SYNOPSYS®
Silicon to Software™

IC Synthesis and Optimization with Fusion Compiler

Synopsys Armenia Educational Department

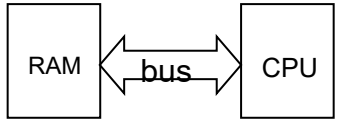
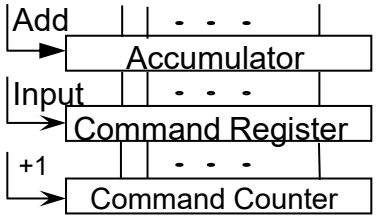
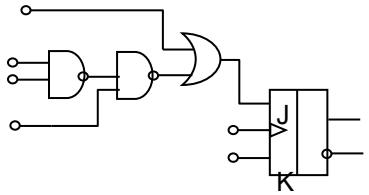
Course Overview

- Introduction
- Timing and Area Constraints
- Multi-Corner Multi-Mode and Mapping
- Application Options and Compile Flow
- Floorplan
- Placement
- Clock Tree Synthesis (not covered)
- Routing (not covered)
- Signoff (not covered)

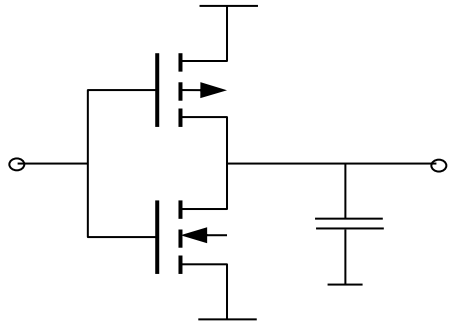
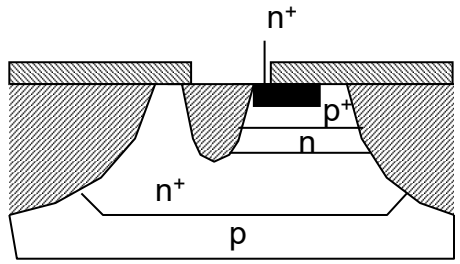
Agenda

- ***IC Design Flow and Implementation Concepts***
- Fusion Compiler Input Data
- Starting Fusion Compiler
- Graphical User Interface

Design Levels

Level	Modeling Object	Example of Modeling Object	Hardware Description Language Used
System (Electronic System Level – ESL)	Structural Circuit		C/C++ System Verilog System C
Register-Transfer Level (RTL)	Functional Circuits on the level of multibit devices – registers, and data transfer between them		System Verilog Verilog VHDL
Gate Level (Gate level netlist, Logic circuit)	Circuit containing logic gates (AND, OR, etc.) and flip-flops		

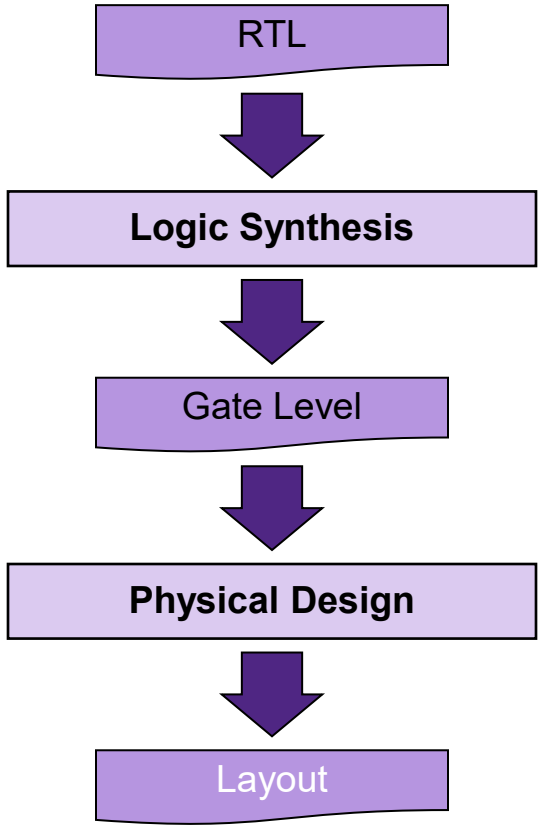
Design Levels (2)

