Physical Library Analysis for Optimal 28-nm and 20-nm Routing (Version 5.0)

Router CAE

08/06/2014

Agenda

Forthcoming challenges at 20-nm process node and below

Pin access checking utility (updated in version 5.0)

Win-win for customers and Synopsys with pin access analysis

Success stories for optimal standard cell pin access

Appendix

- Case study and recommendations

Agenda

Forthcoming challenges at 20-nm process node and below

Pin access checking utility (updated in version 5.0)

Win-win for customers and Synopsys with pin access analysis

Success stories for optimal standard cell pin access

Appendix

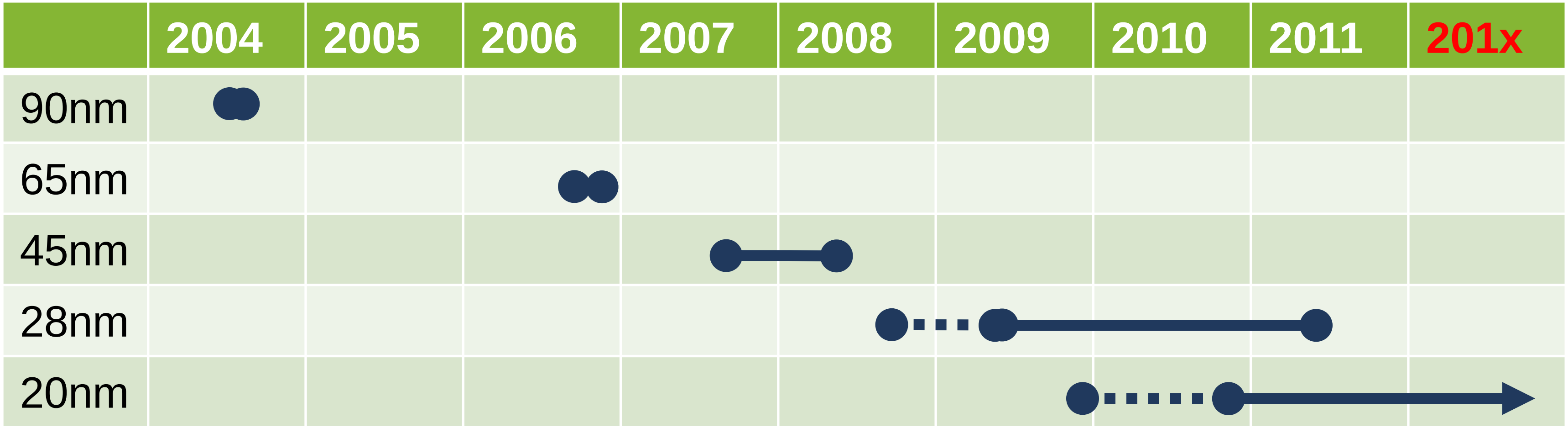
- Case study and recommendations

Forthcoming Challenges at 20-nm and Below

- Long standard cell and foundry process technology development cycle
- Complicated design rules
- Emerging design concern: double patterning

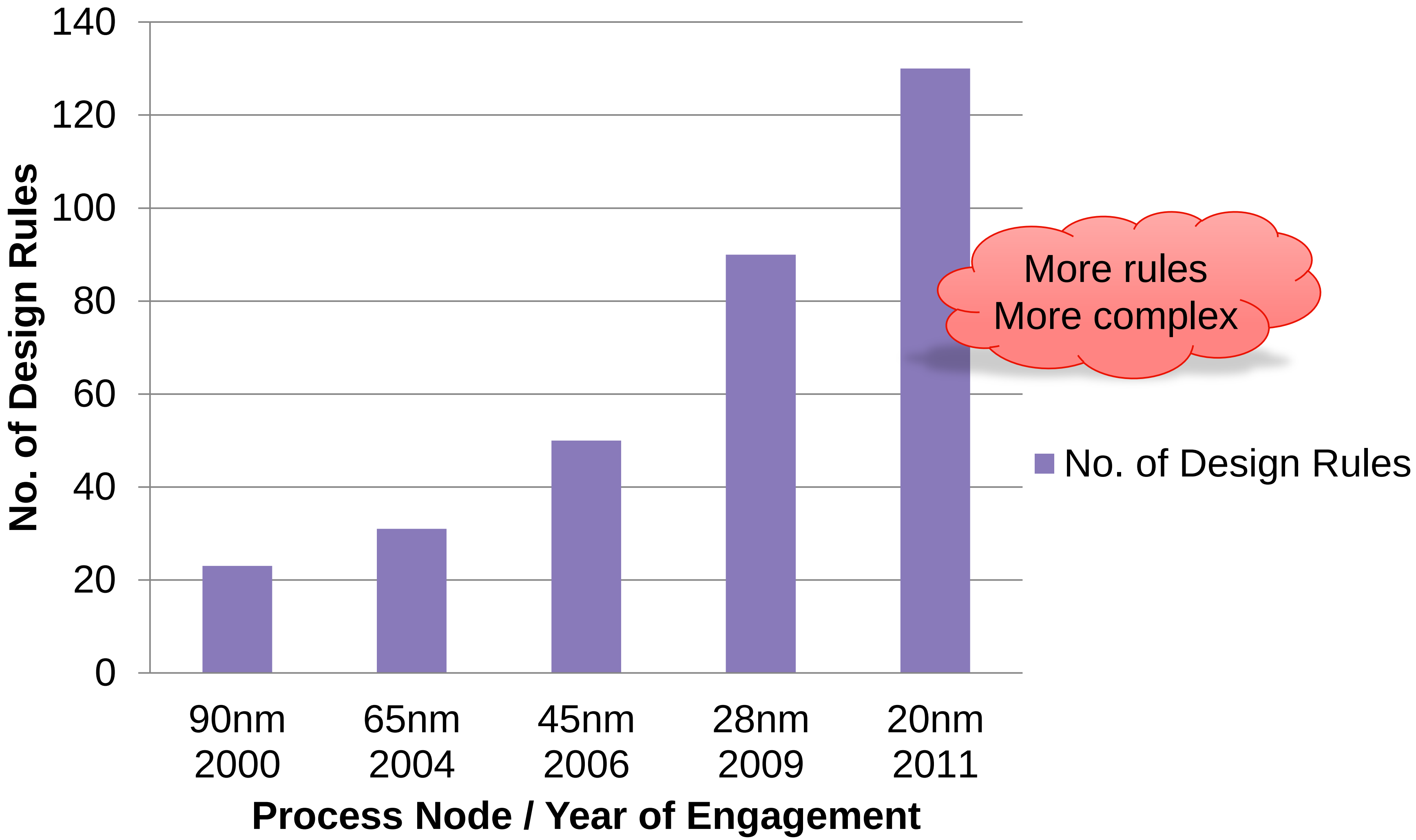
Design Rule Development Cycle

Foundry-SNPS Design Rule Developments



- Longer development cycle at advanced process nodes
- Foundry-Synopsys collaboration is required at a very early stage
- More engineering resources are allocated from both sides

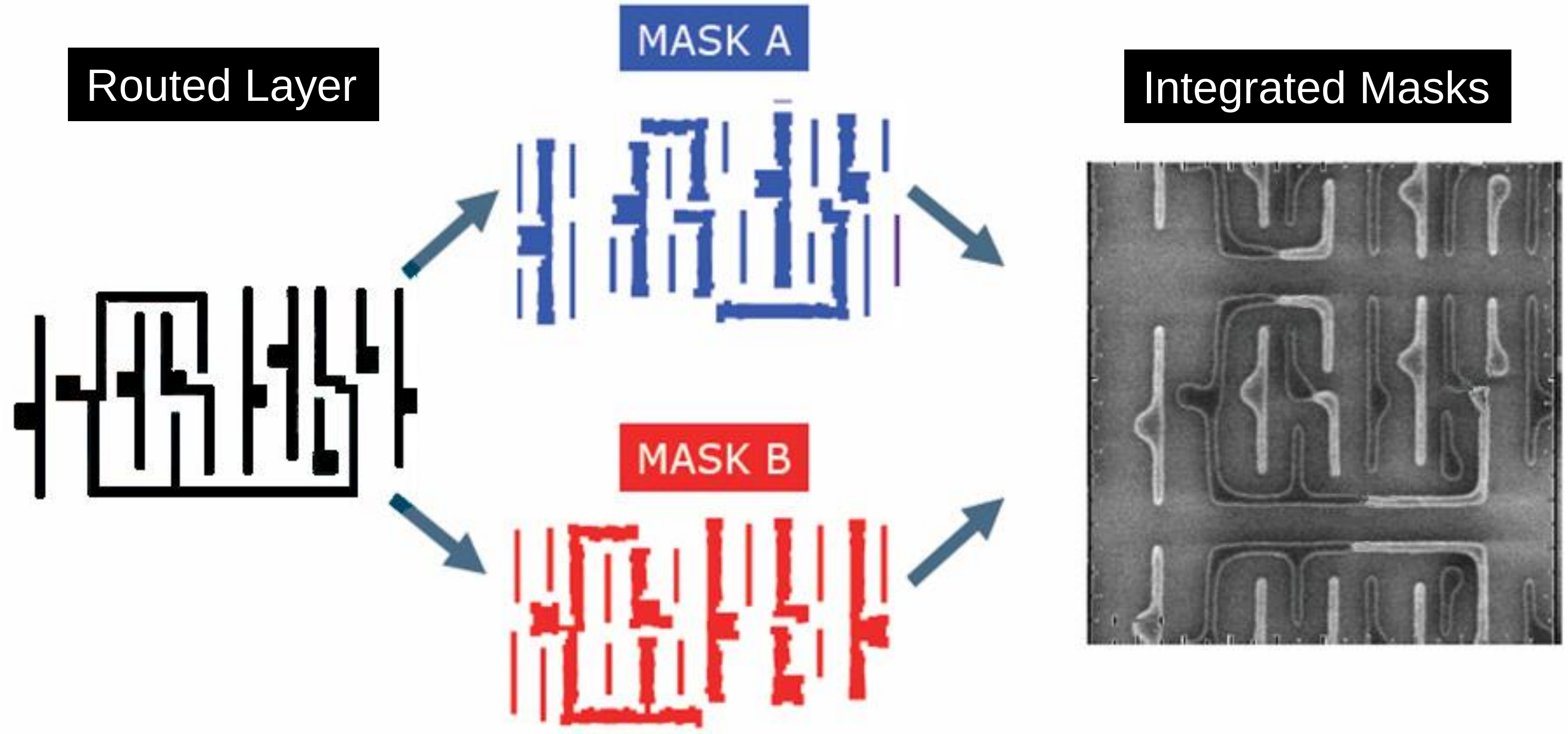
Complicated Design Rules



More rules
More complex

Emerging Design Concern: Double Patterning

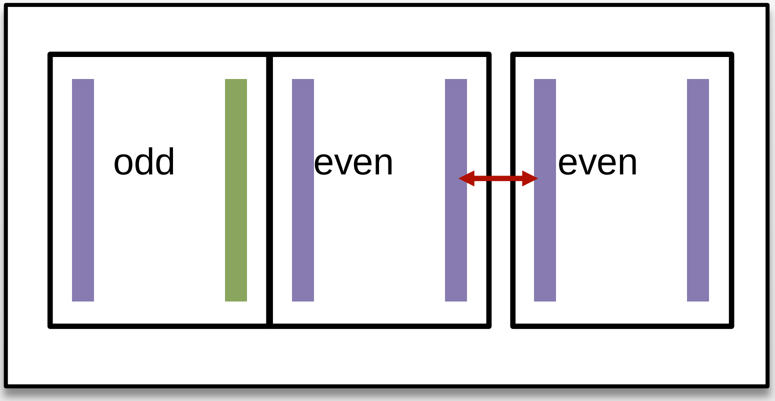
Requires Mask Decomposition



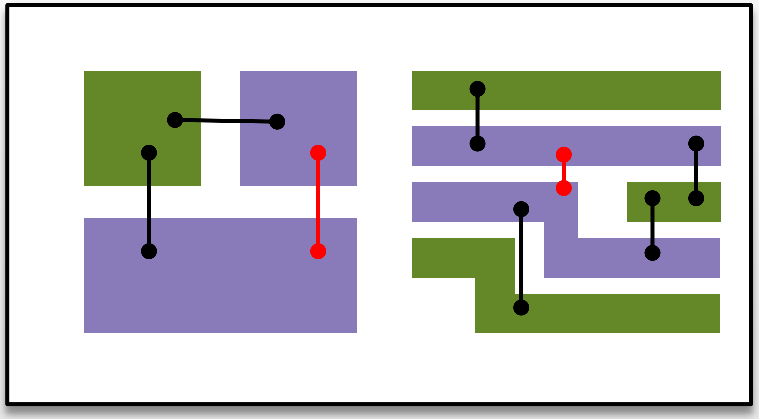
*Source: 2012 SNUG IC Compiler 20nm tutorial

Emerging Design Concern: Double Patterning

Occurrence of Double Patterning Violations



Avoid Double Patterning Violations Due to Cell Placement

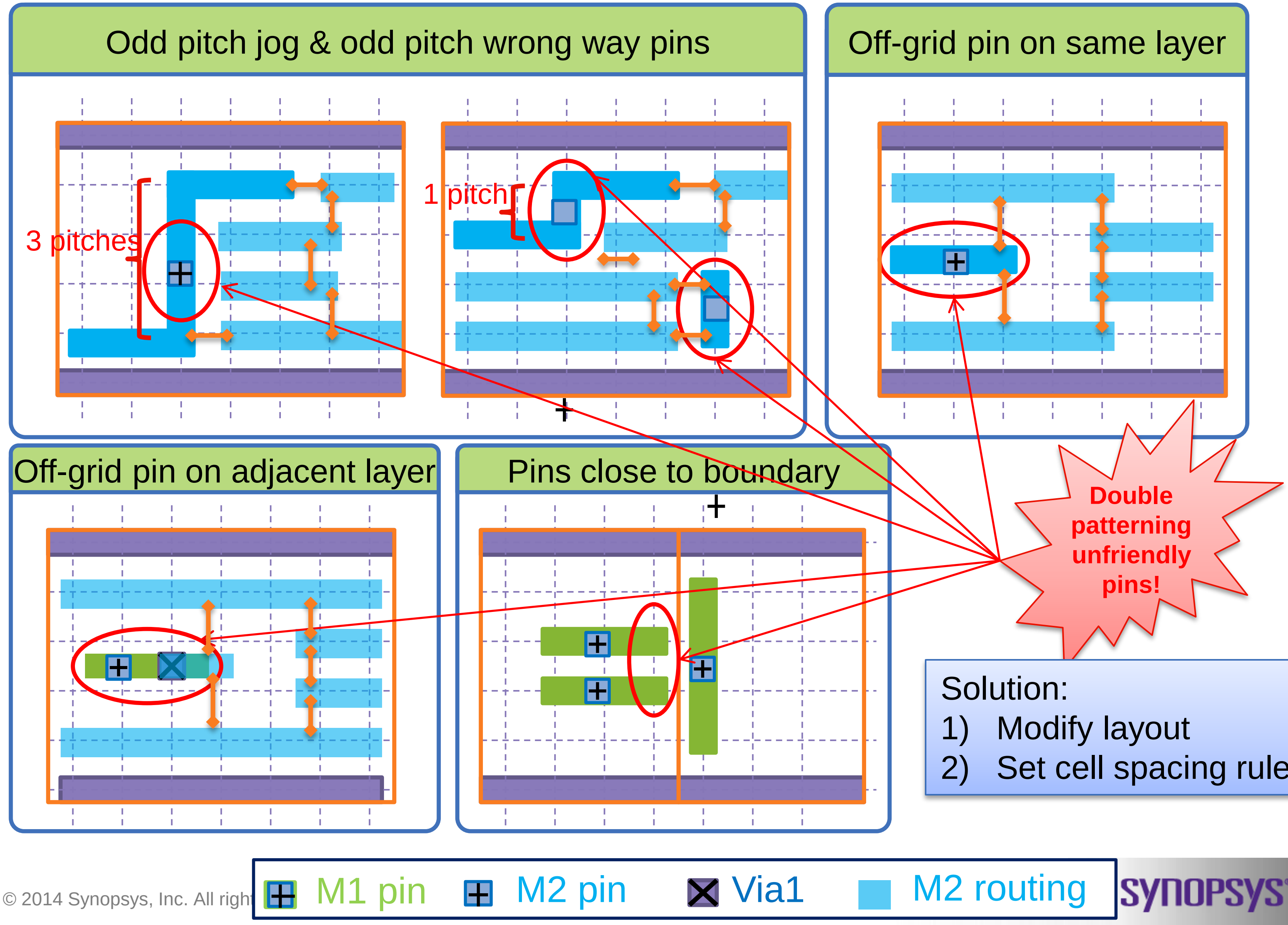


Routing to Avoid Double Patterning Violations

*Source: 2012 SNUG IC Compiler 20nm tutorial

Emerging Design Concern: Double Patterning

Could Not Have Seen If Cells Are Not Placed and Routed …



Odd pitch jog & odd pitch wrong way pins

Off-grid pin on same layer

Off-grid pin on adjacent layer

Pins close to boundary

Double patterning unfriendly pins!

