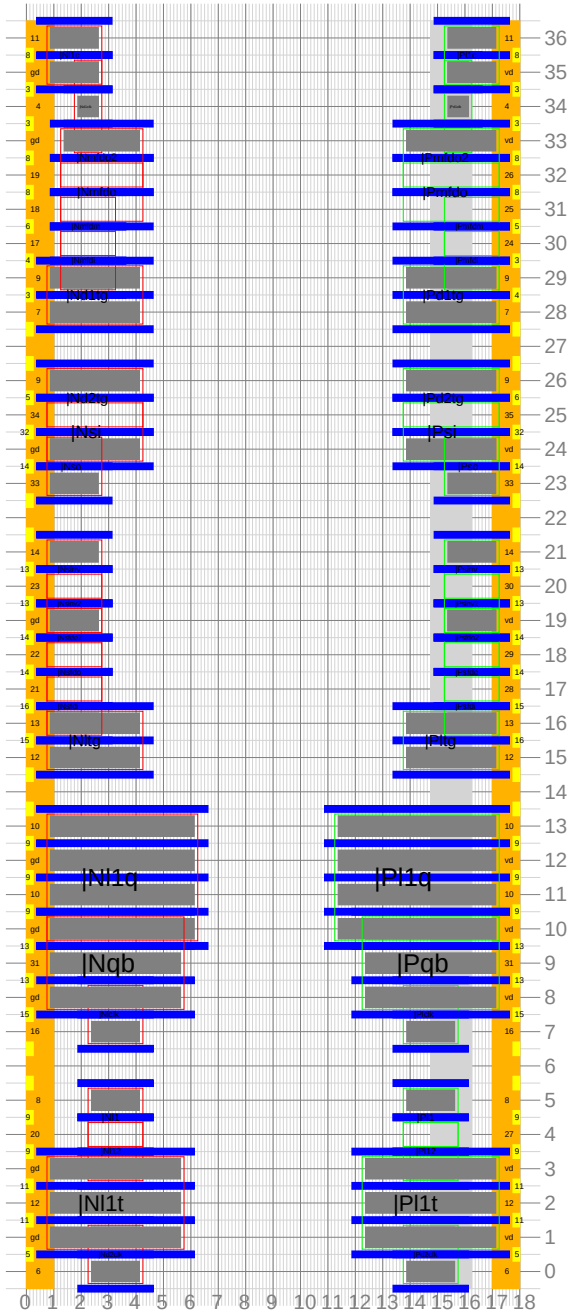
- **FETs** and **nets**,

assign to each FET

- a **location**,
- a **swap status**, and
- a number of **fingers**,

so that the placement

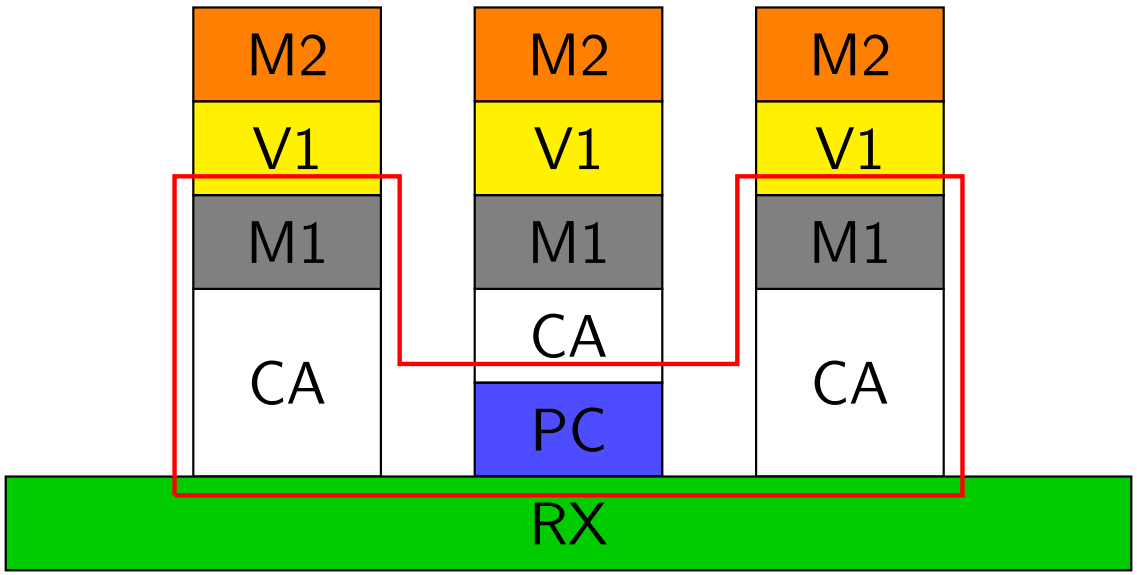
- meets all placement constraints,
- is **routable**, and
- is **optimal**.



Anatomy of a Leaf Cell

Cross Section of a FET

- A leaf cell spans 3 metal layers: **PC, M1, M2**
- **Gates** are on PC
- **Source** and **drain** are on M1