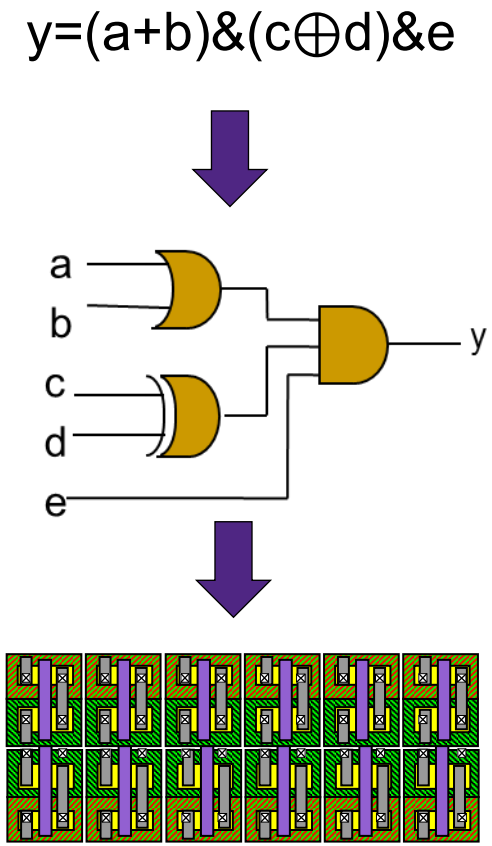
Level	Modeling Object	Example of Modeling Object	Hardware Description Language Used
Circuit Level (Transistor Level, SPICE Netlist)	Electrical Circuit		SPICE CDL
Device Level	IC Components		-

Main Concepts

- Synthesis
  - The process which converts an abstract form of desired circuit behavior into a design implementation in terms of logic gates
- Optimization
  - Changing design to achieve design goal (required by specification)

Main Design Steps





Specification

Specification					
Design Description					
Circuit must turn lights when a button is pressed, or ask for recharge if battery charge is low					
N0	Parameter description	Min	Typ	Max	Units
1.	Process	Modeling of Process Variation			
2.	Voltage	1.08	1.2	1.32	V
3.	Temperature	-40		125	^{0}C
4.	Power Dissipation			100	mW
5.	Die Area			2	um^2
6.	Clock frequency	1GHz			
….	….	….	….	….	….
….	….	….	….	….	….
….	….	….	….	….	….

Specification is the list of goals that should be achieved in the given design.

Design Description

Specification					
Design Description					
Circuit must turn lights when a button is pressed, or ask for recharge if battery charge is low					
N0	Parameter description	Min	Typ	Max	Units
1.	Process	Modeling of Process Variation			
2.	Voltage	1.08	1.2	1.32	V
3.	Temperature	-40		125	^{0}C
4.	Power Dissipation			100	mW
5.	Die Area			2	um^2
6.	Clock frequency	1GHz			
….	….	….	….	….	….
….	….	….	….	….	….
….	….	….	….	….	….

Design description (behavior) must be translated to Hardware Description Language (HDL) understandable by EDA tools.

RTL level description is used as input for synthesis.

RTL Example

```
if button1_pressed
  if (battery_charge > 10)
   turn_on_light();
  else
   prompt_for_recharge();
```

Operating Conditions

Specification					
Design Description					
Circuit must turn lights when a button is pressed, or ask for recharge if battery charge is low					
N0	Parameter description	Min	Typ	Max	Units
1.	Process	Modeling of Process Variation			
2.	Voltage	1.08	1.2	1.32	V
3.	Temperature	-40		125	^{0}C
4.	Power Dissipation			100	mW
5.	Die Area			2	um^2
6.	Clock frequency	1GHz			
….	….	….	….	….	….
….	….	….	….	….	….
….	….	….	….	….	….

Design depends on operating conditions

Design parameters change if designs operates at different

- Process variation
- Voltage change
- Temperature

Design is designed to work at different combinations of P, V, T

- Fast transistors, 1.08V, 1250C
- Slow transistors, 1.2V, 1250C

This cases are called PVT corners

Process variations are usually generalized as: fast , slow, typical

Design Constraints

Specification					
Design Description					
Circuit must turn lights when a button is pressed, or ask for recharge if battery charge is low					
N0	Parameter description	Min	Typ	Max	Units
1.	Process	Modeling of Process Variation			
2.	Voltage	1.08	1.2	1.32	V
3.	Temperature	-40		125	^{0}C
4.	Power Dissipation			100	mW
5.	Die Area			2	um^2
6.	Clock frequency	1GHz			
….	….	….	….	….	….
….	….	….	….	….	….
….	….	….	….	….	….