Solution:

1) Modify layout
2) Set cell spacing rules

Agenda

Forthcoming challenges at 20-nm process node and below

Pin access checking utility (updated in version 5.0)

Win-win for customers and Synopsys with pin access analysis

Success stories for optimal standard cell pin access

Appendix

- Case study and recommendations

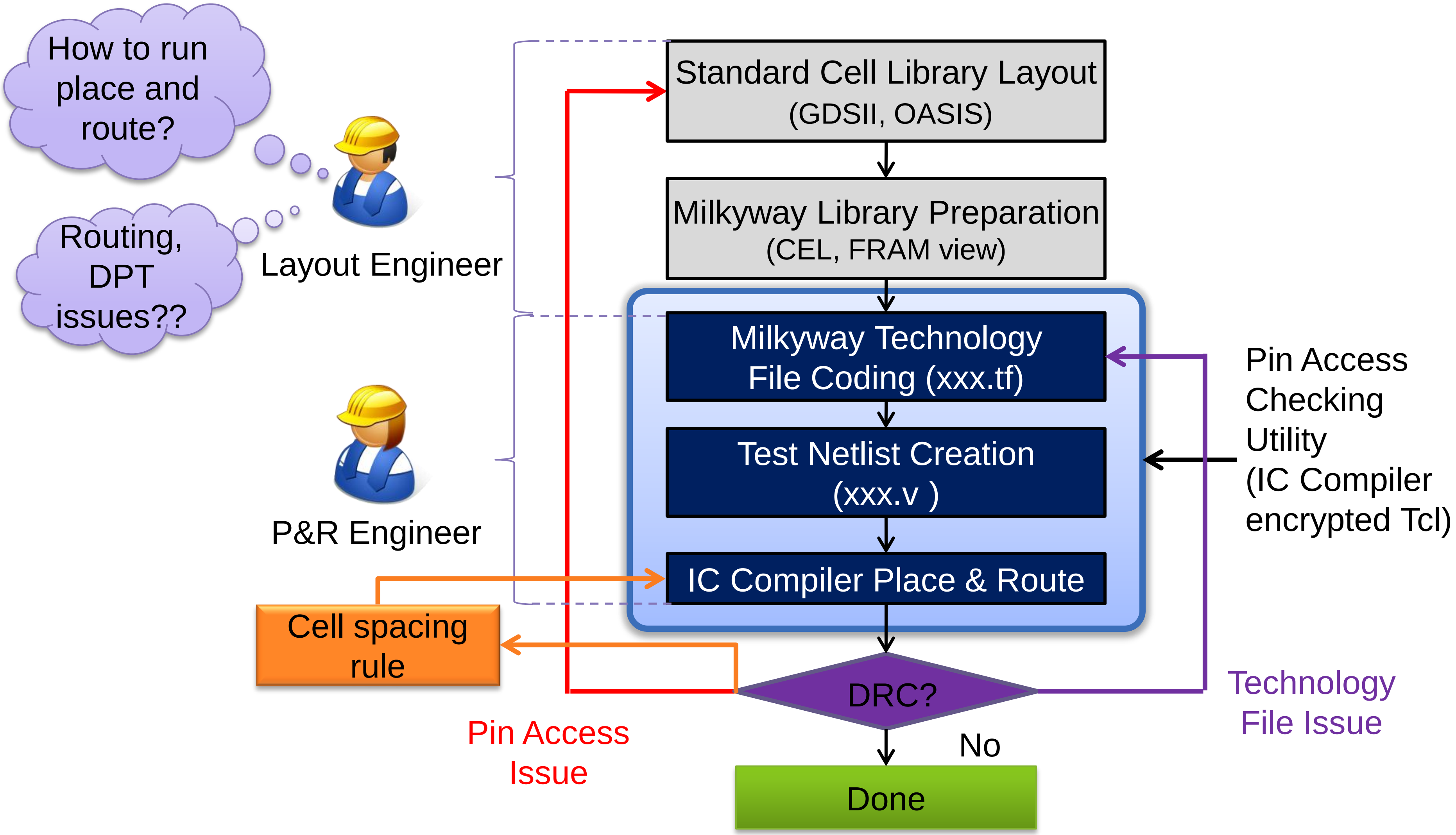
Pin Access Checking Utility

Introduction

- Problem
  - Beginning to see more pin accessibility issues as the solution spaces on a pin become considerably less as a result of more design rules added for advanced process nodes
- Initiation
  - One customer requested that Synopsys develop a script to check routing DRCs between abutted cells
- Motivation
  - Have the capability to see intercell issues
  - To support advanced rules like 20-nm and below
- Solution
  - Develop a script to perform routing on the cells as a way to check pin accessibility

Pin Access Checking Utility

Physical Library Design and Analysis Flow

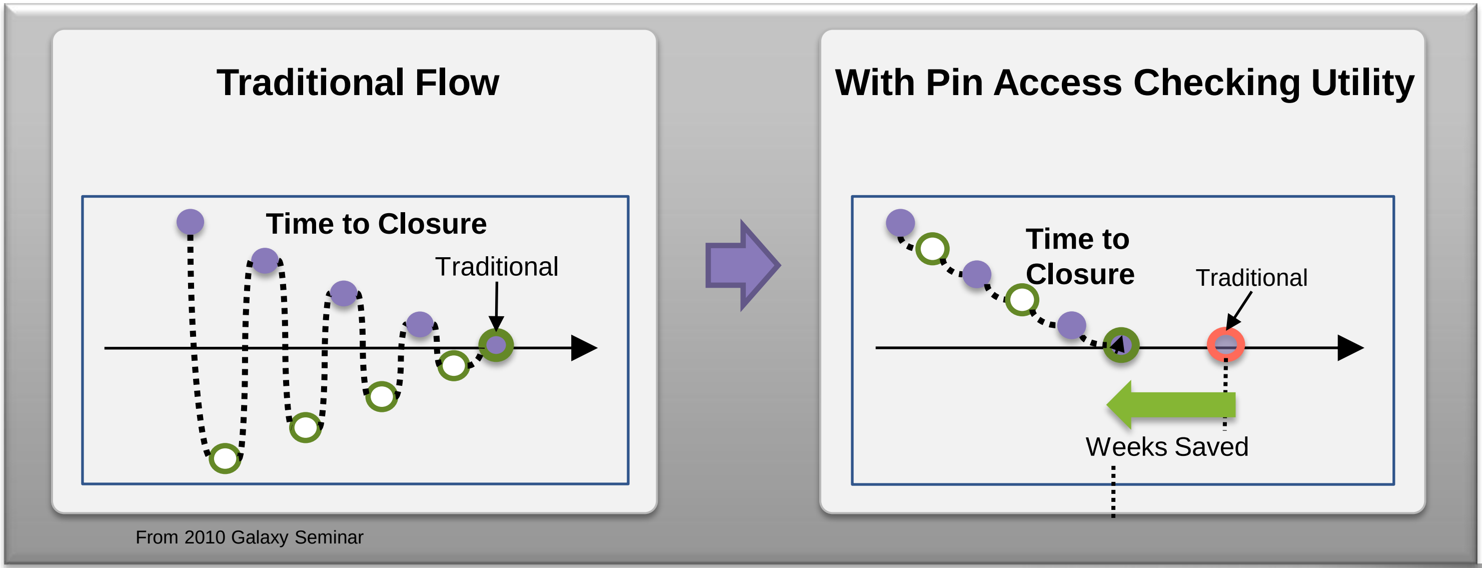


How to run place and route?
Routing, DPT issues??
Layout Engineer
P&R Engineer
Standard Cell Library Layout (GDSII, OASIS)
Milkyway Library Preparation (CEL, FRAM view)
Milkyway Technology File Coding (xxx.tf)
Test Netlist Creation (xxx.v)
IC Compiler Place & Route
Pin Access Checking Utility (IC Compiler encrypted Tcl)
Cell spacing rule
DRC?
Pin Access Issue
Technology File Issue
No
Done

Pin Access Checking Utility

Goals

- Provide an easy-to-use tool to help layout engineers check and fix potential pin accessibility issues during the library development stage
- Avoid seeing DRC surprises at the place and route stage due to bad standard cell library design
- Accumulate pin design experiences

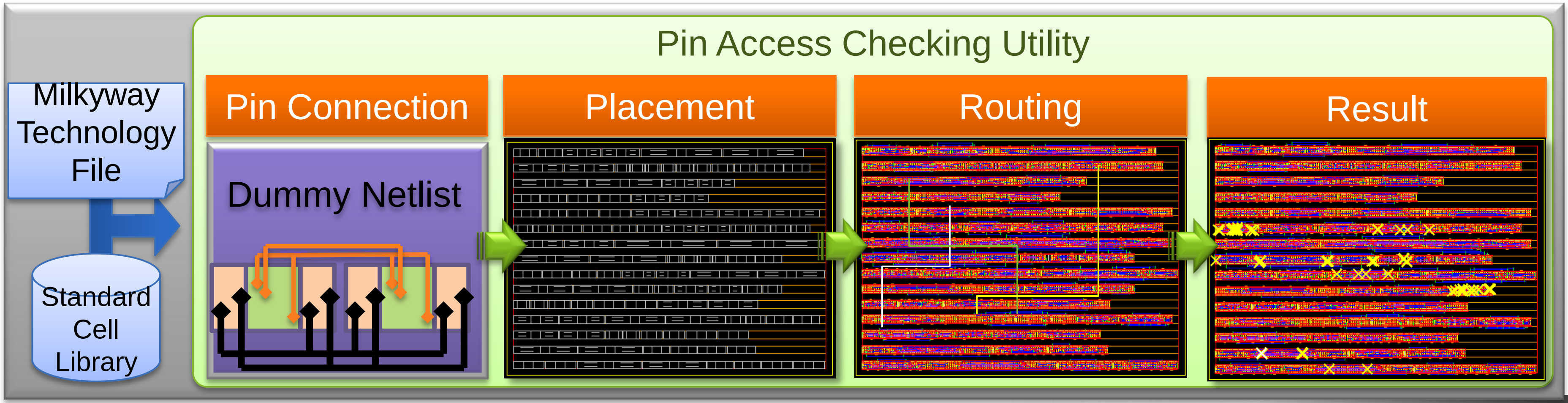


From 2010 Galaxy Seminar

Pin Access Checking Utility

Methodology

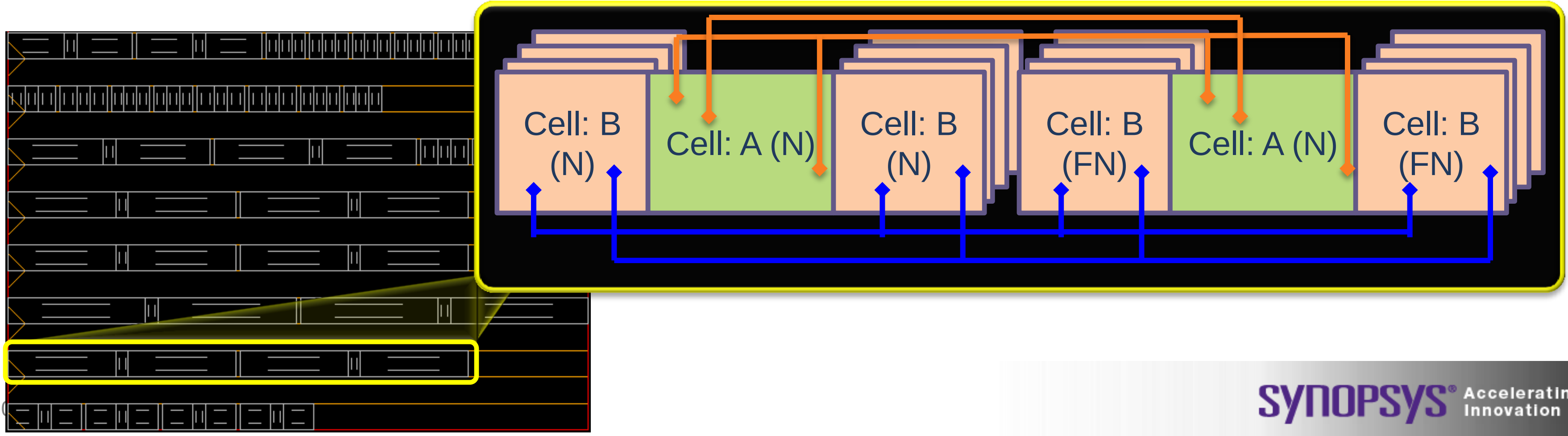
- Netlist
  - Create a dummy netlist that includes all the library cells and connections between pins
- Placement
  - Place all the library cells in the top cell with all possible cell combinations and orientations
- Routing
  - Run Zroute to route all the pins
  - Analyze pin accessibility issues with the DRC result
- Run `check_zrt_routability` (optional)



Pin Access Checking Utility

How Is the Test Cell Created and Checked?