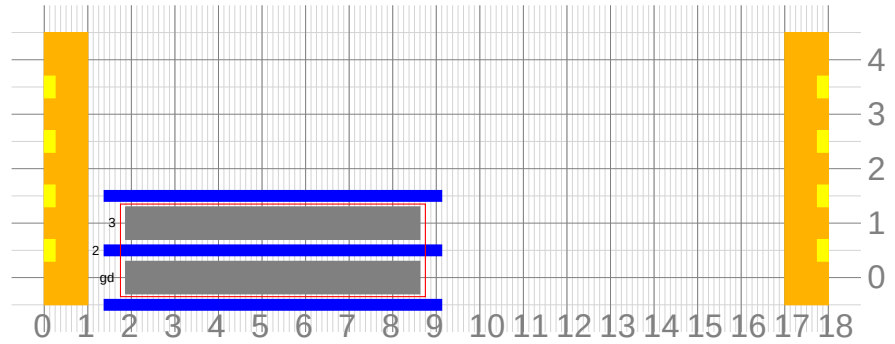


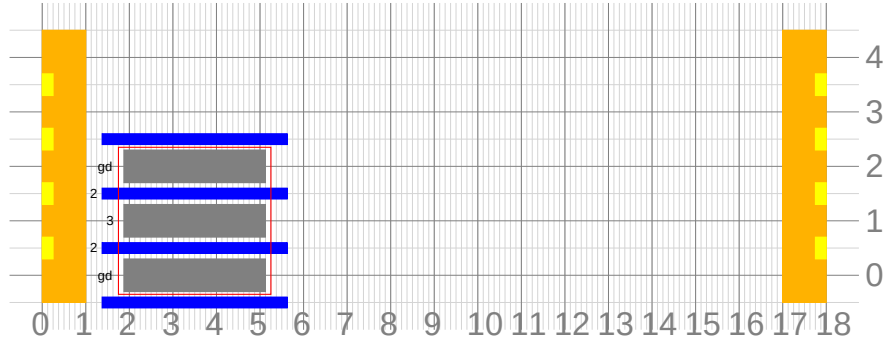
Cross section of a FET

Placing a FET

Placing a FET allows many degrees of freedom

- x-coordinate
- y-coordinate
- Number of fingers
- Swap status



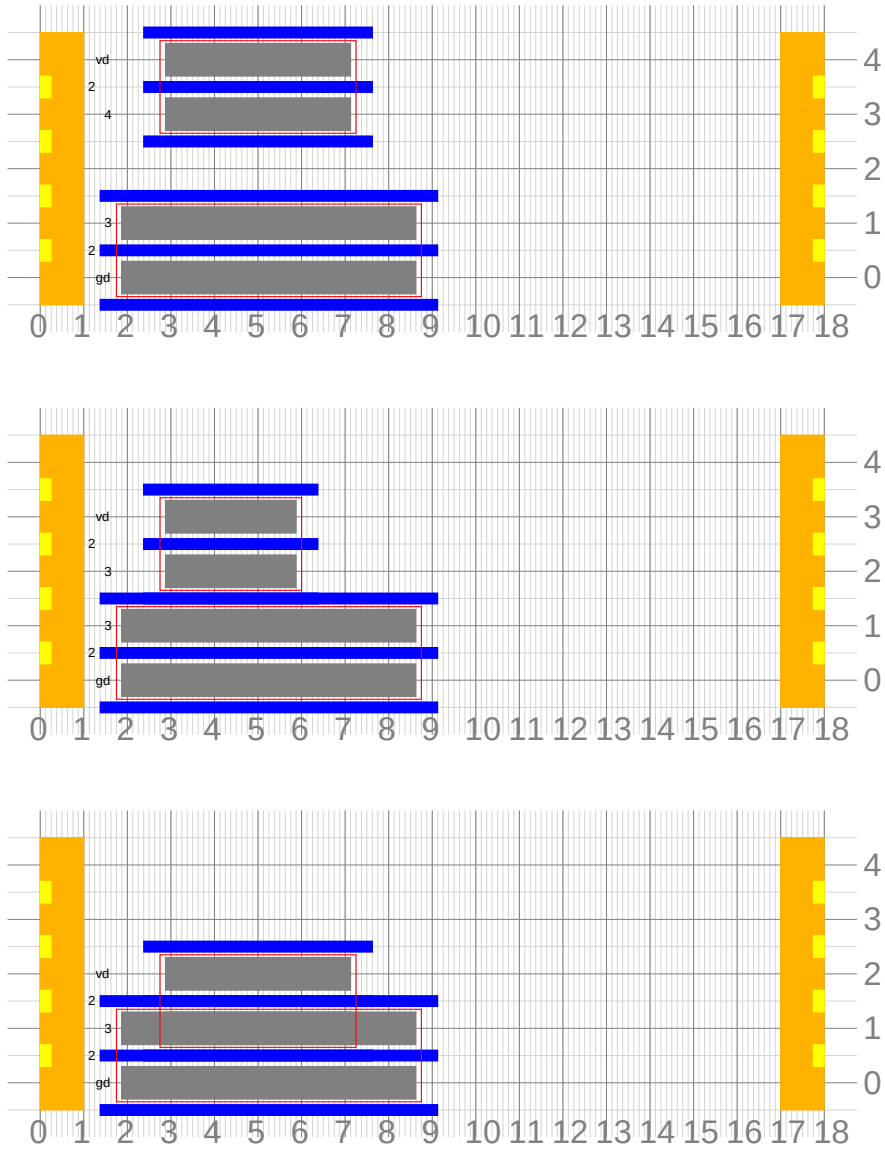


Which Placement Constraints?

Depending on the FETs' properties, they can/must be placed in various ways:

- With a gap
- Abutting
- **Overlapping**

Plus more technology-dependent placement constraints



What is Optimality?

Optimality in BonnCell

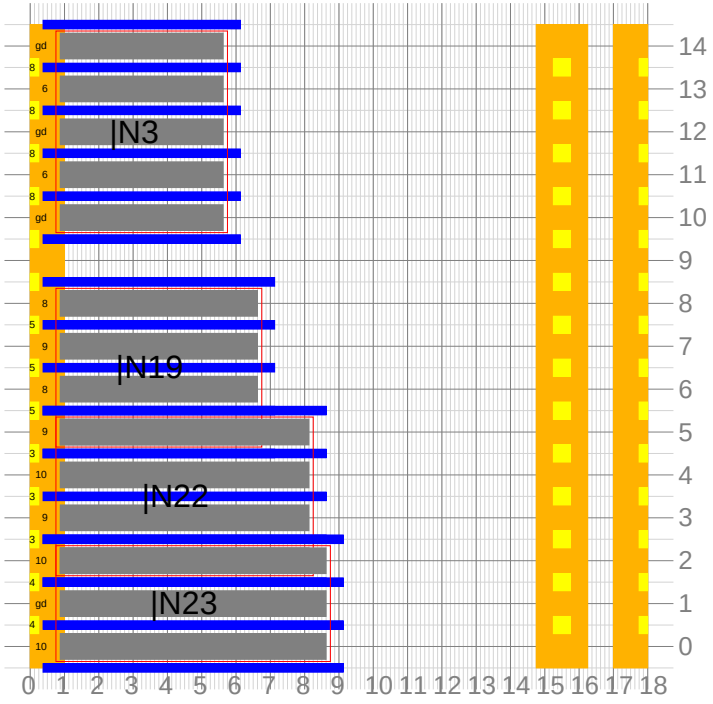
- Our **quality measure** is a quadruple $(h, ggNL, wNL, \sigma)$ with:
  - $h :=$ cell height in #tracks
  - $ggNL :=$ sum of sizes of net's **gate intervals**
    - (where "gate interval" is the interval between bottommost and topmost **gate**)
  - $wNL :=$ **weighted** sum of sizes of **net intervals**
    - (where "net interval" is the interval between bottommost and topmost **terminal**)
  - $\sigma :=$ free **routing** space
    - (i.e. free spaces on every track summed up)
- "Optimal" placement is the placement with the **lexicographically best** quality measure
- Measure proved to be **good in practice**