Design goals are specified as constraints .

Power ≤ 100 mW
Area < 2 um^2
Frequency > 1 GHz

Design constraints are used as input for synthesis.

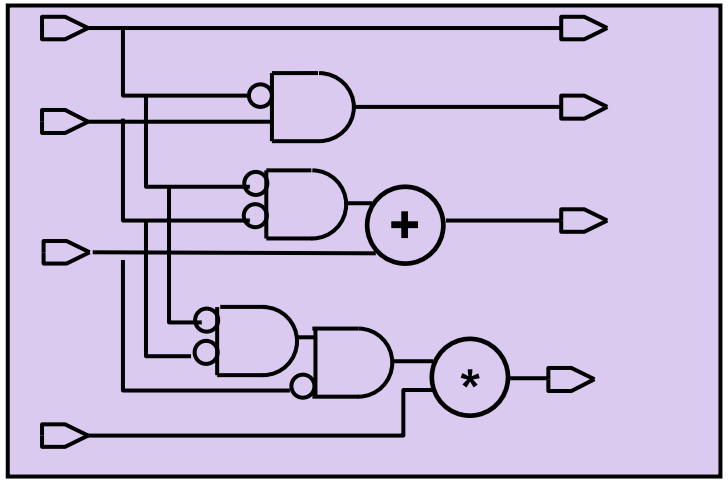
Logic Synthesis Steps

```
if button1_pressed
  if (battery_charge > 10)
   turn_on_light();
  else
   prompt_for_recharge();
```

Design RTL description

Translation

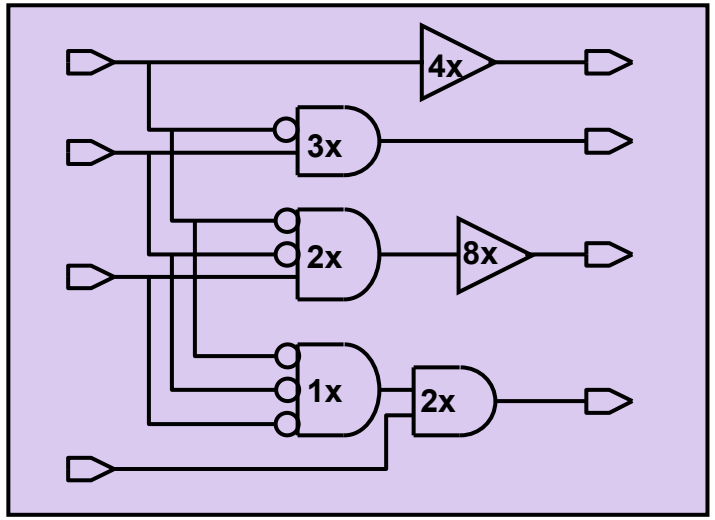
Technology Independent Circuit



Technology **Independent** Circuit is logic circuit which fully implements function described but is built from Generic Boolean Gates and basic operators.