- A test cell is created with one target cell forming all the possible placement combinations with all the library cells (including the target cell)
- The test cell is named after its target cell
  - *Example: Name of the target cell: A => Name of test cell: cell_A*
- Each cell placement combination consists of
  - First set: Place the target cell (A) in the middle with two checked cells (B) abutted on its right and left sides
  - The second set is similar to the first set, but the checked cells (B) are flipped in the horizontal direction
- Connectivity within each pair-placement combination:
  - Pins of the first cell (A) are connected together with corresponding pins of the second cell (A)
  - Same for cell (B)

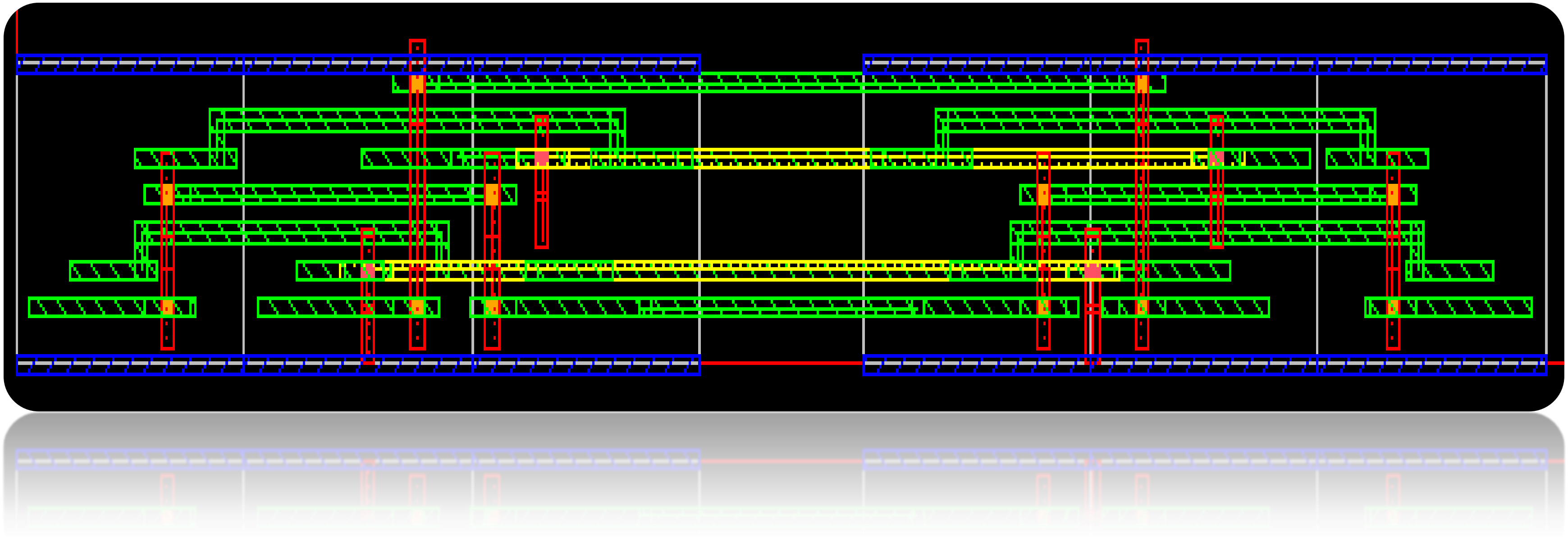


Accelerating Innovation

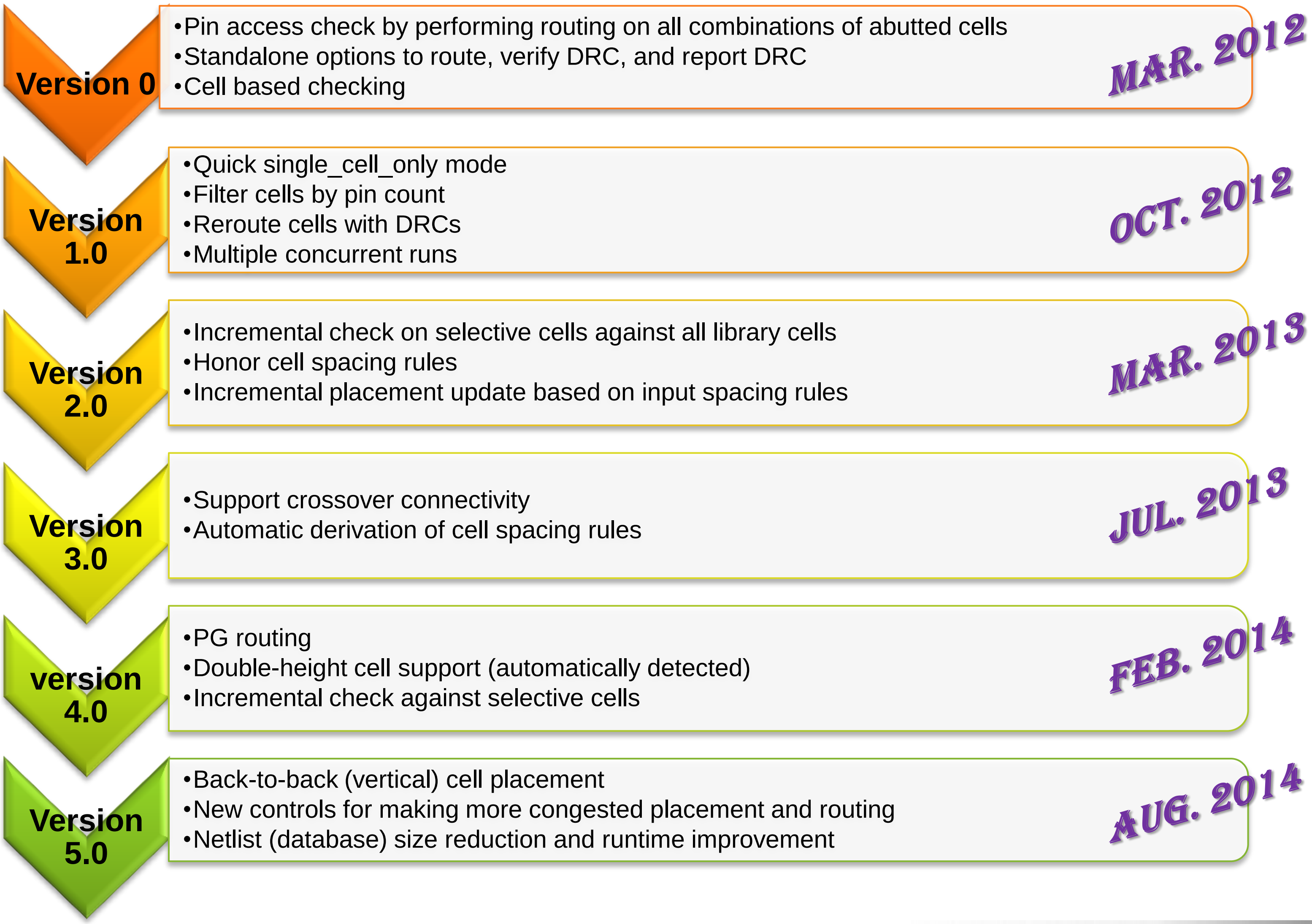
Pin Access Checking Utility

How Is the Test Cell Created and Checked? (Continued)

- `check_zrt_routability` and `route_zrt_auto` are performed on each test cell
- Pin accessibility is checked as the result of `check_zrt_routability` and routing DRC



New Features in Each Version



Version 0

- •Pin access check by performing routing on all combinations of abutted cells
- •Standalone options to route, verify DRC, and report DRC
- •Cell based checking

MAR. 2012

Version 1.0

- •Quick single_cell_only mode
- •Filter cells by pin count
- •Reroute cells with DRCs
- •Multiple concurrent runs

OCT. 2012

Version 2.0

- •Incremental check on selective cells against all library cells
- •Honor cell spacing rules
- •Incremental placement update based on input spacing rules

MAR. 2013

Version 3.0

- •Support crossover connectivity
- •Automatic derivation of cell spacing rules

JUL. 2013

version 4.0

- •PG routing
- •Double-height cell support (automatically detected)
- •Incremental check against selective cells

FEB. 2014

Version 5.0

- •Back-to-back (vertical) cell placement
- •New controls for making more congested placement and routing
- •Netlist (database) size reduction and runtime improvement

AUG. 2014

Pin Access Checking Utility

Syntax Overview

Usage: check_std_cell_pin_access # Check pin accessibility between two abutted standard cells

[-technology *tech_file*] (Technology file name; Required input)

[-std_lib_path *directory_path*] (Path of standard cell library; Required input)

[-lib *lib_name*] (Name of the library that will be created for pin access checking; Default: check_lib.mw)

[-cells *cell_list*] (Name of the standard cells that will be created for pin access checking; Default: all cells in the library)

[-exclude_cells *cell_list*] (Name of the standard cells that will be excluded from pin access checking; Default: None)

[-check_against_all_lib_cells] (Check selected cell(s) against all cells in the input standard cell library or libraries; Default: off)

[-check_against_lib_cells {cell_list}] (Check against selected cells in the input standard cell library or libraries)

[-filter_cells_by_pin_count *integer*] (Skip creating test cell for standard cells with pin count less than the specified threshold; Default: 1 (skip cells with no signal pins); To see only the pin count statistics without creating test cells, enter -1)

[-mode (single_cell_only | cell_by_cell_only | all_combo_in_one_cell_only | all)] (Default: cell_by_cell_only)

single_cell_only (check all cells standalone)

cell_by_cell_only (check one cell at a time against rest of library cells)

all_combo_in_one_cell_only (check one flat cell with all cell combinations)

all (cell_by_cell + all_combo_in_one_cell)

[-congested_netlist] (Create crossover connections and make test cells more congested; Default: off)

[-thread *integer*] (Number of threads to be used for routing; Default: 4)

[-preplace_option_file *file_name*] (Tcl file to be sourced before placement; Optional)

New in ver5.0

[-route_option_file *file_name*] (Tcl file to be sourced during routing; Optional)

[-cell_spacing_rule_file *file_name*] (Tcl file that contains cell spacing rules to be honored for cell placement; Optional)

[-derive_cell_spacing_constraint *file_name*] (Output cell spacing constraint Tcl script to specified file; Optional)

[-update_placement] (Update placement of test cells with respect to spacing rule. Prerequisite: test cells must be created already; Default: off)

[-core_margin] (Specify spacing (imultiple of tile width) between boundary and core. The actual spacing will be N X tile width; Default: 1)

[-group_spacing] (Specify spacing (multiple of tile width) between each cell group. The actual spacing will be N X tile width; Default: 1)

[-use_metal1_routing] (Metal1 will be used for routing. Metal1 pin shapes are likely to get changed; Default: off)

[-run_check_routeability] (Run classic-router-based check_routeability; Default: off)

[-run_check_zrt_routability] (Run Zroute-based check_zrt_routability. Supported only in G-2012.06 or later versions)

[-skip_routing] (Just create the test cells and skip routing; Default: off)

[-reroute_only] (Skip cell creation and just reroute all cells; Default: off)

[-reroute_drc_cells_only] (Rip-up and reroute violated nets in cells with DRC violations; Default: off)

Pin Access Checking Utility

Syntax Overview (Continued)

Usage: check_std_cell_pin_access # Check pin accessibility between two abutted standard cells

[-effort low | medium | high] (Default: medium)

[-max_detail_route_iterations *integer*] (Max number of detail route iteration; Default: 20)

[-num_cells_per_parallel_run *integer*] (Divide run into multiple jobs per number of cells; Default: all cells)

[-parallel_run_icc_shell_path *path/icc_shell*] (Specify path to icc_shell; Default: use icc_shell in the search path)

[-pin_access_check_tcl_path *tcl_path/tcl_name*] (Required if -num_cells_per_parallel_run used)

[-verify_cell_drc_only] (Verify DRC on all cells. Skipping cell creation and routing; Default: off)

[-report_cell_drc_only] (Report DRC on all cells and return cells with DRC violations. Skipping cell creation and routing; Default: off)

[-reroute_drc_cells_iterations *integer*] (Reroute iteration for cells with DRC violations; Default: 0; Max:5)

[-pg_route] (Create horizontal & vertical PG straps; This option is automatically turned on when any of the -pg_xxx option is specified; Default: off)"

[-create_pg_before_placement] (Create PG routes before placement; Default: off (after placement))

New in ver5.0

[-pg_output_tcl] (Output PG routing script to a file when any of -pg_xxx option is specified. Default: create_pg_route_pin_access_check.tcl)"

[-pg_hlayer] (Specify metal layer for horizontal PG straps; Default: topmost horizontal metal layer)

[-pg_vlayer] (Specify metal layer for vertical PG straps; Default: pg_hlayer-1 metal layer)

[-pg_hwidth] (Specify the width of horizontal PG straps; Default: 1um)

[-pg_vwidth] (Specify the width of vertical PG straps; Default: 1um)

[-pg_hpitch] (Specify the pitch of horizontal PG straps; Default: 10um)

[-pg_vpitch] (Specify the pitch of vertical PG straps; Default: 10um)

[-m2_rail] (Create horizontal M2 PG rails on top of M1 rails; Default: off)

[-m2_rail_width] (Specify the width of M2 PG rails; Default: M2 default width)

[-remove_duplicate_cell_pairs] (Remove repeating cell pairs in each test block "cell_xxx"; Default: off)

[-init_utilization *ratio*] (Specify initial design utilization; Default: 0.25)

[-back2back_placement] (Place cells back to back or in vertical direction. Only support single height cell. Default: off)

[-back2back_target_cell_offset *integer*] (Specify the offset distance for the target cell with respect to its neighbor cells. The offset distance will be N X tile width; Default: 0)

[-fill_empty_rows_with_blockages {*metal_list*}] (Specify metal blockage layers to be filled in the empty rows. Default: off)

[-no_empty_row] (Place cells in consecutive rows. Only support single height cell. Default: leave one empty for every placed row)

New in ver5.0

Pin Access Checking Utility

Supports Four Checking Modes

- `single_cell_only`
  - Fast pin accessibility check without considering other cells
  - You should run this mode first before using other modes

- `cell_by_cell_only` (default)
  - Checks one cell against the all the library cells
  - The test cell is named by the target cell name with prefix of “cell_”
  - Easier to debug by isolating one cell at a time

- `all_combo_in_one_cell_only`
  - Checks all cells with all the possible cell combinations at once
  - By default, the cell is named “pin_access”

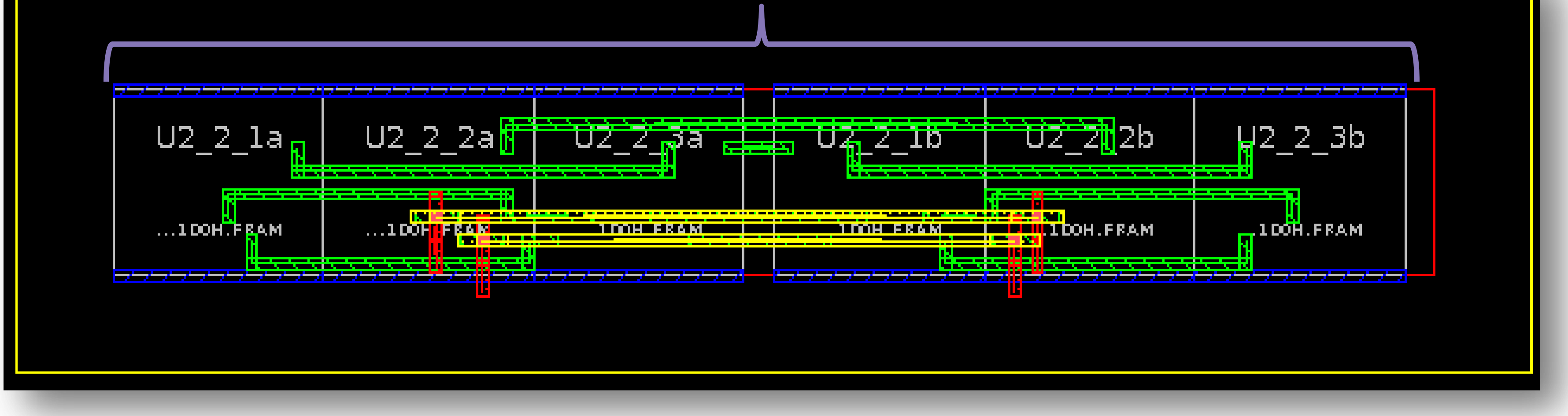
- `all`
  - `cell_by_cell` plus `all_combo_in_one_cell`

Pin Access Checking Utility

Supports Four Checking Modes (Continued)

- `-mode single_cell_only`
  - Performs fast checking on all cells standalone
  - Ensures there is no pin accessibility issue within a cell
  - You should run this mode first before running other modes
  - Always run with a single thread

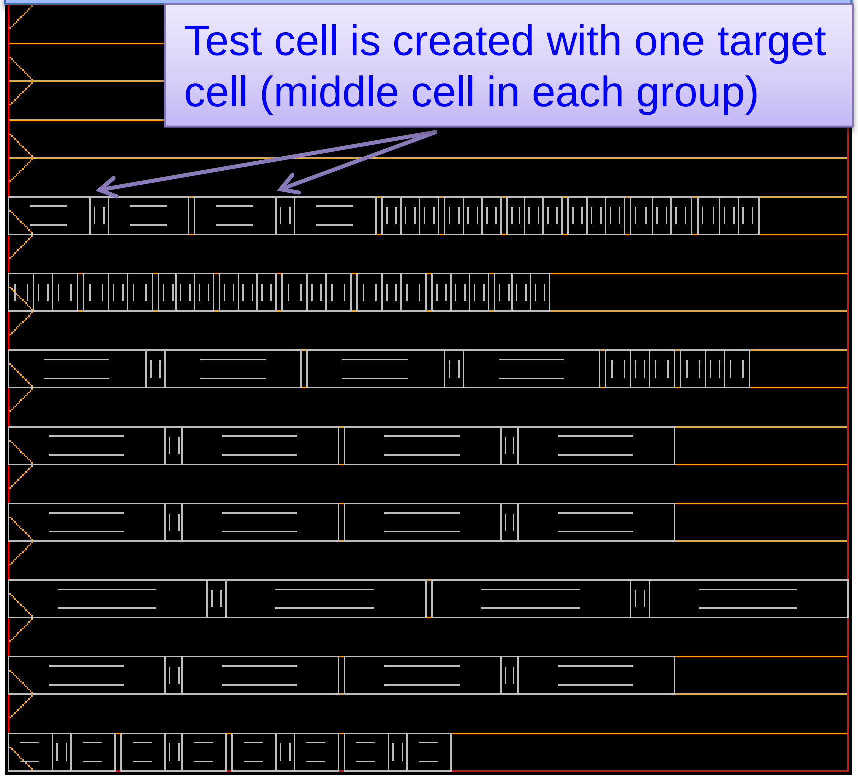
Test cell formed with six identical cells