Algorithm Overview

Algorithm Overview

- Assume that FETs are placed in **2 stacks**
- BonnCell Placement runs in **2 phases**:
  - **Phase 1:** Compute an optimal placement with the **restriction** that both stacks can be divided by a **vertical line**
  - **Phase 2:** Compute an optimal placement **without the restriction**
- On a **timeout**, the best solution found so far is returned

1-Stack Algorithm



1-Stack Subroutine

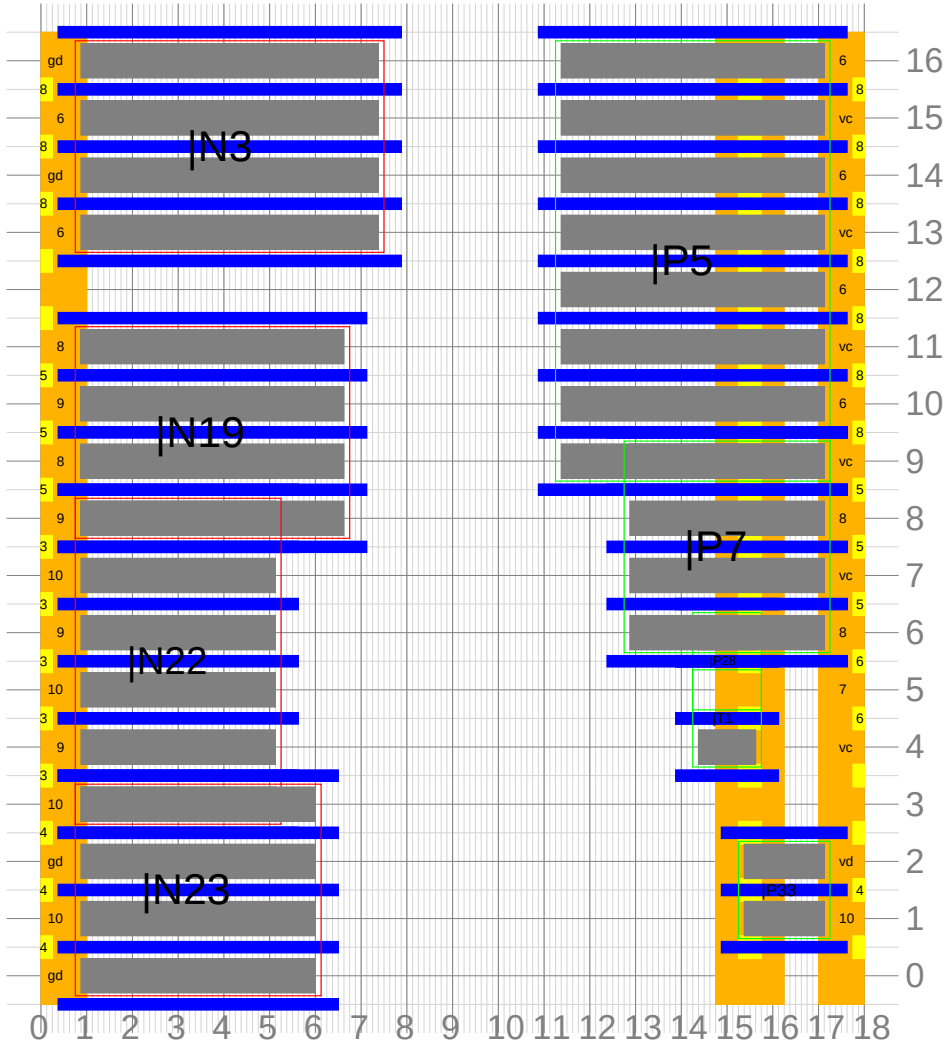
- Important subroutine of the main algorithm
- Find one (or all) **track-minimal** placements of a **single stack**
- Can be solved by simple **enumeration**

2-Stack Flow

2-Stack Flow – Phase 1

- Assume that both stacks are divided by a vertical line
- **Step 1:** Choose x coordinate for this line
- **Step 2:** Find **single** track-minimal 1-stack placements for both stacks independently
- **Step 3:** For **all** track-minimal placements of "bigger" stack
  - ... find **all** placements of "smaller" stack
  - ... which do not exceed the big stack's height
  - ... and **evaluate their quality**
  - If a placement is best so far, **save it**

2-Stack Flow



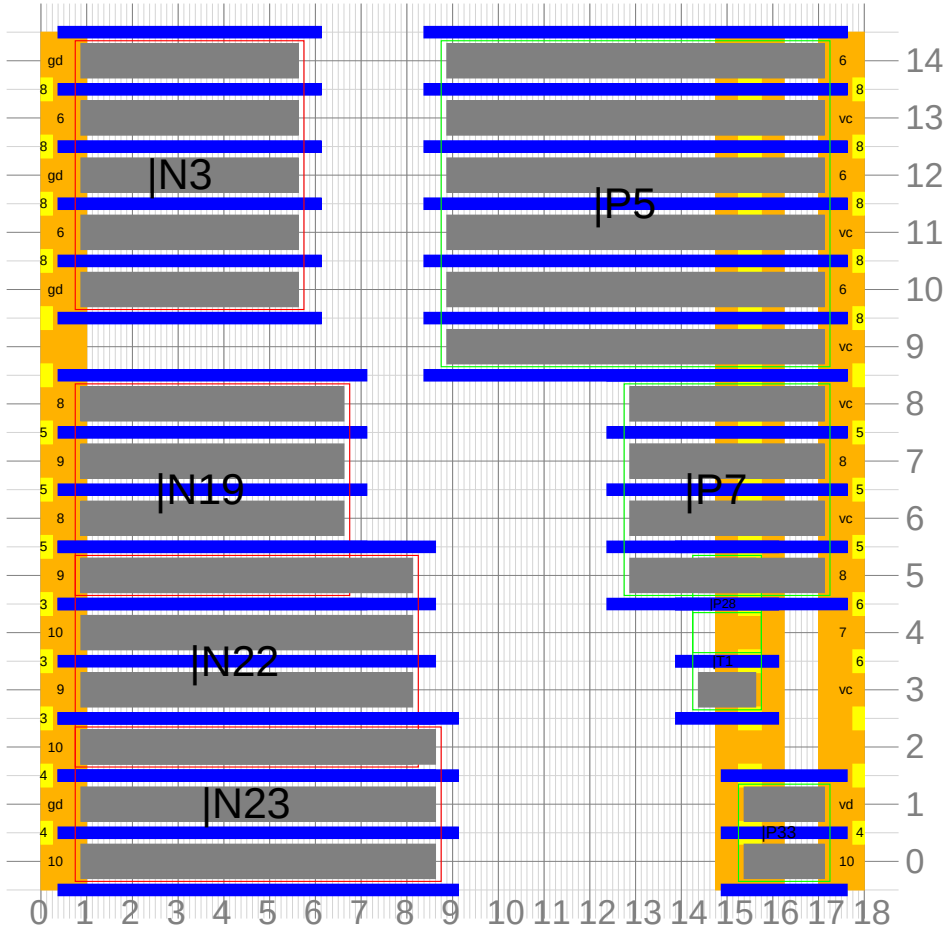
Optimal placement after phase 1.