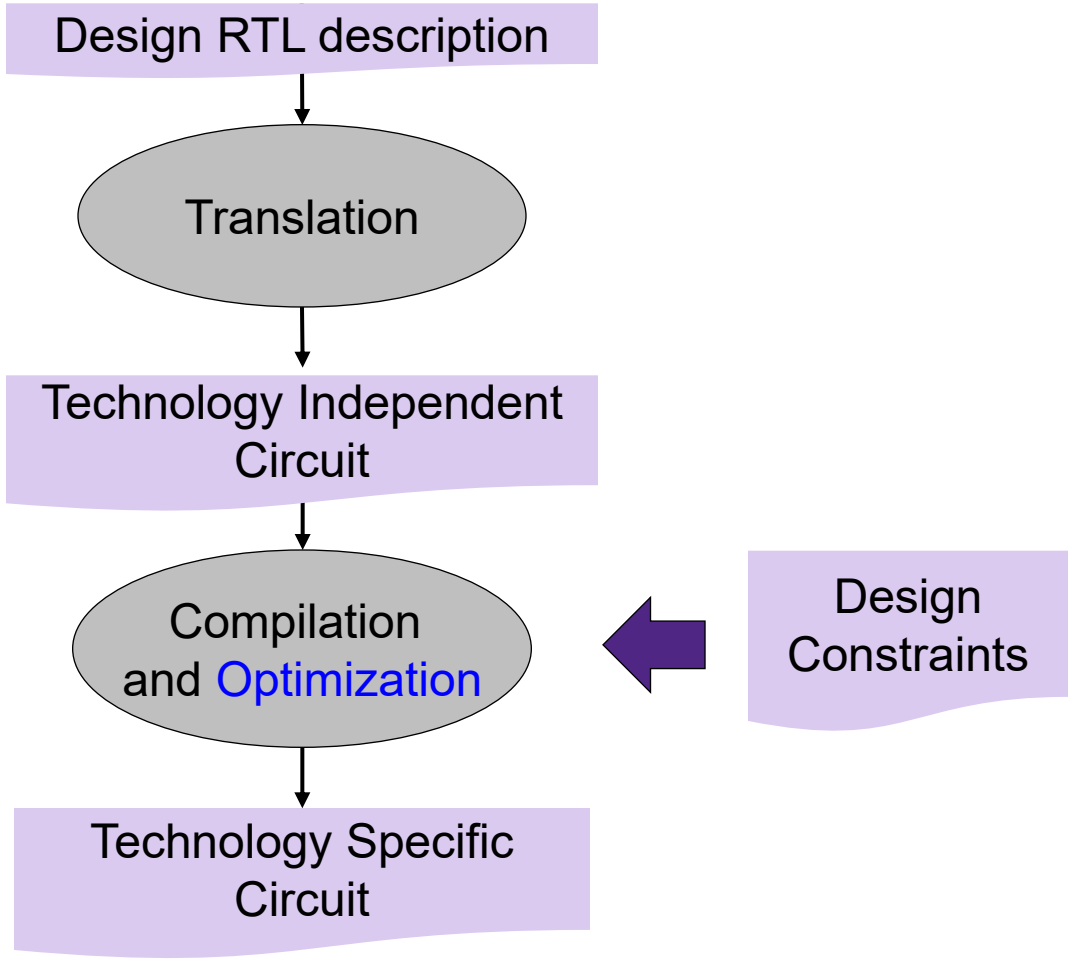
Compilation and Optimization

Technology Specific Circuit



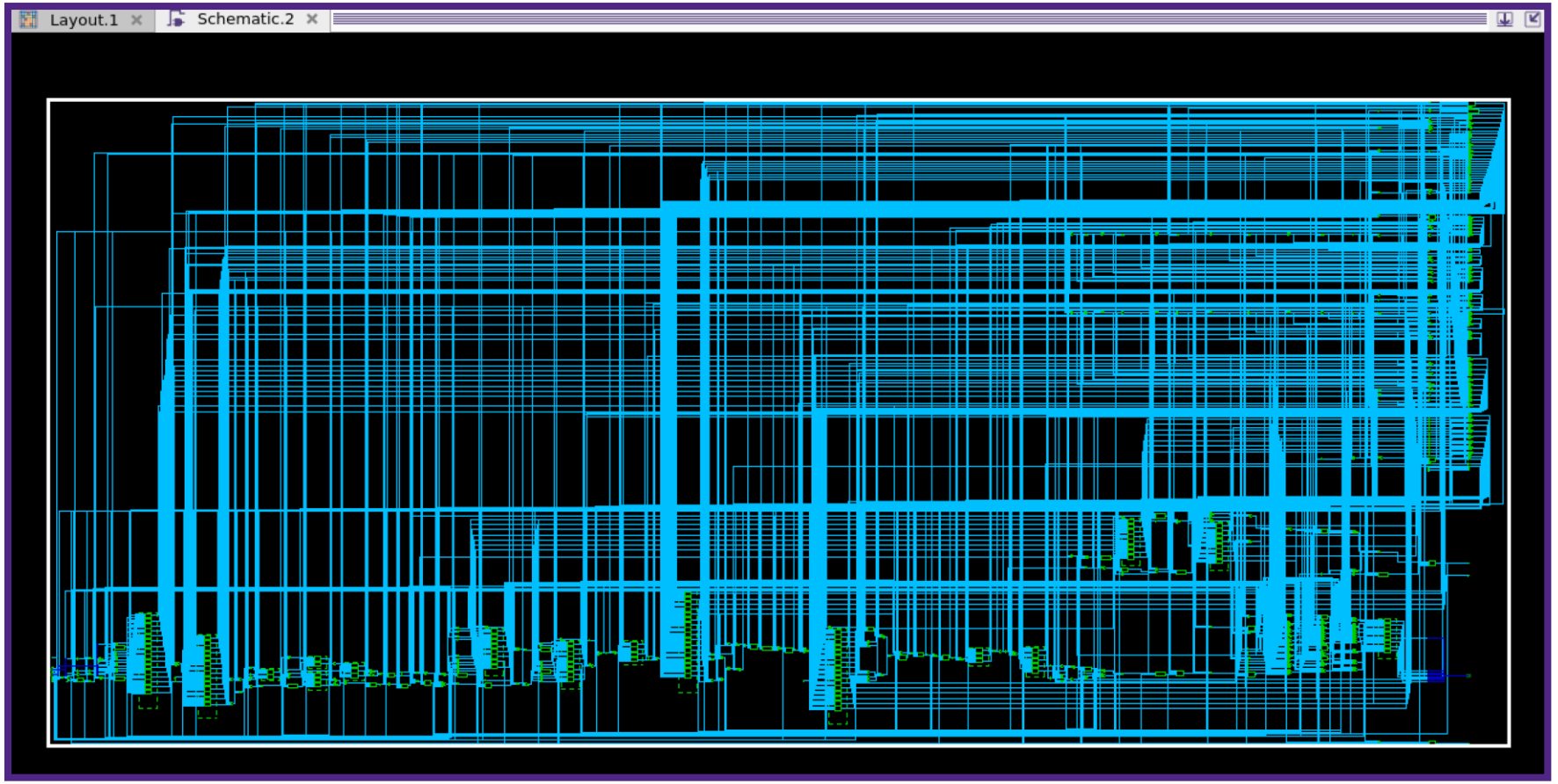
Technology **Specific** Circuit is obtained from independent one by replacing all components by real blocks (standard cells). This replacement process is also called **mapping**.