Pin Access Checking Utility

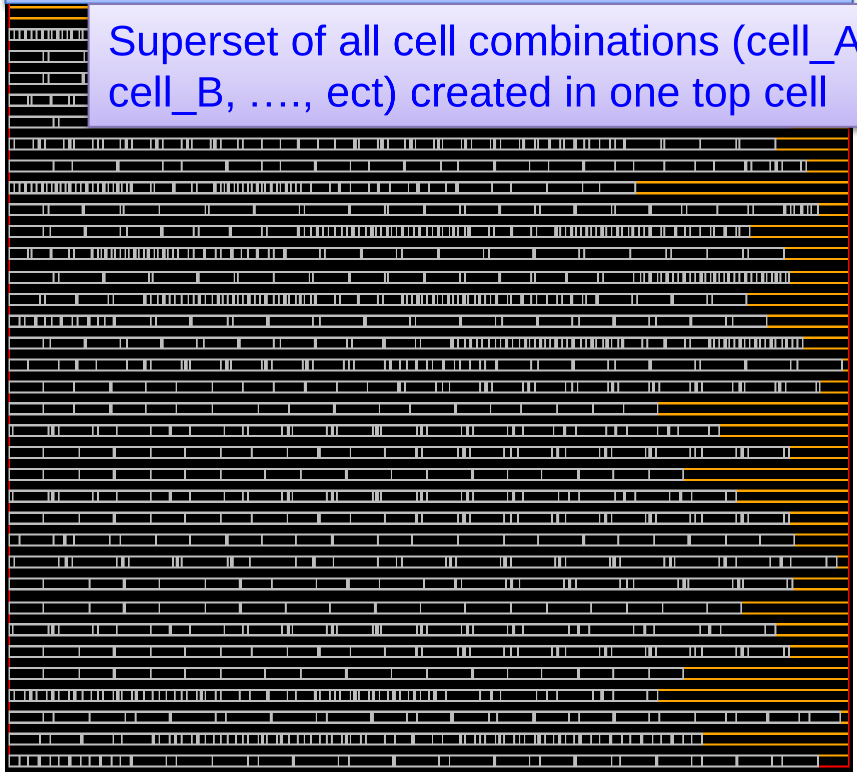
Supports Four Checking Modes (Continued)

```
-mode cell_by_cell_only
cell_A.CEL
```



Test cell is created with one target cell (middle cell in each group)

```
-mode all_combo_in_one_cell_only
pin_cell.CEL
```



Superset of all cell combinations (cell_A, cell_B, ..., ect) created in one top cell

*Note: Cells are placed on every other rows

Pin Access Checking Utility

Other Options

- `-cells {`*`cell_list`*`}` and `-exclude_cells {`*`cell_list`*`}`
  - Includes or excludes library cells for checking pin accessibility
  - Helpful in reducing the runtime by only checking the cells of interest
  - Exclude cells that have the same physical layout on routing layers
- `-filter_cells_by_pin_count` *`integer`* (default: 1)
  - Skips test cell creation for cells with pin count less than the specified number
  - 1 (default): Skip cells without any signal pins
  - -1: Show pin count statistics without creating test cells
- `-run_check_zrt_routability`
  - Runs the Zroute-based `check_zrt_routability` command before routing
  - Supported only in version G-2012.06 and later versions
- `-run_check_routeability`
  - Runs the classic-router-based `check_routeability` command before routing

Pin Access Checking Utility

Other Options (Continued)

- `-skip_routing`
  - Creates test cells without running routing
  - Route options are saved in the cell
  - Can use with `-run_check_zrt_routability`
  - Cannot use with `-reroute_only`

- `-reroute_only`
  - Prerequisite: Test cells must be created
  - Reroutes all or specific cells when no netlist or placement change is made to the already created test cells
  - Can use with `-cells` and `-exclude_cells`

- `-reroute_drc_cells_only`
  - Rips up and reroutes violated nets in cells with DRC violations

- `-effort low | medium | high` (default: `medium`)
  - `low`: Runs a maximum of 10 detail route iterations
  - `medium` (default): Runs a maximum of 50 detail route iterations
  - `high`: Runs medium effort plus an additional incremental `route_zrt_detail` if there are DRC violations left in the medium-effort run

Pin Access Checking Utility

Other Options (Continued)

- `-use_metal1_routing`
  - Enables metal1 routing
  - Metal1 routing is not recommended for 20nm and below. Changing metal1 pin shape is likely to create double patterning violations and other DRC violations

- `-route_option_file` *`file_name`*
  - Specifies route options to be honored during routing
  - The route option file is sourced before routing. If any route settings in the route option file conflict with the utility's embedded options, such as `-use_metal1_routing`, the route option file takes precedence
  - To get consistent results, always add non-persistent settings to the route option file
  - You can include PG rails in the pin access checking by using this option
    - Use the `derive_pg_connection` and `preroute_standard_cells` commands in the route option file to create the PG rails

Pin Access Checking Utility

Other Options (Continued)

- `-num_cells_per_parallel_run` (default: all cells)
  - Divides routing into multiple jobs per number of cells
- `-parallel_run_icc_shell_path` (default: icc_shell in search path)
  - Specifies the icc_shell binary to be used for parallel runs
- `-pin_access_check_tcl_path` *`path/pin_check_script`*
  - Specifies the path of the pin access checking utility
  - Required if you use the `-num_cells_per_parallel_run` option
  - Example: *path*/check_std_cell_pin_access.tcl

Pin Access Checking Utility

Other Options (Continued)

- `-check_against_all_lib_cells` (default: off)
  - Checks the specified cells against all cells in the input standard cell library
  - Useful for incrementally checking newly added cells
- `-cell_spacing_rule_file` *`file_name`*
  - Uses the cell spacing rules defined in the input Tcl file to place the cells
- `-group_spacing` (default: 1)
  - Specifies the spacing (multiple of tile width) between each cell group
- `-update_placement` (default: off)
  - Updates the test cell placement according to the cell spacing rules and group spacing
  - Prerequisite: The test cells must already exist

Pin Access Checking Utility

Other Options (Continued)

- `-verify_cell_drc_only`
  - Verifies DRCs on all cells (skips cell creation and routing)
- `-report_cell_drc_only`
  - Reports DRCs on all cells (skips cell creation and routing)
  - Run this option at the end of parallel runs to get an overall DRC summary

Pin Access Checking Utility

Other Options (Continued)

- `-derive_cell_spacing_constraint` *`file_name`* (default: off)
  - Derives a cell spacing constraint file based on the locations of unfixable DRC violations
  - Accumulates spacing rules from the `–cell_spacing_rule_file` option or the spacing rules saved in the cell
  - Output spacing rule commands:

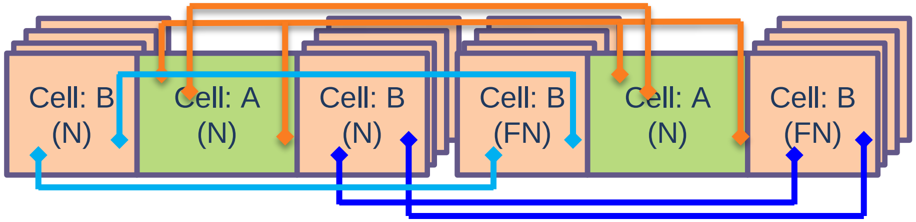
```
set_lib_cell_spacing_label -names -left_lib_cells
set_lib_cell_spacing_label -names -right_lib_cells
set_spacing_label_rule labels
```

  - Limitations:
    - The generated spacing rule keeps at most one site spacing between cells
    - Try to correct pin access issues within the cell layout as much as possible; the placement runtime could be affected by the number of cell spacing rules used
    - The output cell spacing constraint file is generated based only on routing DRC violations; it does not consider other process-related or placement rules

Pin Access Checking Utility

Other Options (Continued)

- `-congested_netlist` (default: off)
  - Creates crossover connections to make the netlist more congested



- `-core_margin` (default: 1)
  - Specifies the spacing (multiple of tile width) between the cell boundary and the core region

- `-max_detail_route_iterations` (default: 20)
  - Specifies the maximum number of detail route iterations

- `-report_cell_drc_only`
  - Enhanced to return cells with DRC violations
  - You can use the returned cells for incremental spacing rule derivation or rerouting

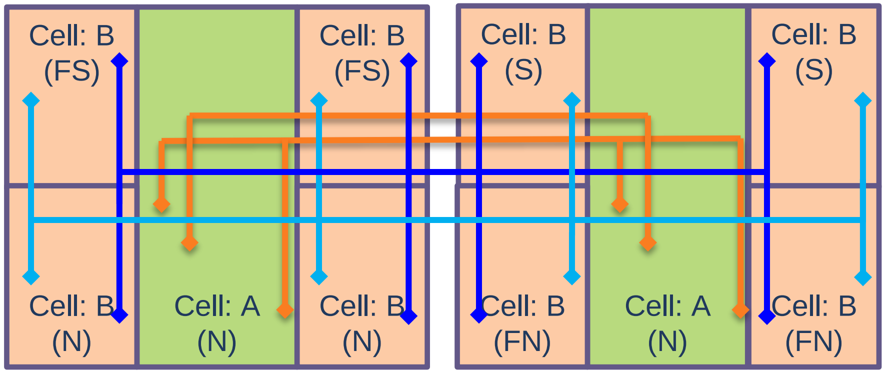
Pin Access Checking Utility

Other Options (Continued)

- `-check_against_lib_cells` *`{cell_list}`* (default: off)
  - Specify cells to be check against. An alternative for -check_against_all_lib_cells

- `-max_detail_route_iterations` (default: 20)
  - Max number of detail route iterations

- Support double height cell (automatically detected)
  - The upper row of double height cell is placed with same cells in the lower row but flipped in the vertical direction



Pin Access Checking Utility

Other Options (Continued)

PG Routing Options

- Design rules at emerging nodes bring new challenges to power planning and design routing closure. Due to complicated advanced design rules, the signal router can be very sensitive to pre-routed shapes that cannot be moved automatically.
- It is recommended to include PG routes in the final phase of pin access checking after all cells have been verified “accessible” without PG routes
- All the PG routing commands by default honors pre-route advanced via rule. If the user wishes to use any advanced via rule for PG routing, please use “set_preroute_advanced_via_rule” to define it in the –route_option_file

Pin Access Checking Utility

Other Options (Continued)

PG Routing Options :

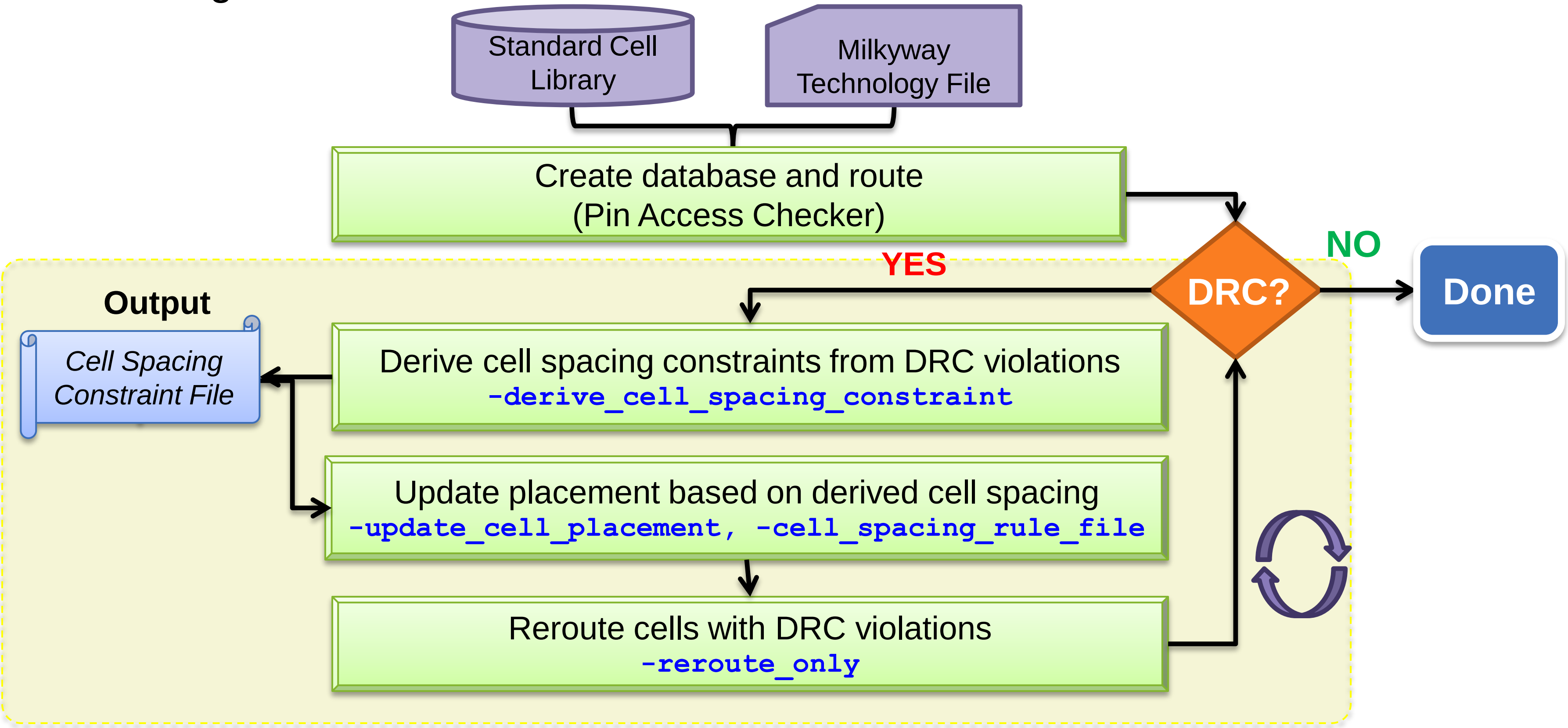
- `-pg_route` (Default: off)
  - Create horizontal and vertical PG straps; This option is automatically turned on when any of the -pg_xxx option is specified
- `-pg_output_tcl` (Default: create_pg_route_pin_access_check.tcl)
  - Output PG routing script to a file when any of -pg_xxx option is specified
- `-pg_hlayer` (Default: topmost horizontal metal layer)
  - Specify metal layer for horizontal PG straps;
- `-pg_vlayer` (Default: pg_hlayer–1 metal layer)
  - Specify metal layer for vertical PG straps
- `-pg_hwidth` (Default: 1um)
  - Specify the width of horizontal PG straps
- `-pg_vwidth` (Default: 1um)
  - Specify the width of vertical PG straps
- `-pg_hpitch` (Default: 10um)
  - Specify the pitch of horizontal PG straps
- `-pg_vpitch` (Default: 10um)
  - Specify the pitch of vertical PG straps
- `-m2_rail` (Default: off)
  - Create horizontal M2 PG rails on top of M1 rails
- `-m2_rail_width` (Default: M2 default width)
  - Specify the width of M2 PG rails

Pin Access Checking Utility

Other Options (Continued)

- `-derive_cell_spacing_constraint`

  Derives the cell spacing constraints iteratively to improve the spacing rule coverage