2-Stack Flow

2-Stack Flow – Phase 2

- Do not assume the vertical line and allow **interleaved placements**
- Problem: Height of optimal solution is **unknown**
  - **Solution:** Start to run phase 2 with cell height 0
  - Run phase 2 for **increasing height** as long as no solution exists
- **Step 1:** For **all** placements of "bigger" stack
  - ... find **all legal** placements of "smaller" stack
  - ... which do not exceed the big stack's height
  - ... and **evaluate their quality**
  - If a placement is best so far, **save it**
- In case of **time-out**, return **best result** so far (including phase 1)

2-Stack Flow



Optimal placement after phase 2.

Branch & Bound

Branch & Bound

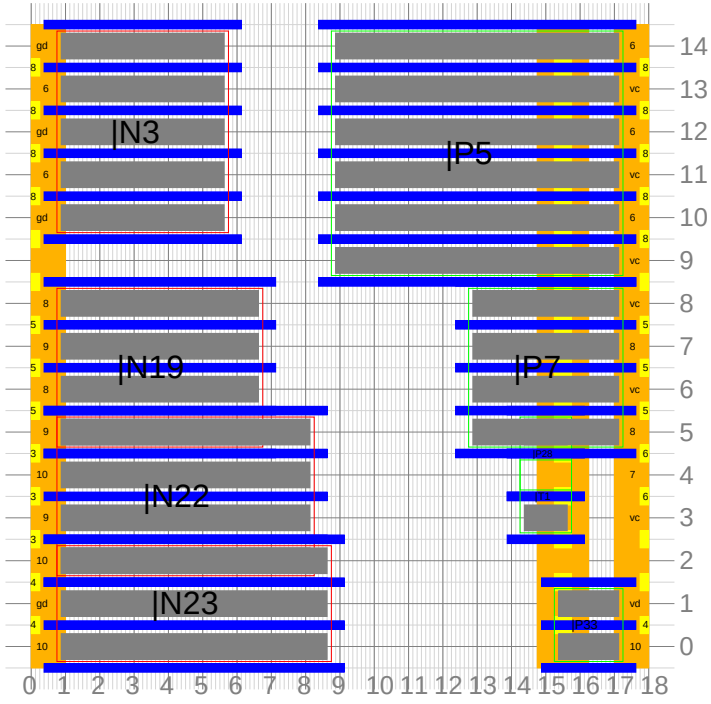
- The **1-Stack subroutine** can be implemented **recursively**
  - StackPlacer(S, k) places stack S, assuming that the lower k FETs are already placed
- A **lower bound** `lb` can be computed in every step
  - `lb` $:=$ `cur_height` $+$ `remaining_fingers`
- Stop extending the stack if `lb` $>$ `best_height`

Branch & Bound – Improvements

Improvements

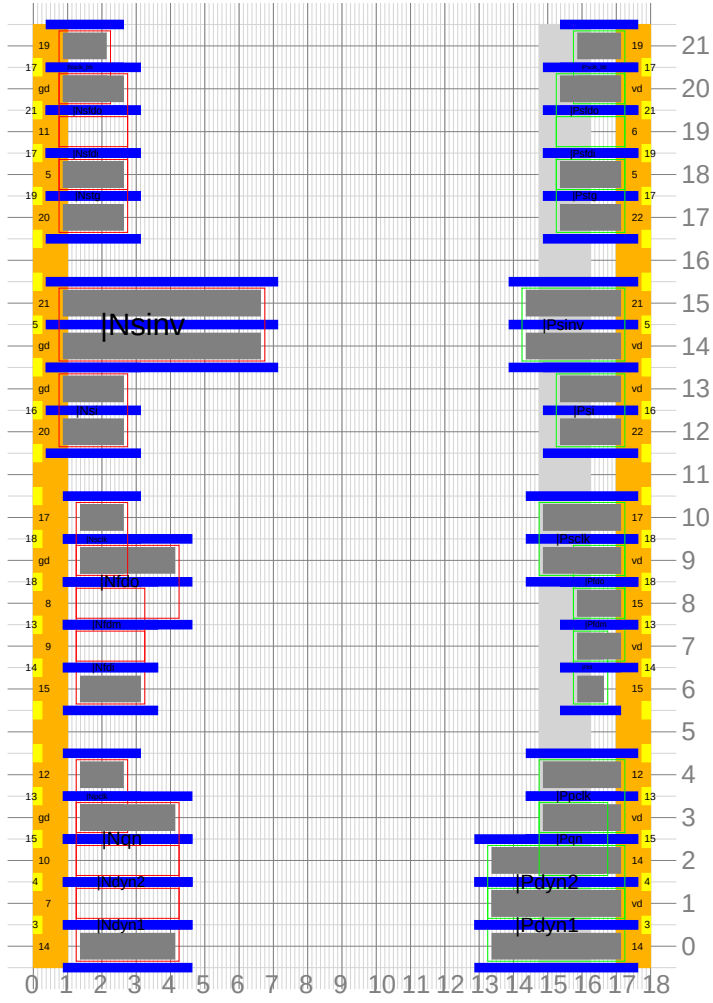
- Use the number of nets accessing an **odd number** of remaining FETs in the **lower bound computation**
- In a first step, look for a **single** height-optimal solution by bounding on $\texttt{lb} \geq \texttt{best\_height}$
  - Additionally, only look for solutions with a specific structure
- Ininitialize **upper bound** with optimal value
- More bounding possibilities in **2-Stack flow**
  - Netlength-based bounding of secondary stack
  - Smart pruning for primary stack
  - Look-ahead **netlength estimation**
- Some **DRC rules** can be implemented as additional bounding steps