Standard Cell Library
Milkyway Technology File
Create database and route (Pin Access Checker)
DRC?
NO
Done
YES
Output
Cell Spacing Constraint File
Derive cell spacing constraints from DRC violations
-derive_cell_spacing_constraint
Update placement based on derived cell spacing
-update_cell_placement, -cell_spacing_rule_file
Reroute cells with DRC violations
-reroute_only

Pin Access Checking Utility

New Options in Version 5.0

- `-preplace_option_file` *`file_name`*
  - Specifies a Tcl file to be sourced before placement
  - Use this file to specify floorplan- or placement-related variables and options for IC Compiler to honor. For example, you could include the following command to specify the unit tile name:

    ```
    set_fp_options -unit_tile_name unitA
    ```

New in ver5.0

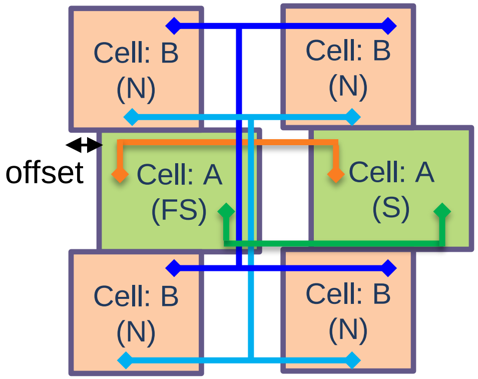
- `-create_pg_before_placement` (Default: off)
  - Routes the PG nets before placement
  - If PG routes cannot be created due to DRC errors imposed by the placed cells, use this option to bypass the DRC errors to route the PG nets

- `-init_utilization` *`ratio`* (Default: 0.25)
  - Specifies the initial design utilization
  - Use this option to enlarge the core area if the default core size is too small to fit all the cells

Pin Access Checking Utility

New Options in Version 5.0

- `-back2back_placement` (Default: off)
  - Places cells back-to-back or vertically
  - Supports only single-height cells
- `-back2back_target_cell_offset` *`integer`* (Default: 0)
  - Specifies the offset distance for the target cell with respect to its neighbor cells
  - The offset distance will be N times the tile width



Positive offset moves cell A to the right
Negative offset moves cell A to the left

New in ver5.0

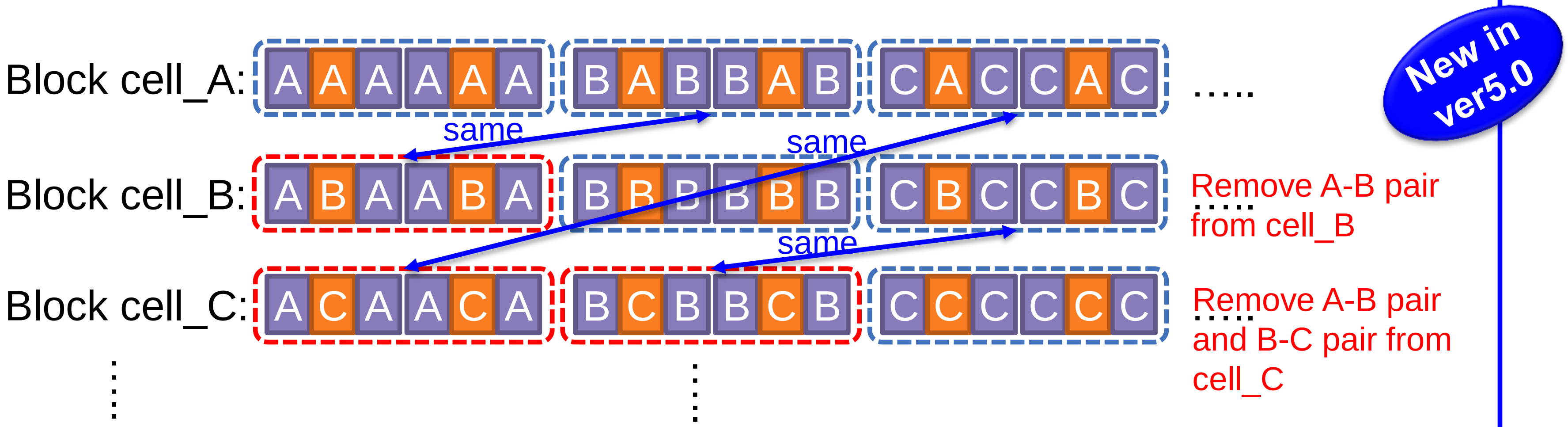
- `-fill_empty_rows_with_blockages {`*`metal_list`*`}` (Default: off)
  - Creates zero-spacing route guides in all empty rows of the specified metal layers
  - Disallows routing in the empty rows

Pin Access Checking Utility

New Options in Version 5.0

- `-no_empty_row` (Default: leave one empty for every placed row)
  - Places cells in consecutive rows
  - Supports only single-height cells

- `remove_duplicate_cell_pairs` (Default: off)
  - Removes repeating cell pairs in each test block "cell_xxx“
  - Reduces the netlist (database) size by half, which improves overall runtime



Block cell_A: A A A A A A | B A B B A B | C A C C A C …..

Block cell_B: A B A A B A | B B B B B B | C B C C B C …..

Block cell_C: A C A A C A | B C B B C B | C C C C C C …..

⋮ ⋮

Remove A-B pair from cell_B

Remove A-B pair and B-C pair from cell_C

Pin Access Checking Utility

Examples

1) -check_against_all_lib_cells

- The following command creates two test cells (cell_nand2.CEL for the nand2 library cell and cell_inv8.CEL for the inv8 library cell; each test cell contains all possible cell placements of the selected library cell against all the cells in the STD library

```
icc_shell> check_std_cell_pin_access -technology test.tf \
-std_lib_path STD -cells {nand2 inv8} -check_against_all_lib_cells
```

2) -cell_spacing_rule_file

- The following command creates a cell placement based on the cell spacing rule defined in the cell_spacing_rule.tcl file

```
icc_shell> check_std_cell_pin_access -technology test.tf \
-std_lib_path STD -cell_spacing_rule_file cell_spacing_rule.tcl
```

cell_spacing_rule.tcl

```
set_lib_cell_spacing_label -names {X} -left_lib_cells {AND*}
set_lib_cell_spacing_label -names {Y} -right_lib_cells {NAND*}
set_spacing_label_rule -labels {X Y} {0 1}
```

Pin Access Checking Utility

Examples (Continued)

3) `-cell_spacing_rule_file`, `-update_placement`

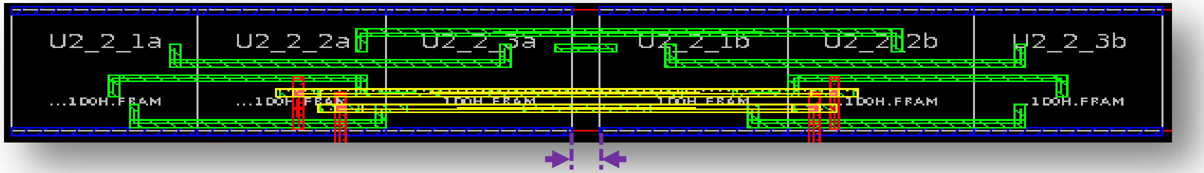
- The following command quickly updates the cell placement without re-creating the whole library; you can use this to recheck the pin accessibility on an existing library, but with a different cell spacing rule

```
icc_shell> check_std_cell_pin_access -technology test.tf \
-std_lib_path STD -cell_spacing_rule_file cell_spacing_rule.tcl \
-update_placement
```

4) `-group_spacing`

- The following command keeps two tile widths between the first and second set of cells

```
icc_shell> check_std_cell_pin_access -technology test.tf \
-std_lib_path STD -group_spacing 2
```



Group spacing (default: 1 tile width)

Example Script to Derive Cell Spacing Rules in Five Routing Iterations

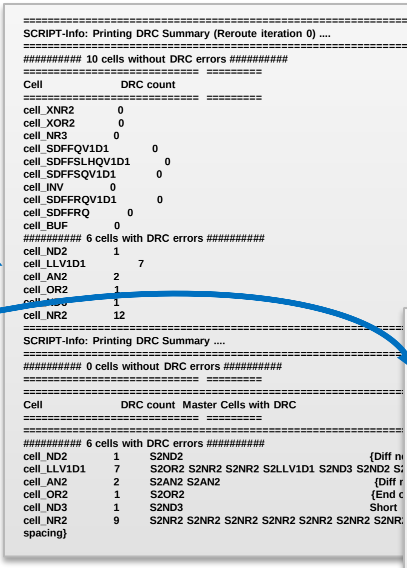
```
set stdLib "std_cell_library"
set techFile “test.tf“
set cellSpacingFile "cell_spacing_rules.tcl"
source ./check_std_cell_pin_access.tcl

check_std_cell_pin_access -technology $techFile -std_lib_path $stdLib
# Get cells with DRC violations and save them to $drc_cells
set drc_cells [check_std_cell_pin_access -technology $techFile -lib $lib -std_lib_path $stdLib -
report_cell_drc_only]
set num_drc_cell [llength $drc_cells]

# Run 5 iterations to derive all cell spacing rules or until reached 0 cell with drc
set derive_spacing_iterations 5
set count 1
while {$count <= $derive_spacing_iterations && $num_drc_cell > 0} {
  set drc_cells [check_std_cell_pin_access -technology $techFile -lib $lib -std_lib_path $stdLib -
report_cell_drc_only -derive_cell_spacing_constraint ${cellSpacingFile}.$count]
  set num_drc_cell [llength $drc_cells]
  # Update placement for cells with drcs
  check_std_cell_pin_access -technology $techFile -lib $lib -std_lib_path $stdLib -update_placement -
cell_spacing_rule_file ${cellSpacingFile}.$count -skip_routing -cells $drc_cells
  # Reroute cells with drcs
  check_std_cell_pin_access -technology $techFile -lib $lib -std_lib_path $stdLib -reroute_only -cells
$drc_cells -route_option_file $route_setup
  # get cells with drcs and save it to $drc_cells
  set drc_cells [check_std_cell_pin_access -technology $techFile -lib $lib -std_lib_path $stdLib -
report_cell_drc_only]
  set num_drc_cell [llength $drc_cells]
  incr count
}
```

Pin Access Checking Utility

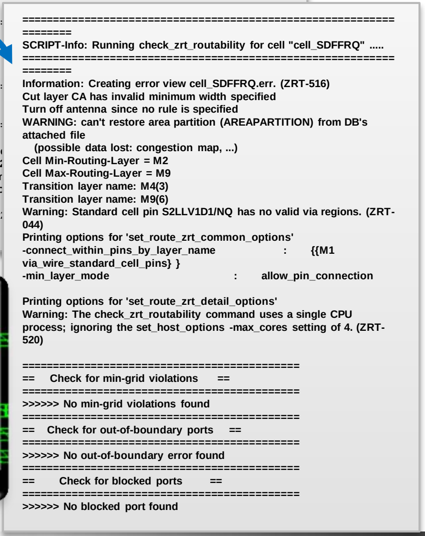
Results From the Pin Access Checking Utility

- Types of checks performed
  - Routing on individual library cells with all possible cell placement combinations
  - Pin access points
  - Blocked ports
  - Cell instance wire tracks
  - Pins out of boundary
  - Minimum grid
  - Pin design rules
  - Blockages

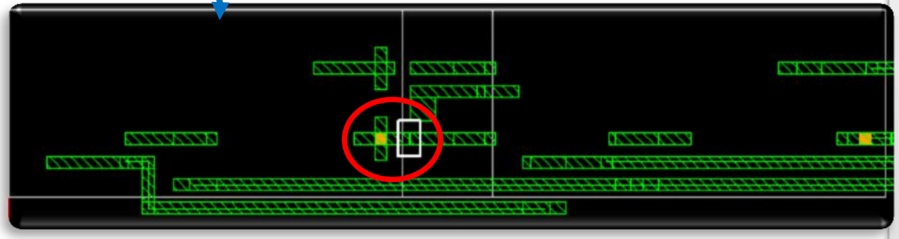
- Output
  - DRC summary report
  - `check_zrt_routability` report
  - DRC error view



```
=================================================================
SCRIPT-Info: Printing DRC Summary (Reroute iteration 0) ....
=================================================================
########## 10 cells without DRC errors ##########
==========================  ========
Cell                  DRC count
==========================  ========
cell_XNR2           0
cell_XOR2           0
cell_NR3           0
cell_SDFFQV1D1           0
cell_SDFFSLHQV1D1           0
cell_SDFFSQV1D1           0
cell_INV           0
cell_SDFFRQV1D1           0
cell_SDFFRQ           0
cell_BUF           0
########## 6 cells with DRC errors ##########
cell_ND2           1
cell_LLV1D1           7
cell_AN2           2
cell_OR2           1
cell_ND3           1
cell_NR2           12
=================================================================
SCRIPT-Info: Printing DRC Summary ....
=================================================================
########## 0 cells without DRC errors ##########
==========================  ========
=================================================================
Cell                  DRC count   Master Cells with DRC
==========================  ========
=================================================================
########## 6 cells with DRC errors ##########
cell_ND2           1        S2ND2                                             {Diff n
cell_LLV1D1        7        S2OR2 S2NR2 S2NR2 S2LLV1D1 S2ND3 S2ND2 S2
cell_AN2           2        S2AN2 S2AN2                                        {Diff r
cell_OR2           1        S2OR2                                             {End c
cell_ND3           1        S2ND3                                             Short
cell_NR2           9        S2NR2 S2NR2 S2NR2 S2NR2 S2NR2 S2NR2 S2NR2
spacing}
```



```
=========================================================
========
SCRIPT-Info: Running check_zrt_routability for cell "cell_SDFFRQ" .....
=========================================================
========
Information: Creating error view cell_SDFFRQ.err. (ZRT-516)
Cut layer CA has invalid minimum width specified
Turn off antenna since no rule is specified
WARNING: can't restore area partition (AREAPARTITION) from DB's
attached file
   (possible data lost: congestion map, ...)
Cell Min-Routing-Layer = M2
Cell Max-Routing-Layer = M9
Transition layer name: M4(3)
Transition layer name: M9(6)
Warning: Standard cell pin S2LLV1D1/NQ has no valid via regions. (ZRT-
044)
Printing options for 'set_route_zrt_common_options'
-connect_within_pins_by_layer_name                 :      {{M1
via_wire_standard_cell_pins} }
-min_layer_mode                                    :      allow_pin_connection

Printing options for 'set_route_zrt_detail_options'
Warning: The check_zrt_routability command uses a single CPU
process; ignoring the set_host_options -max_cores setting of 4. (ZRT-
520)

==========================================
==    Check for min-grid violations     ==
==========================================
>>>>>> No min-grid violations found
==========================================
==   Check for out-of-boundary ports    ==
==========================================
>>>>>> No out-of-boundary error found
==========================================
==       Check for blocked ports        ==
==========================================
>>>>>> No blocked port found
```



How Are Pin Access Issues Found?

Analyze check_zrt_routability Results

```
====================================================================
SCRIPT-Info: Running check_zrt_routability for cell "cell_SDFFRQ" .....
====================================================================
Information: Creating error view cell_SDFFRQ.err. (ZRT-516)
Cut layer CA has invalid minimum width specified
Turn off antenna since no rule is specified
WARNING: can't restore area partition (AREAPARTITION) from DB's attached file
   (possible data lost: congestion map, ...)
Cell Min-Routing-Layer = M2
Cell Max-Routing-Layer = M9
Transition layer name: M4(3)
Transition layer name: M9(6)
Warning: Standard cell pin S2LLV1D1/NQ has no valid via regions. (ZRT-044)
Printing options for 'set_route_zrt_common_options'
-connect_within_pins_by_layer_name          :     {{M1 via_wire_standard_cell_pins} }
-min_layer_mode                             :     allow_pin_connection

Printing options for 'set_route_zrt_detail_options'
Warning: The check_zrt_routability command uses a single CPU process; ignoring the set_host_options -max_cores setting of 4. (ZRT-520)

=============================================
==    Check for min-grid violations      ==
=============================================
>>>>>> No min-grid violations found
=============================================
==   Check for out-of-boundary ports    ==
=============================================
>>>>>> No out-of-boundary error found
=============================================
==      Check for blocked ports        ==
=============================================
>>>>>> No blocked port found
```

No violations!

How Are Pin Access Issues Found?

Check DRC Summary Output From the Script

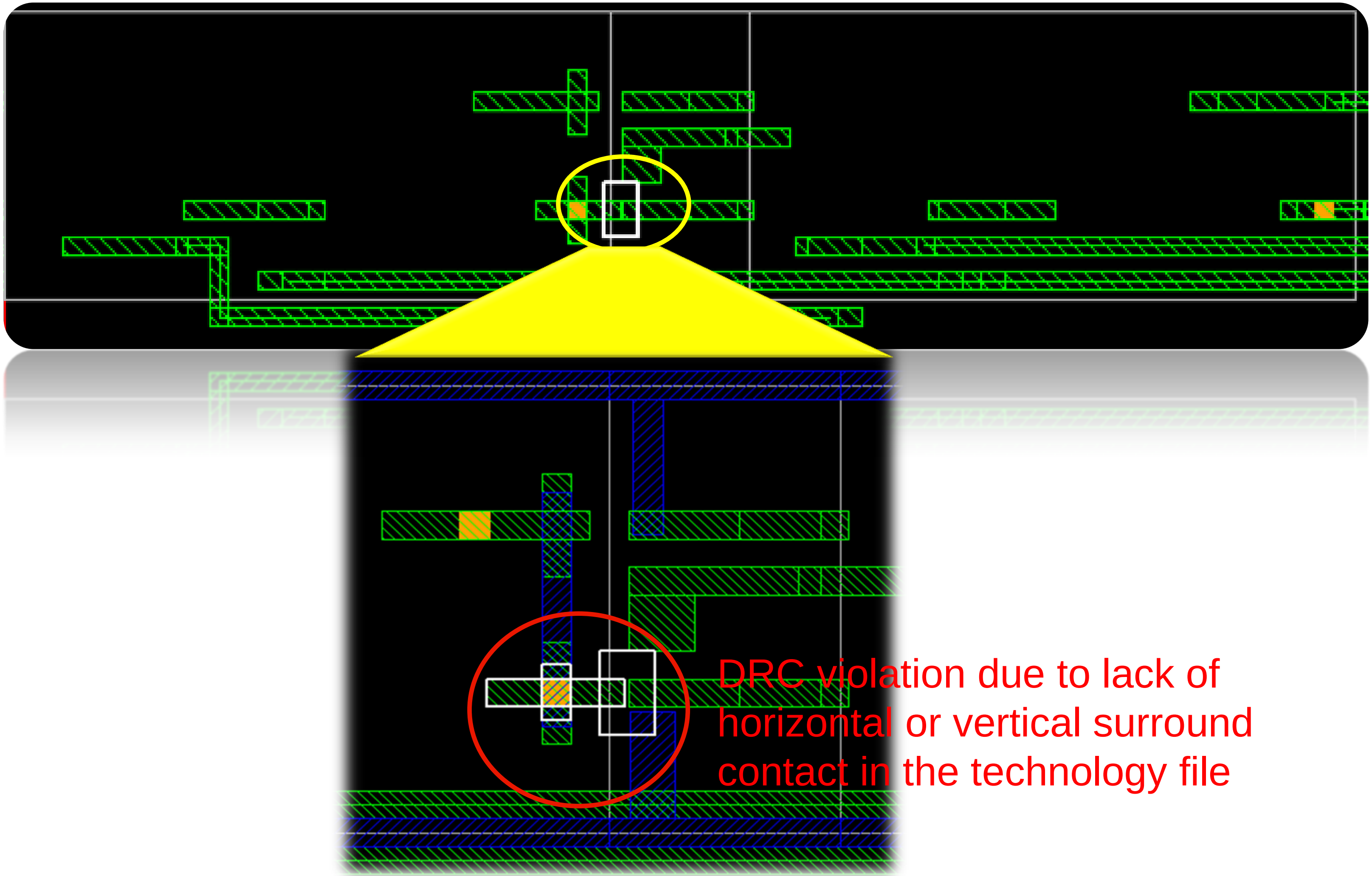
```
==================================================================
SCRIPT-Info: Printing DRC Summary (Reroute iteration 0) ....
==================================================================
########## 10 cells without DRC errors ##########
============================ =========
Cell                         DRC count
============================ =========
cell_XNR2            0
cell_XOR2            0
cell_NR3             0
cell_SDFFQV1D1            0
cell_SDFFSLHQV1D1         0
cell_SDFFSQV1D1           0
cell_INV             0
cell_SDFFRQV1D1           0
cell_SDFFRQ          0
cell_BUF             0
########## 6 cells with DRC errors ##########
cell_ND2             1
cell_LLV1D1               7
cell_AN2             2
cell_OR2             1
cell_ND3             1
cell_NR2             12
==================================================================
SCRIPT-Info: Printing DRC Summary ....
==================================================================
########## 0 cells without DRC errors ##########
============================ ========= ====================================================================== ======================
Cell                         DRC count  Master Cells with DRC                             DRC Types
============================ ========= ====================================================================== ======================
########## 6 cells with DRC errors ##########
cell_ND2             1      S2ND2                                  {Diff net spacing}
cell_LLV1D1          7      S2OR2 S2NR2 S2NR2 S2LLV1D1 S2ND3 S2ND2 S2AN2 S2AN2  {End of line spacing} {Diff net spacing}
cell_AN2             2      S2AN2 S2AN2                             {Diff net spacing}
cell_OR2             1      S2OR2                                   {End of line spacing}
cell_ND3             1      S2ND3                                  Short
cell_NR2             9      S2NR2 S2NR2 S2NR2 S2NR2 S2NR2 S2NR2 S2NR2 S2NR2 S2NR2  {Less than minimum area} {Diff net spacing}
```

Investigate the cells that have DRC violations with the error browser!

Possible problematic cells!

Pin Access Checking Utility

DRC Violations



DRC violation due to lack of horizontal or vertical surround contact in the technology file

Pin Access Checking Utility

Recommended Setup Before Running the Utility

- Make sure that icc_shell is in your UNIX search path
  - Example:

    ```
    set path = (/../os_platform/bin/icc_shell $path)
    ```
  - By default, the program uses icc_shell in the search path for parallel runs
- Add any required and non-persistent route settings to a file and specify the file name with the `-route_option_file` option

Pin Access Checking Utility

Recommended Step-by-Step Pin Checking Flow

1

```
check_std_cell_pin_access \
 -mode single_cell_only
```

- Fast checking pin access on standalone cell

2

```
check_std_cell_pin_access \
 -filter_cells_by_pin_count -1
```

- Analyze cell pin count in the standard cell library
- You can filter cells with fewer than specified pin count in the next step

3

```
check_std_cell_pin_access \
 -cells {} -exclude_cells {} \
 -filter_cells_by_pin_count n \
 -run_check_zrt_routability \
 -skip_routing
```

- Check pin access on all cell combinations
- Reduce the cell count with the –cells, -exclude_cells, and –filter_cells_by_pin_count options
- Check routability issues with the check_zrt_routability command before routing

4

```
check_std_cell_pin_access \
 -reroute_only \
 -num_cells_per_parallel_run n \
 -pin_access_check_tcl_path path
```

- Perform more accurate checking with routing
- Use –num_cells_per_parallel_run and –pin_access_check_tcl_path to run several divided jobs in parallel

5

```
check_std_cell_pin_access \
 -report_cell_drc_only
```

- Analyze DRC results

6

```
check_std_cell_pin_access \
 -reroute_drc_cells_only
```

- Reroute cells that still have DRC violations

Single command run (include all 6 steps):

check_std_cell_pin_access -technology xx.tf -std_lib_path STD -num_cells_per_parallel_run 5 –filter_cells_by_pin_count 3\ -pin_access_check_tcl_path ./check_std_cell_pin_access.tcl -reroute_drc_cells_iterations 2 -run_check_zrt_routability

Pin Access Checking Utility

Always Run Baseline Check First!!

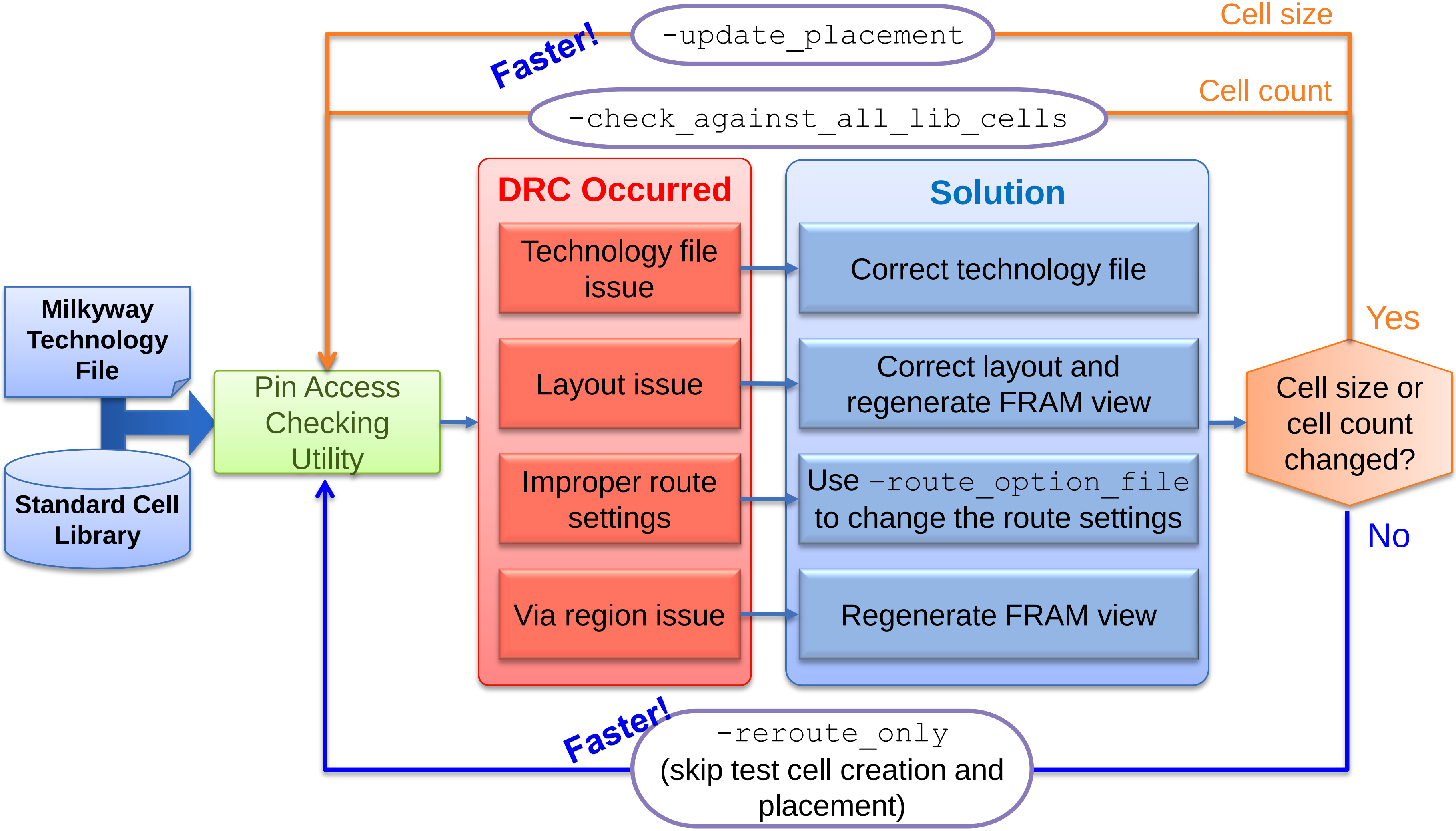
- Baseline pin accessibility check:
  - `-mode single_cell_only` + `-mode cell_by_cell_only`

- Advanced pin accessibility checks:
  - `-pg_route`
  - `-congested_netlist`
  - `-back2back_placement`
  - `-no_empty_row`
  - `-fill_empty_row_with_blockages`

- Feasibility check (big cells):
  - `-mode all_combo_in_one_cell_only`

Pin Access Checking Utility

Debugging Methodology (1)



Pin Access Checking Utility

Debugging Methodology (2)

- The pin access checking utility issues debugging messages in various stages of the run
- If anything goes wrong while running the utility, look for the “SCRIPT-Info” keyword for debugging messages
- Example:

```
SCRIPT-Info: Creating a hierachical verilog netlist (.dummy.v) .....
SCRIPT-Info: All library cells will be created as submodules of top cell "pin_access"
SCRIPT-Info: Dummmy verilog (.dummy.v) has been successfully created.
SCRIPT-Info: Reading verilog (.dummy.v) .......
SCRIPT-Info: Create library cells as plangroups .......
SCRIPT-Info: Split all library cells into individusl cells using "commit_fp_plan_groups" ...
SCRIPT-Info: Create placement for cell "cell_BFX10" .....
SCRIPT-Info: Running check_zrt_routability for cell "cell_BFX10" .....
SCRIPT-Info: Source routing option file "dpt_route_options.tcl" .....
SCRIPT-Info: Running route_zrt_auto (effort: med) on cell "cell_BFX10" .....
SCRIPT-Info: Number of cells to be checked: 16
SCRIPT-Info: Divide run into 4 jobs
SCRIPT-Info: Create run csh "check_pin_access_run.csh.0" and tcl "check_pin_access_run.tcl.0"
SCRIPT-Info: Create run csh "check_pin_access_run.csh.1" and tcl "check_pin_access_run.tcl.1"
SCRIPT-Info: Create run csh "check_pin_access_run.csh.2" and tcl "check_pin_access_run.tcl.2"
SCRIPT-Info: Create run csh "check_pin_access_run.csh.3" and tcl "check_pin_access_run.tcl.3“
SCRIPT-Info: Printing DRC Summary ....
```

Pin Access Checking Utility

Limitations and Documentation

- Limitations
  - The utility is an IC Compiler encrypted Tcl script; it can be used only in the IC Compiler environment
  - It has the same licensing requirements as running Zroute with 28-nm or 20-nm design rules
  - The data size, memory consumption, and runtime depend on number of checked cells; to avoid disk space or memory issues, you should reduce the number of checked cells to less than 2000
- Documentation
  - The latest version of this document and the pin access checking utility is available on the Synopsys SolvNet website

    “Application Note for Physical Library Analysis for Optimal 28/20nm Routing” (DocId:035309)

Pin Access Checking Utility

Summary

Fully automated and easy to use

- Requires minimal inputs: Milkyway technology file and standard cell library (with FRAM views)
- Requires minimum IC Compiler usage knowledge

Thorough checking

- Runs real routing with all kinds of cell placement combinations

Aligned with the latest Zroute technology

- Always supports the latest design rules and router enhancements

Quick turnaround time (within a day)

- Physical library developer quickly identifies pin access or technology file problem
- Can run a second pin verification after fix

Agenda

- Forthcoming challenges at 20-nm process node and below
- Pin access checking utility (updated in version 5.0)
- Win-win for customers and Synopsys with pin access analysis
- Success stories for optimal standard cell pin access
- Appendix
  - Case study and recommendations

Win-Win for Customers and Synopsys

Customers

- Detect and fix potential pin access issues in the early library development stage
- Shorten library development cycle
- Avoid seeing DRC surprises at place and routing stage due to bad standard cell library design

Synopsys

- IC Compiler (Zroute) friendly library
- Leverage Synopsys resources to spend more effort on development and support of advanced rules and new router features

Agenda

Forthcoming challenges at 20-nm process node and below

Pin access checking utility (updated in version 5.0)