Runtime Analysis



9 FETs, 15 tracks

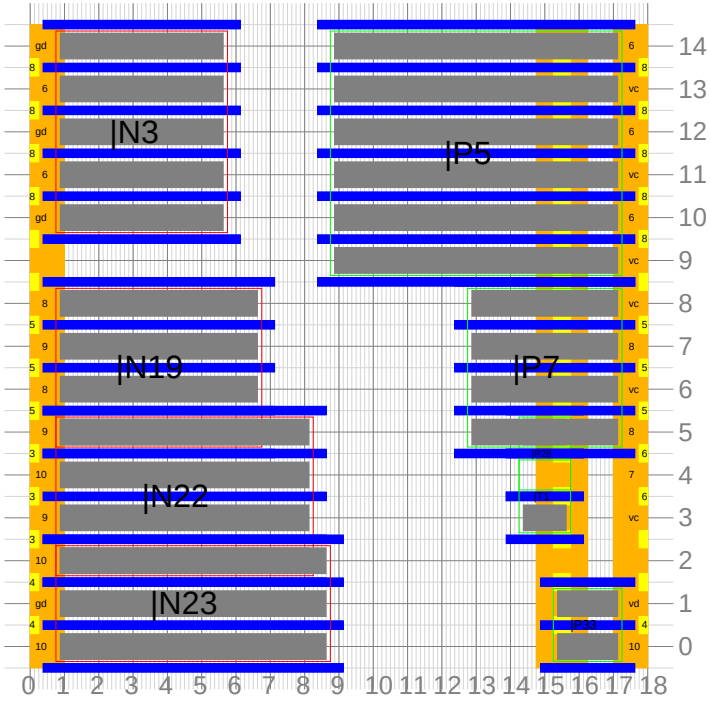
Estimated runtime: 3 years



28 FETs, 22 tracks

Estimated runtime: 10^{30} years

Runtime Analysis

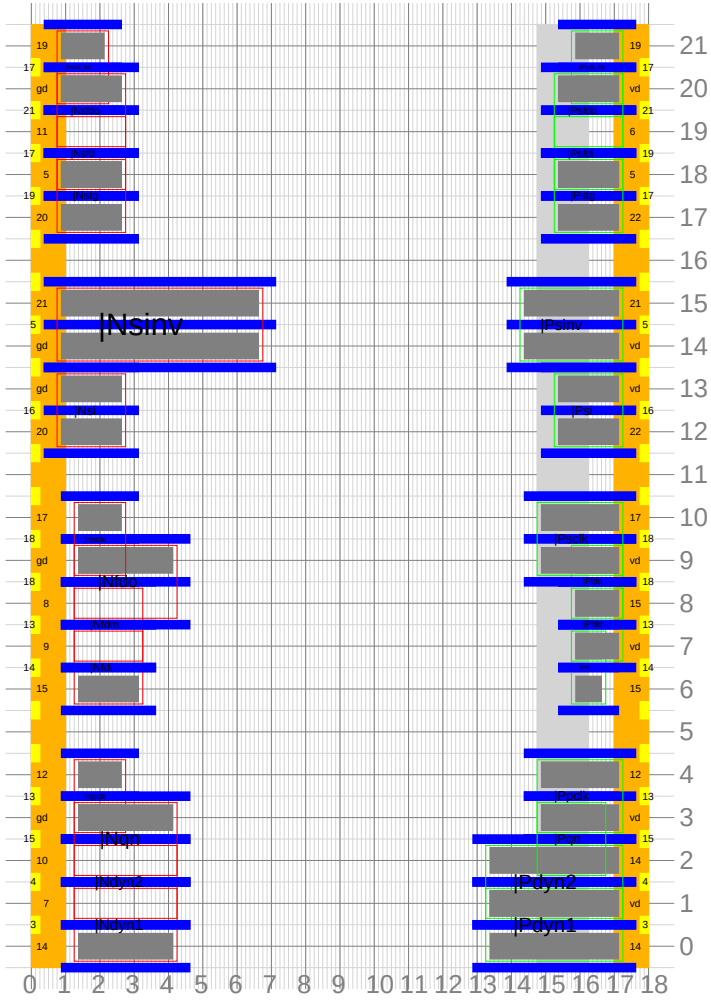


9 FETs, 15 tracks

Estimated runtime: 3 years

BonnCell runtime: 0.8 sec.

(7,782,302 branch & bound nodes)



28 FETs, 22 tracks

Estimated runtime: 10^{30} years

BonnCell runtime: 38 seconds

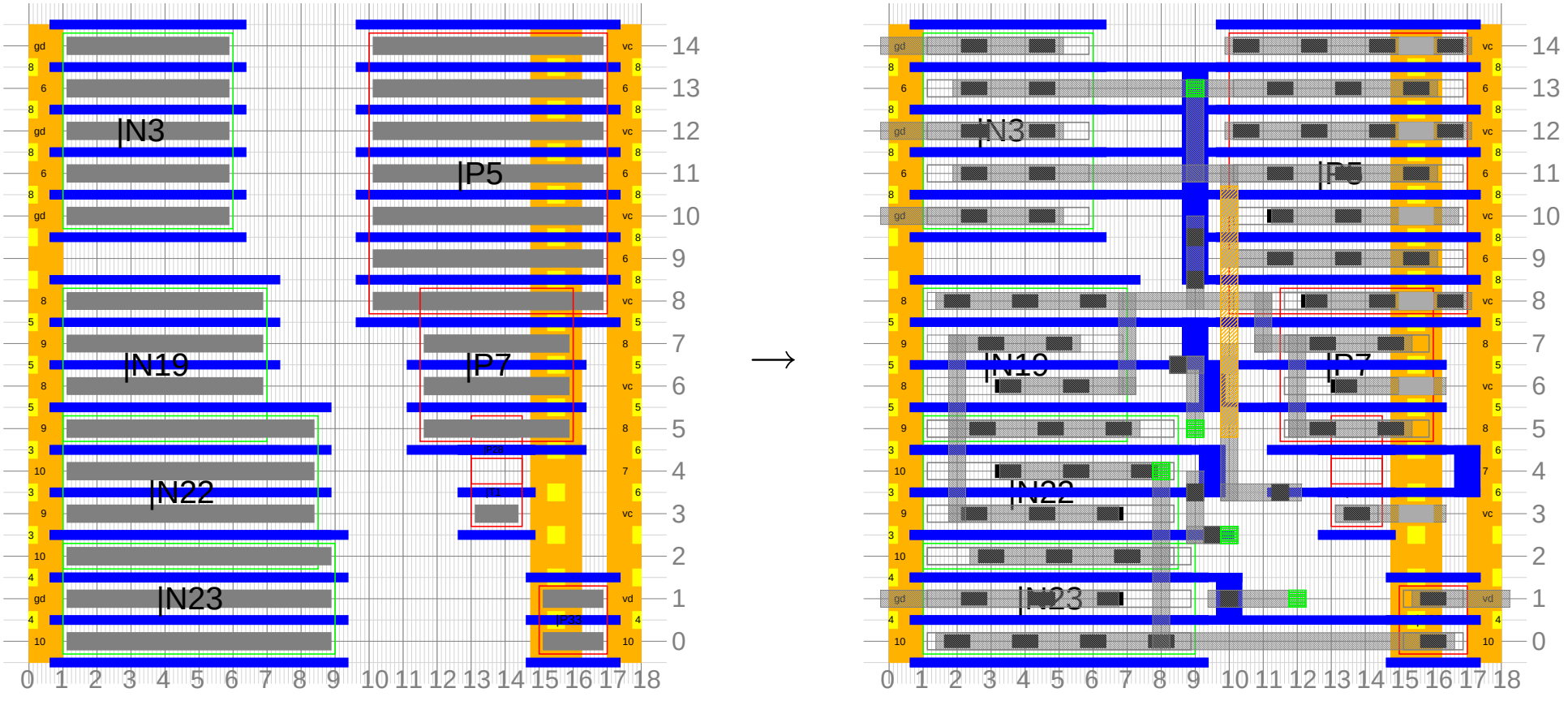
(83,198,650 branch & bound nodes)

Leaf Cell Routing in BonnCell

Leaf Cell Routing Problem

Input: Placed leaf cell with net set $\mathcal{N} = \{T_1, \ldots, T_n\}$

Output: Packing of Steiner trees for T_k, $k = 1, \ldots, n$, subject to various constraints (DRC, limited M2 availabilty, ...)



Leaf Cell Routing Problem

Input: Placed leaf cell with net set $\mathcal{N} = \{T_1, \ldots, T_n\}$

Output: Packing of Steiner trees for T_k, $k = 1, \ldots, n$, subject to various constraints (DRC, limited M2 availabilty, ...)

Our approach:

- Construct half-track **routing grid**
- Block edges for gates, RX, ...

  $\Rightarrow$ $G = (V, E)$ with $T_k \subset V$ for all k
- Route by MIP-based **constraint generation approach**
  - Identify each edge usage (per net) with variable

Packing Steiner Trees (s.t. add/l constraints)

Core **MIP formulation** to be solved on $G = (V, E)$:

$$\text{min} \quad \sum_{(i,j)\in E} c_{ij} x_{ij}$$

$$\begin{aligned}
\text{s.t.} \quad x_{ij} - \sum_{k=1}^{n} x_{ij}^k &= 0 && \forall (i,j) \in E \\
x_{ij} &\leq 1 && \forall (i,j) \in E \\
\sum_{(i,j)\in E | i\in W, j\notin W} x_{ij}^k &\geq 1 && \forall W \subset V, W \cap T_k \neq \emptyset \\
& && (V \setminus W) \cap T_k \neq \emptyset, \forall k \\
x_{ij}^k &\in \{0,1\} && \forall (i,j) \in E, \forall k
\end{aligned}$$

- Note: third set of constraints (*Steiner Cut Inequalities*) has **exponential** cardinality
- Gives edge-disjoint Steiner tree packing [Grötschel et al. 97]

Packing Steiner Trees (s.t. add/l constraints)

Add constraints to ensure connectivity of net

- e.g. flow-based LP formulation for single Steiner tree $k \in \{1, \ldots, n\}, r \in T_k$ [Goemans et al. 93]

$$S_{x^k f} := \{(x^k, f) \mid f^t(\delta^+(i)) - f^t(\delta^-(i)) = \text{rhs}_i, i \in V, t \in T_k\}$$

$$\begin{aligned} f_{ij}^t &\leq x_{ij}^k \quad \forall \{i,j\} \in E, \forall t \in T_k \\ f_e^t &\geq 0 \quad \forall e \in A, \forall t \in T_k \end{aligned}$$

$$\text{rhs}_i := \begin{cases} 1 & i = r \\ -1 & i = t \\ 0 & i \in V \setminus \{t, r\} \end{cases}$$

Design Rules

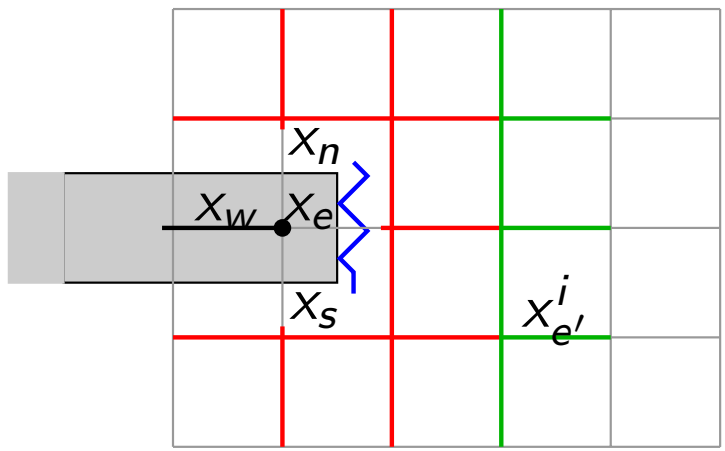
Extend core MIP so that solution satisfies **additional constraints**

- Distance rules between different nets (minspace, interlayer via,...)
- Samenet rules (samenet minspace, minarea...)
- ...