Win-win for customers and Synopsys with pin access analysis

Success stories for optimal standard cell pin access

Appendix

- Case study and recommendations

Success Stories

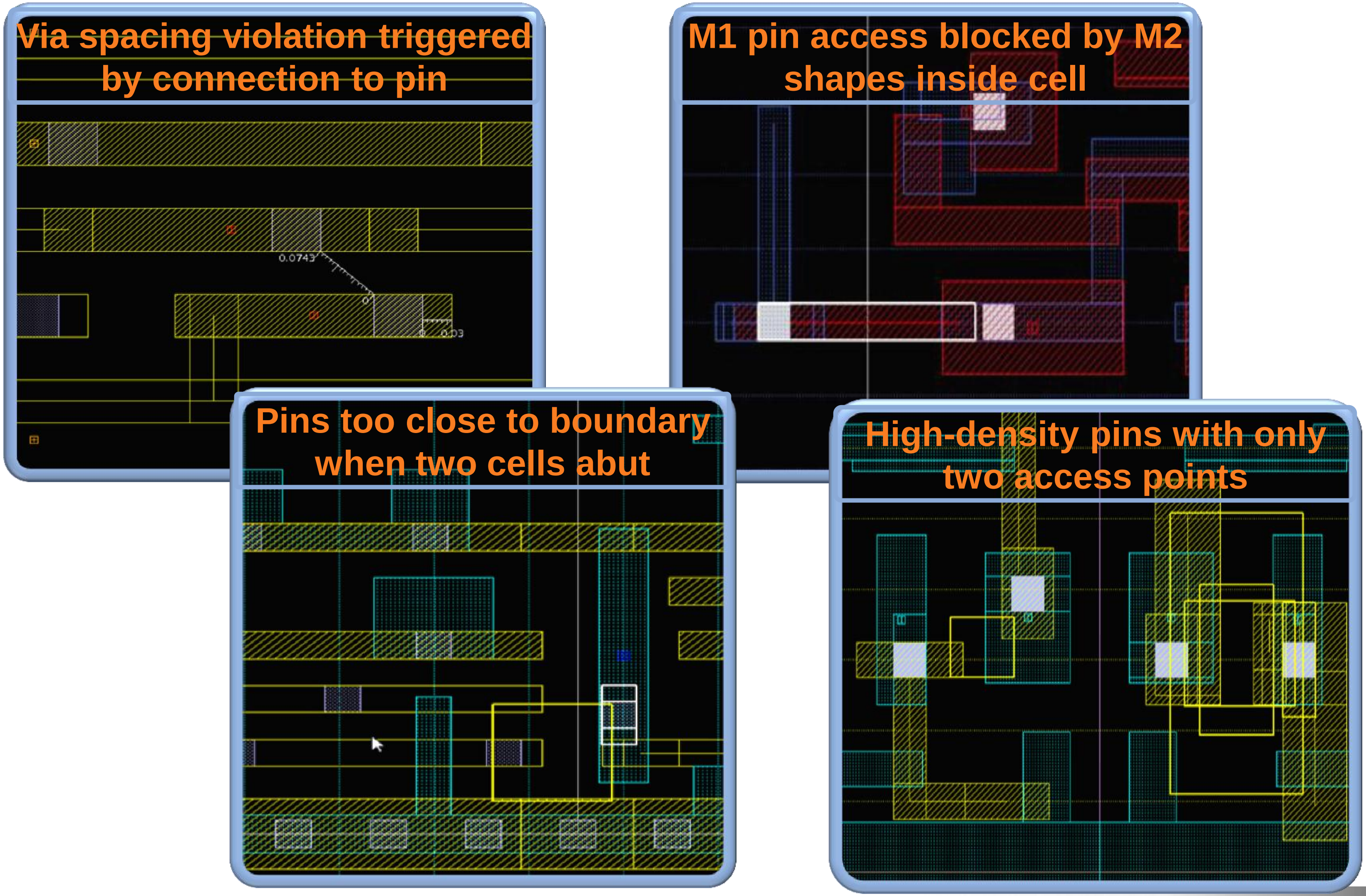
7 Libraries Analyzed With -mode all (Two Process Nodes, Two Foundries)

Library	Process	Cell height (pitch)	No. of cells	Runtime (4 threads)	No. of pin access issues
Lib #1 (HD)	28nm	9.5X	1138	~24 hrs	1
Lib #2 (HS)	28nm	12.5X	1138	~24 hrs	3
Lib #3	20nm	9X	252	5.5 hrs	4
Lib #4	20nm	9X	109	3.5 hrs	2
Lib #5	20nm	9X	83	3.5 hrs	0
Lib #6	20nm	9X	57	5.5 hrs	3
Lib #7	20nm	10X	60	6.7 hrs	5

- Factors that affect runtime:
  - Number of cells in the library
  - Number of DRC violations in the cells
    (more routing iterations for more DRC violations)
  - DRC rules defined in the technology file

Types of Pin Access Issues Found

DRC Issues Not Detectable at the Layout Design Stage!



Via spacing violation triggered by connection to pin

M1 pin access blocked by M2 shapes inside cell

Pins too close to boundary when two cells abut

High-density pins with only two access points

Agenda

Forthcoming challenges at 20-nm process node and below

Pin access checking utility (updated in version 5.0)

Win-win for customers and Synopsys with pin access analysis

Success stories for optimal standard cell pin access

Appendix

- Case study and recommendations

Agenda

Forthcoming challenges at 20-nm process node and below

Pin access checking utility (updated in version 5.0)

Win-win for customers and Synopsys with pin access analysis

Success stories for optimal standard cell pin access

Appendix

- Case study and recommendations

Content

- Standard cell design routing layer recommendations
  - General recommendations
  - 20-nm specific recommendations

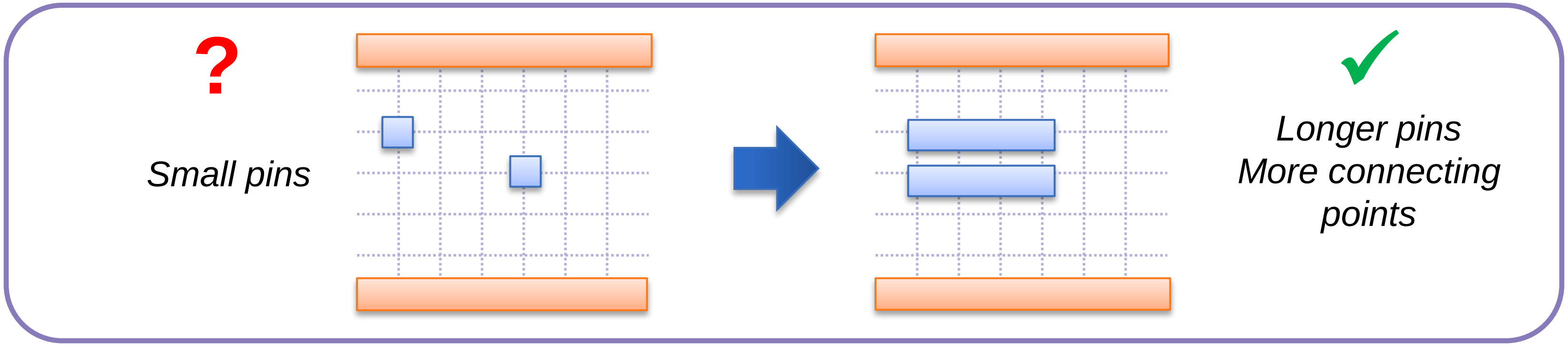
- Synopsys internal Tcl script to check pin accessibility
  - Pin accessibility checking utility

Standard Cell Design Routing Layer Recommendations

- Introduction
  - These are recommendations from the router’s point of view
    - Only metal layer recommendations related to routing
    - To be balanced with other objectives, such as minimizing cell areas
  - The recommendations are general, not specific to Zroute
    - Some routers might handle certain difficulties slightly differently than others

General Recommendations - 1/3

- Longer pins - Give the router more flexibility in making connections
  - Helpful when a specific pin layer is not to be used by the router



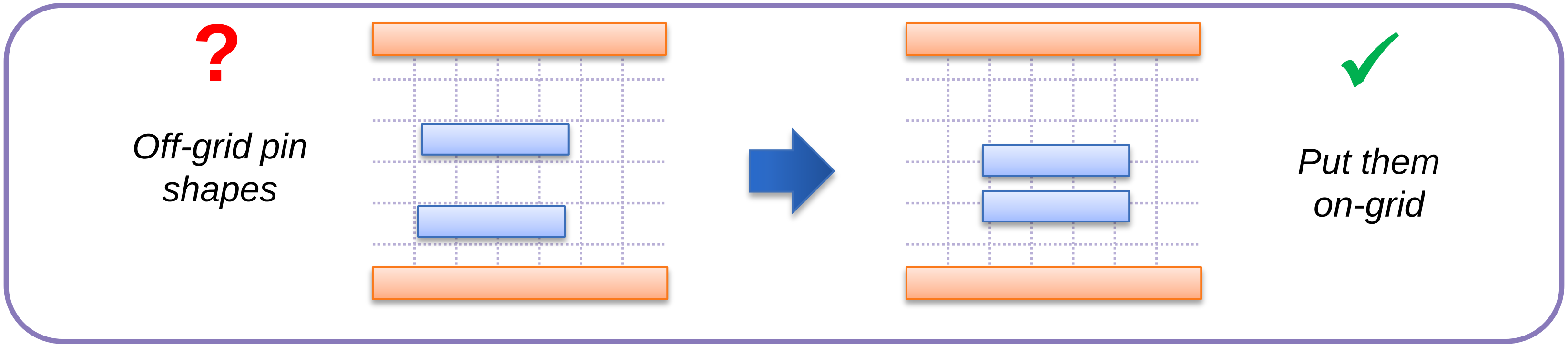
?

Small pins

✓

Longer pins
More connecting points

- More on-grid connecting points for pins – Better routing resource usage
  - Routing shapes are best packed when on-grid



?

Off-grid pin shapes

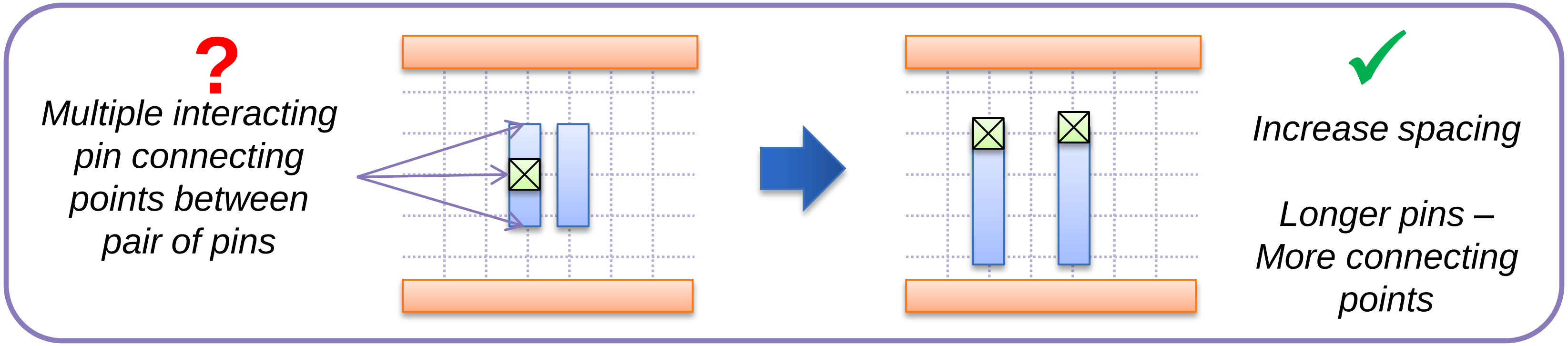
✓

Put them on-grid

General Recommendations - 2/3

Fewer Adjacent Pin Connection Interactions

- Interacting pin connecting points between a pair of pins



?

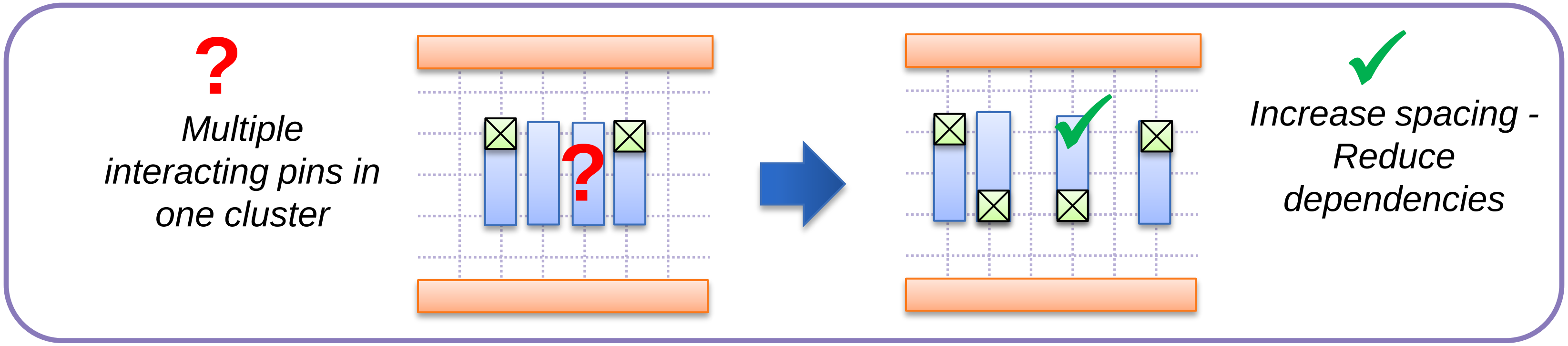
Multiple interacting pin connecting points between pair of pins

✓

Increase spacing

Longer pins – More connecting points

- Interacting pins in one cluster



?

Multiple interacting pins in one cluster

✓

Increase spacing - Reduce dependencies

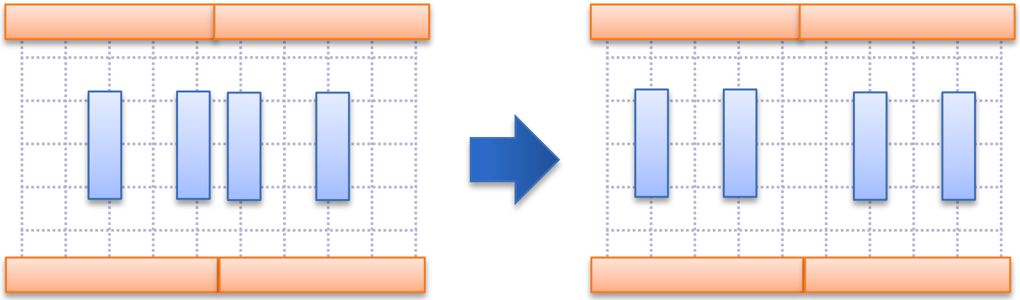
General Recommendations - 3/3

Fewer Adjacent Pin Connection Interactions

- Pin shapes at cell boundaries that can interact with adjacent cell pin shapes

?

Pin shapes at cell boundaries interacting with adjacent cell pin shapes



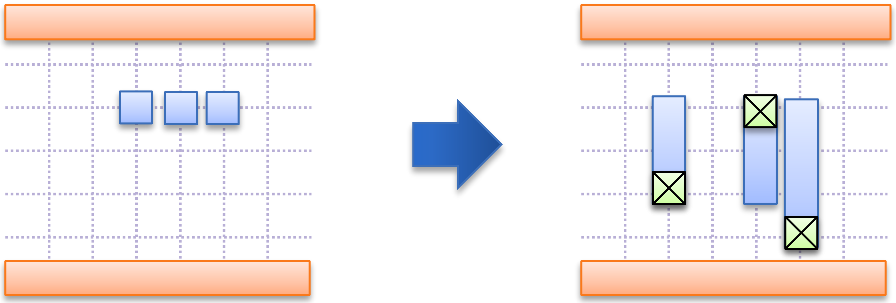
✓

Keep pins shapes away from cell boundaries

- Aligned pin shapes competing for common routing resources

?