Overall, there are **80+ constraints** induced per (edge, net) pair

Design Rules – Example

Line-End $\Longleftrightarrow$ Polygonal edge between two convex corners closer than $t_{\mathsf{LE}} \in \mathbb{N}$
Line-End requires additional spacing for OPC



In this example, $x_w - (x_n + x_e + x_s) + x^i_{e'} \leq 1$ forces $x^i_{e'}$ to zero.

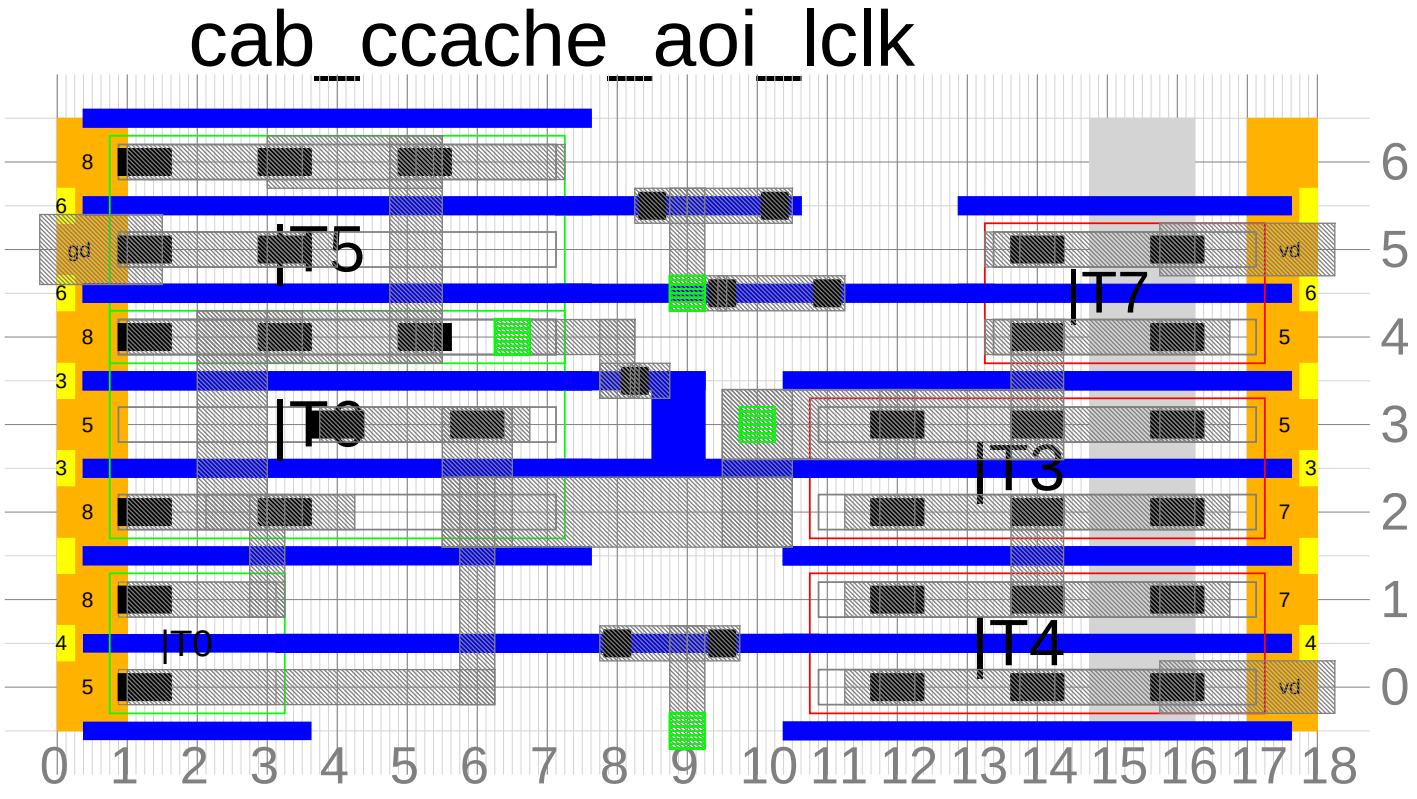
Constraint Generation

Solution Approach

- Due to high number of constraints (of which many are nonbinding), we use **constraint generation**:
  - Start with a limited but useful subset of constraints
  - **Solve MIP**
  - Check *all* constraints for feasibility
    - **All feasible**: Optimal solution found
    - **Not all feasible**: Add (some) violated constraints to MIP and resolve
- Efficient constraint checking:
  - Net connectivity $\Rightarrow$ New cuts or flow formulation
  - DRC violations $\Rightarrow$ Wire-dependent constraints

Add/I Benefit: MIP infeasible $\Rightarrow$ Placement **unroutable**!

Example BonnCell Routing



BonnCell result (basic routing flow)