Aligned pin shapes competing for common routing resources



✓

Increase spacing - Reduce dependencies

Longer pins - More connecting points

General Recommendations

Summary

- Longer pins - Give router more flexibility in making connections
  - Helpful when a specific pin layer is not to be used by router
  - Helpful for tricky configurations when there is a consistent good way to connect to the same pins
- More on-grid connecting points for pins – Better routing resource usage
  - Routing shapes are best packed when on-grid
- Fewer adjacent pin connection interactions – Routing process makes one connection at a time, not as effective in handling heavy interactions – Prefers fewer of the following:
  - Interacting pin connecting points between a pair of pins
  - Interacting pins in one cluster
  - Pin shapes at cell boundaries that can interact with adjacent cell pin shapes
  - Aligned pin shapes competing for common routing resources

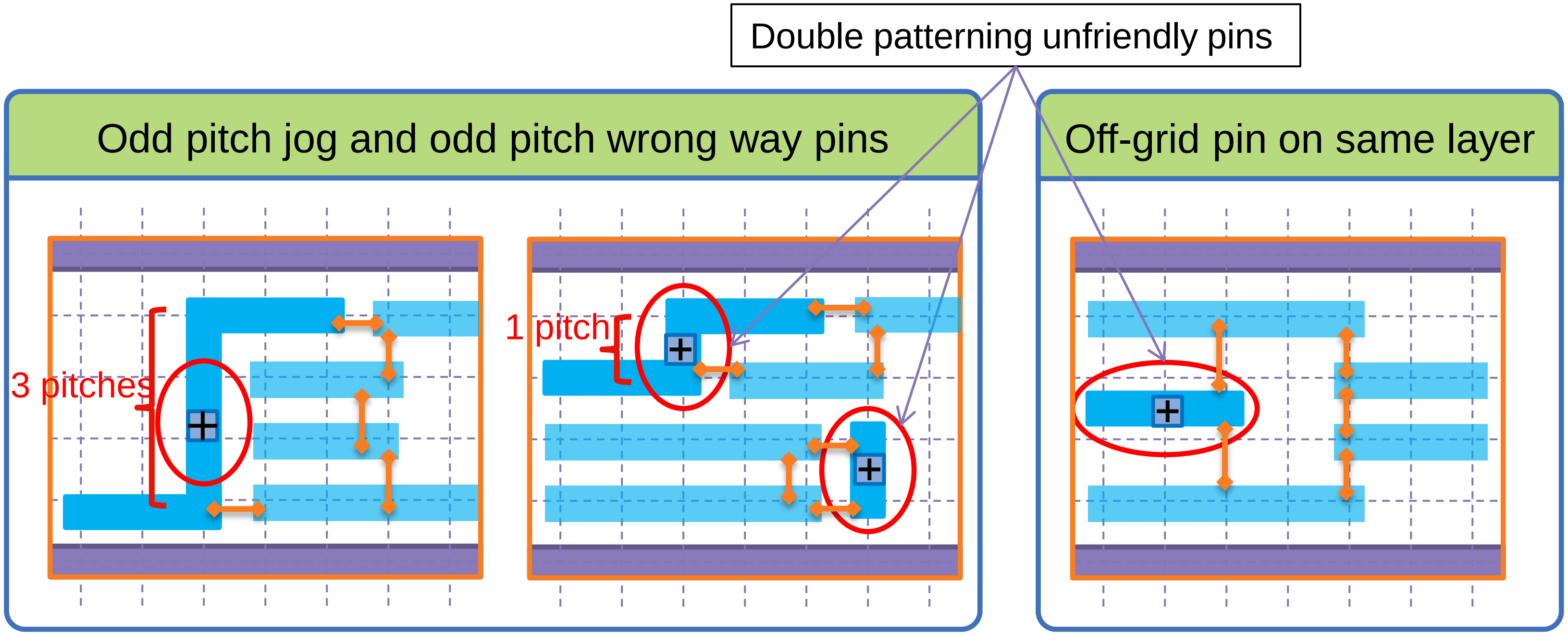
These recommendations became more important at 20 nm

More geometries, more rules and limitations, closer proximity

20-nm Specific Recommendations (1/4)

Double Patterning Considerations

- Avoid shapes that have the potential to cause double patterning cycles on the same routing layer
  - Odd pitch jogs and odd pitch wrong way shapes
  - Off-grid shapes that block even tracks and have double patterning interaction with adjacent shapes



Double patterning unfriendly pins

Odd pitch jog and odd pitch wrong way pins

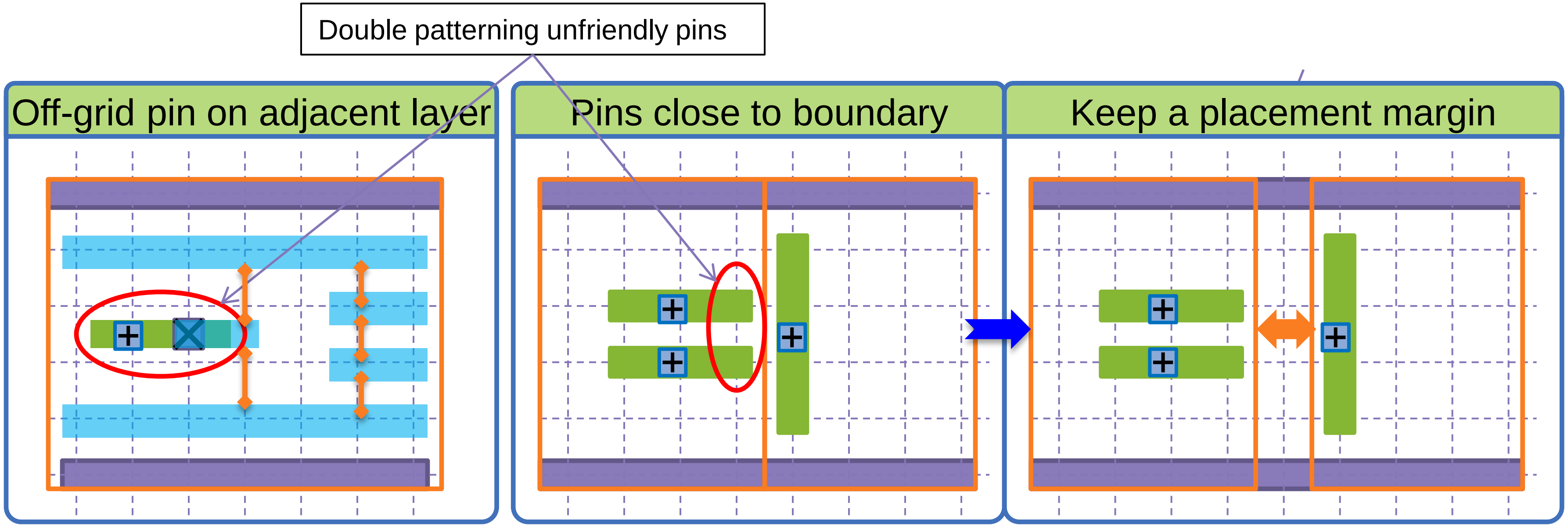
Off-grid pin on same layer

⊞ M2 pin ☒ Via1 ■ M2 routing

20-nm Specific Recommendations (2/4)

Double Patterning Considerations

- Avoid shapes that cause double patterning cycles on adjacent layers
  - Prefer pin shapes to be accessible by vias on grid from adjacent layers
- Avoid boundary shapes that can cause intercell double patterning cycles
  - We do have placement capability to space such cells apart at the cost of utilization. To enable double-patterning-aware placement (`enable_double_patterning_rules`), coloring information must be present as data types in the FRAM view and double patterning rules must be defined in the technology file.



Double patterning unfriendly pins

Off-grid pin on adjacent layer

Pins close to boundary

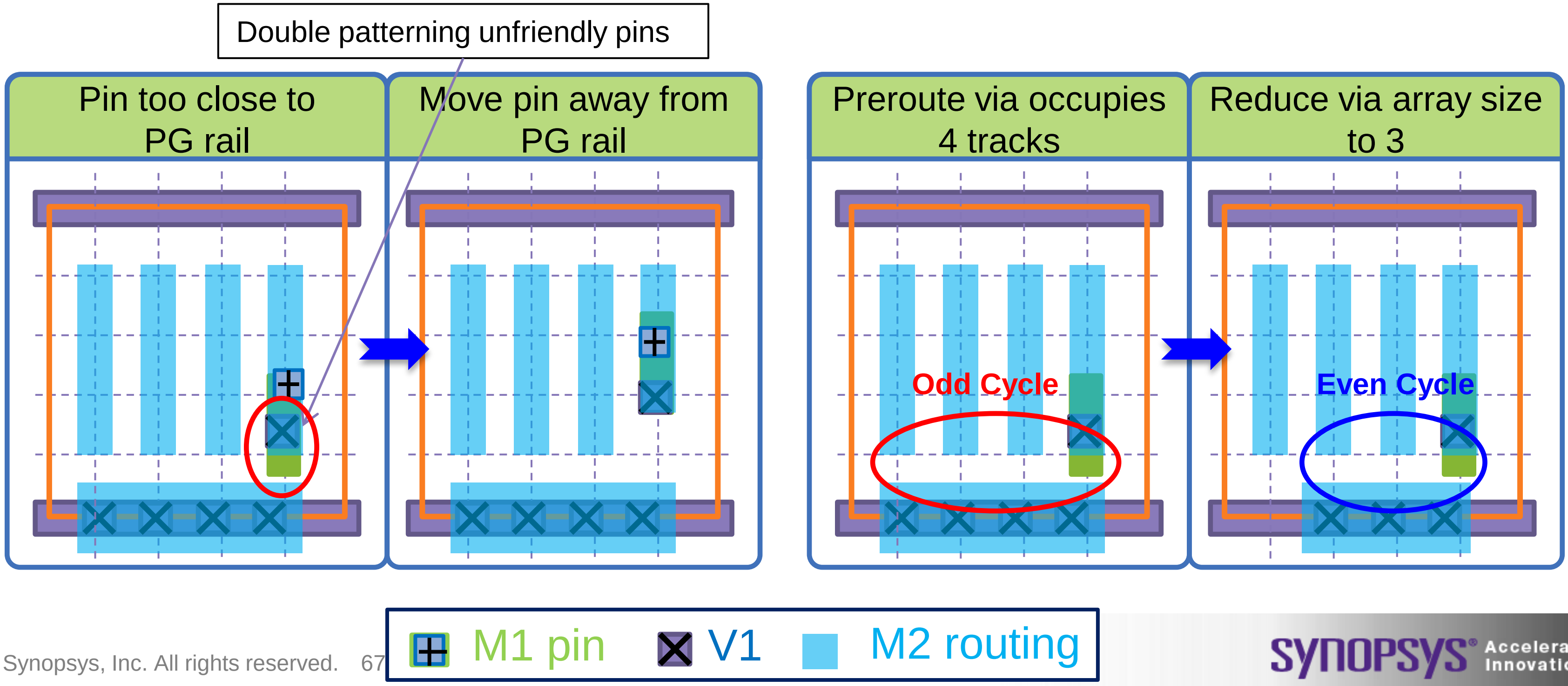
Keep a placement margin

⊞ M1 pin ☒ V1 ■ M2 routing

20-nm Specific Recommendations (3/4)

Double Patterning Considerations

- Avoid shapes too close to a PG rail
  - Try not to design a small pin close to a PG rail
- Avoid having preroute via arrays that occupy an even number of tracks
  - Use `set_preroute_advanced_via_rule` to adjust the via array size



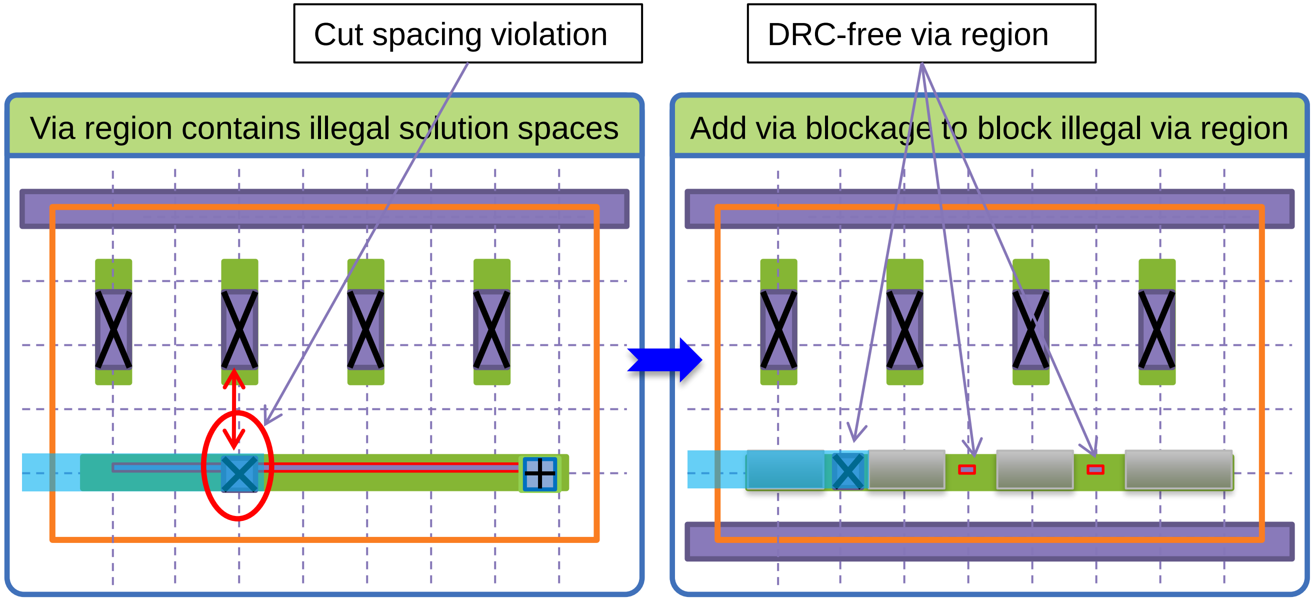
Double patterning unfriendly pins
Pin too close to PG rail
Move pin away from PG rail
Preroute via occupies 4 tracks
Reduce via array size to 3
Odd Cycle
Even Cycle
⊞ M1 pin
☒ V1
M2 routing

SYNOPSYS® Accelerating Innovation

20-nm Specific Recommendations (4/4)

Blockage, Pin, and Via Extraction Limitation

- Current blockage, pin, and via extraction (BPV) does not honor the generalized cut spacing rule
  - Via region might contain a lot of illegal solution spaces if the pin is in the proximity of a bar or large via
  - Router will have difficulty finding a DRC-free spot for dropping Via1
- Solution
  - Cover illegal via region with via blockage in the CEL view and redo BPV


Cut spacing violation
DRC-free via region
Via region contains illegal solution spaces
Add via blockage to block illegal via region

Generalized Cut Spacing Rule:

```
DesignRule      {
  layer1        = "V1"
  layer2        = "V1"
  cut1TblSize   = 4
  cut2TblSize   = 4
  cut1NameTbl = (Vsm,Vv,Vh_V,Vlg)
  cut2NameTbl = (Vsm,Vv,Vh_V,Vlg)
  orthoSpacingExcludeCornerTbl = …
  sameNetXMinSpacingTbl = …
  sameNetYMinSpacingTbl = …
  sameNetCenterMinSpacingTbl = …
  diffNetYMinSpacingTbl = …
  diffNetCenterMinSpacingTbl = …
}
```

⊞ M1 pin ☒ V1 🟦 M2 routing ⬜ V1 blockage

20-nm Specific Recommendations

Summary

- Avoid having the following pin shapes that could induce Zroute to get trapped in a double patterning unfriendly environment
  - Odd pitch jog
  - Odd pitch wrong way
  - Not on routing grid
  - Near cell boundary
- Avoid prerouting to form a double patterning cycle link with neighboring fixed shapes
  - Use `set_preroute_advance_vias_rule` to specify a smaller via array size
- Assist router to achieve faster DRC convergence
  - The more clear guidelines given in the FRAM view, the faster the runtime. You can manually add via or metal blockages to cover the design rules that are not supported in BPV.

SYNOPSYS®

Accelerating Innovation