Running time: 324 seconds

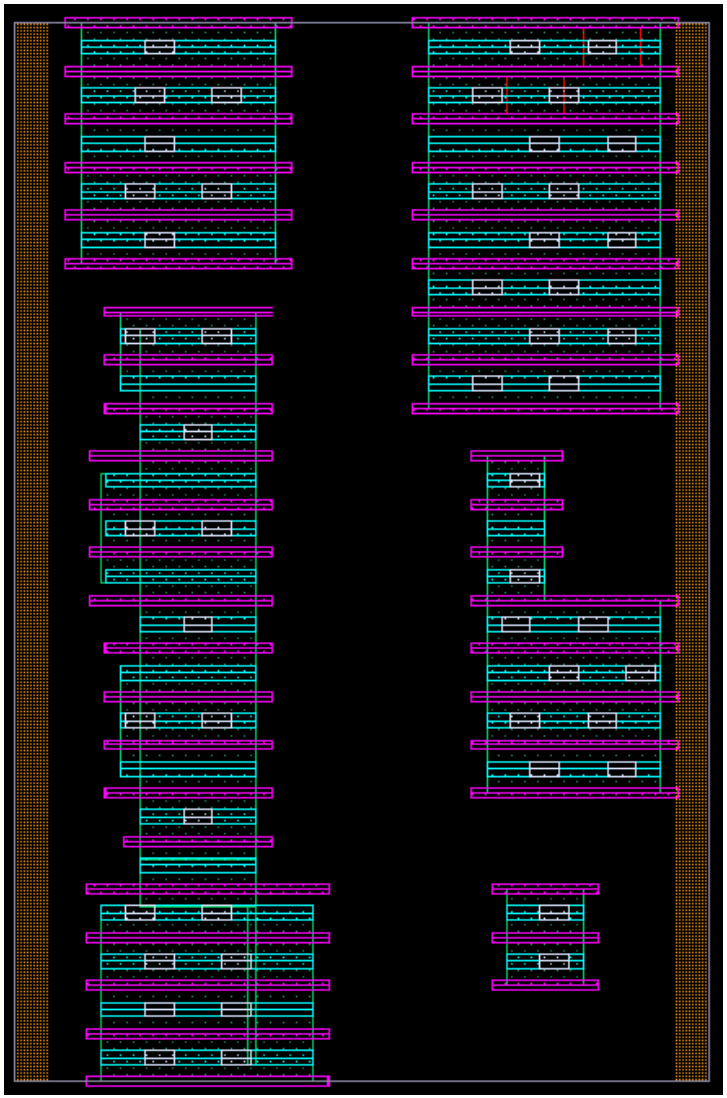
Additional Features

Many features beyond basic functionality

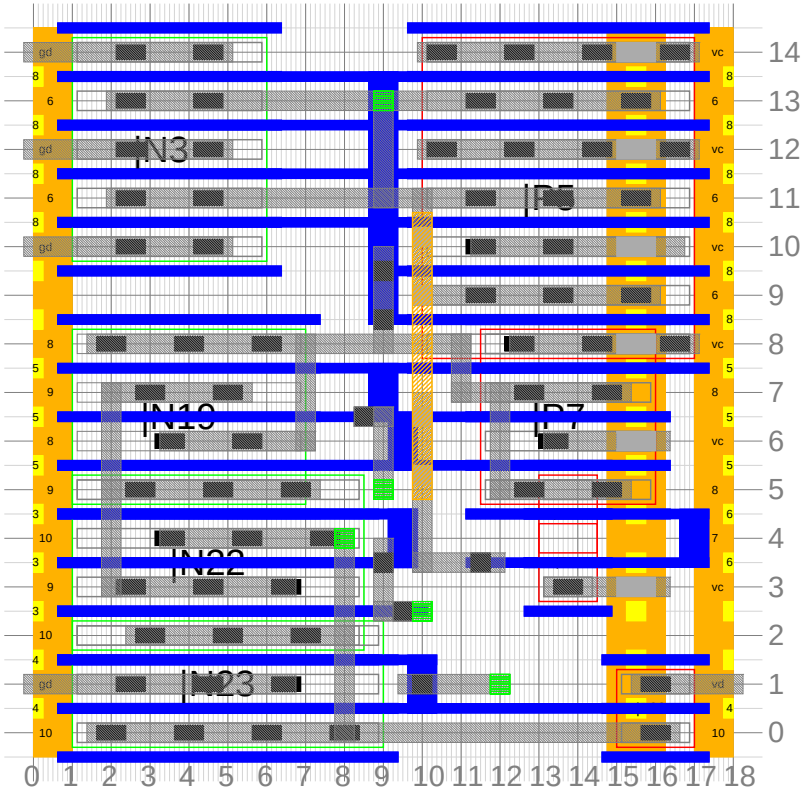
- Spend **additional tracks** to improve routability
- Place within **fixed cell height**
- **Multi-dimensional** placement & routing
- **Folding** multiple FETs
- GUI integration into **Cadence** environment
- Pin placement w/o routing
- **User-defined constraints** like fixations and maximum/minimum widths
- Several modes for **external pins**
- Powerful **Postprocessing** heuristics

Results & Outlook

Designer vs. BonnCell



Designer placement: 22 tracks



BonnCell
placement: 15 tracks
Routing runtime: 14:33

Results on 22 nm testbed

Cell	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{N}\rvert$	$\lvert T\rvert$	Placement h	Placement time	Routing time
1	9	12	57	15	0:04	2:55
2	7	10	32	10	0:01	0:12
3	6	9	41	10	0:01	0:13
4	12	13	57	14	0:43	27:45
5	6	10	25	7	0:01	0:08
6	14	16	39	10	0:01	0:33
7	4	9	47	11	0:01	3:26
8	5	9	34	13	0:01	0:23
9	15	23	47	14	0:01	1:49
10	8	12	46	11	0:01	38:27
11	2	6	32	8	0:00	2:10
12	2	6	56	14	0:01	5:08
13	16	16	51	14	0:01	14:33
14	5	8	16	4	0:00	2:24

$\lvert\mathcal{N}\rvert$ number of nets, $\lvert T\rvert$ number of terminals (active gates, contacts, and external pins), h height in tracks, time is [mm:ss]

Additional BonnCell Applications

- Cell Tuning
  - **resizing** transistors
  - **timing** optimization
  - optimize **pin positions**
  - create **different versions** of same cell
- Postoptimization of Cell Libraries
- Library Design
  - Evaluation of early Design Decisions:
  - how many **tracks** should be used?
  - how much **M2** will be needed?

BonnCell 22nm Results on Library

cell type	# in library	% area reduction	# used*	% area reduction
aoi21	156	7.69	56	0.79
aoi22	80	5.99	55	6.15
buff	96	2.89	0	—
invert	92	0.00	66	0.00
latch	48	2.12	17	1.72
nand2	176	12.88	58	5.28
nand3	132	15.26	50	8.97
nand4	56	7.49	39	7.73
nor2	160	10.17	52	1.77
nor3	116	15.65	44	10.12
oai21	148	6.24	53	0.62
oai22	80	6.32	54	6.98
xnor2	80	12.36	54	12.03
xor2	80	11.20	57	10.78
other	9	5.14	2	4.13
all	1509	8.91	657	6.05

* = used on a testbed of current chips.

Future Work

14 nm version

- 14 nm placement (done)
- 14 nm routing (WIP)
  - Mix of MIP-based and combinatorial approach
- Many technology changes and new design rule types

Thank you!