Constraint-Driven Synthesis

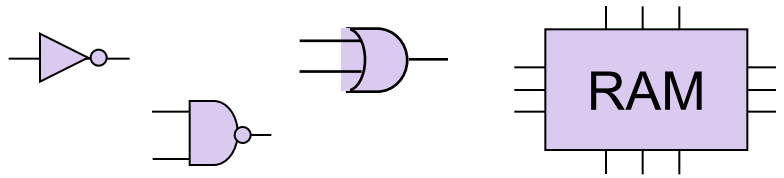


User can control synthesis process by providing constraints. Constraints are considered in the optimization step

Logic Circuit



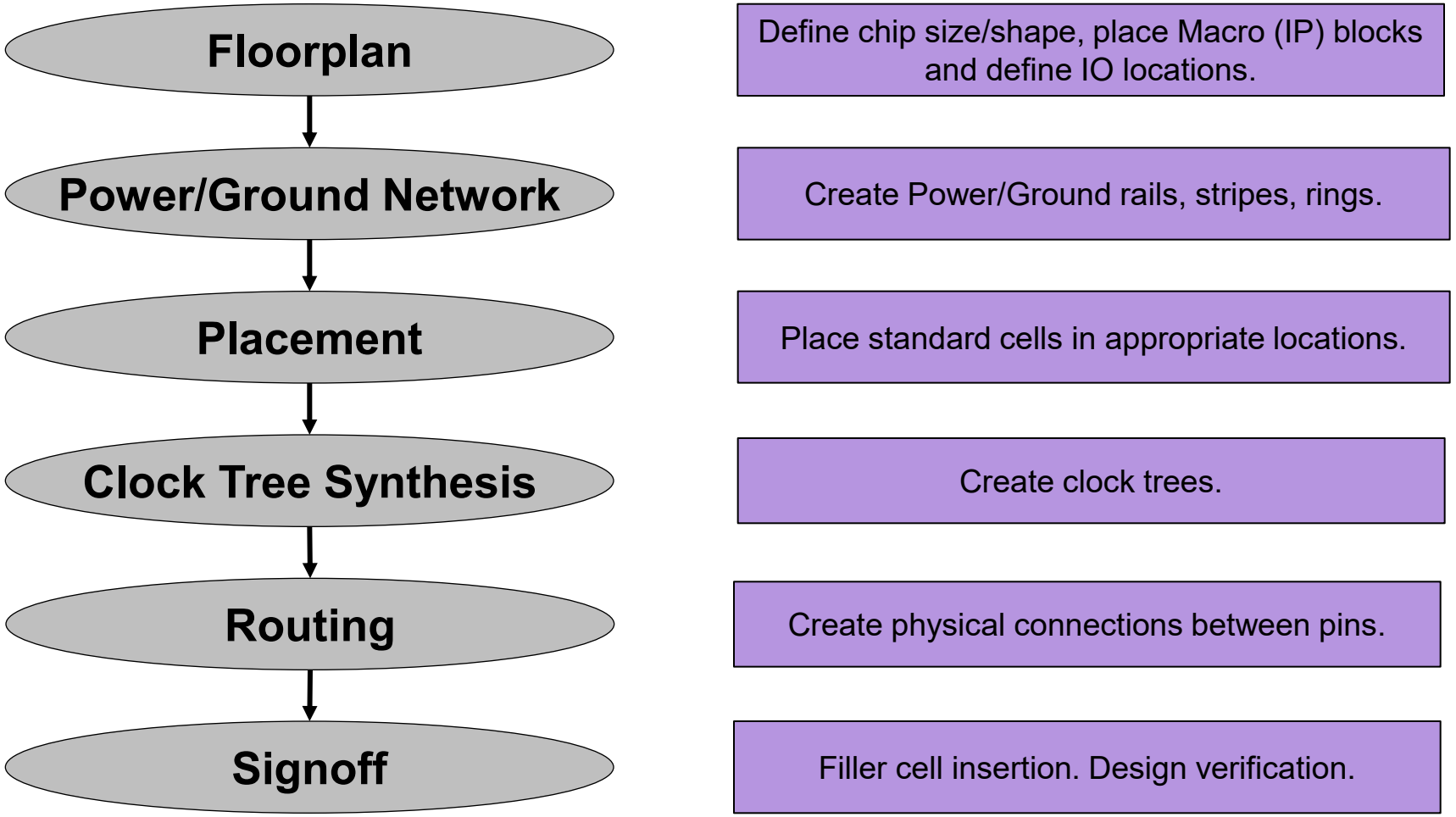
- Logic circuit consists of digital standard cells and Intellectual Property (IP) blocks.



Digital Standard Cells

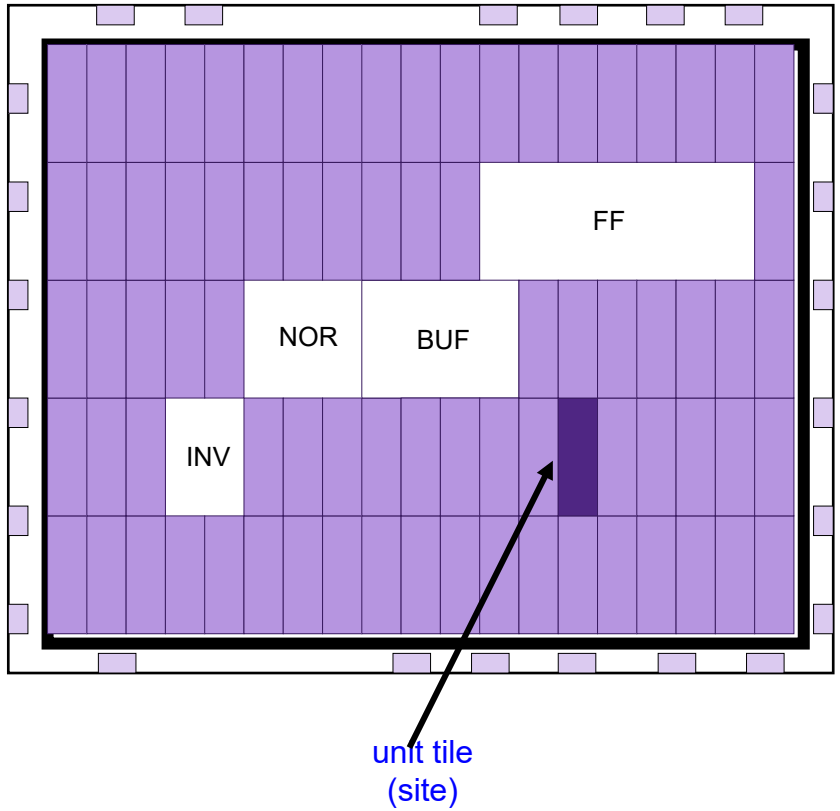
Intellectual property (IP) block

Physical Design Steps



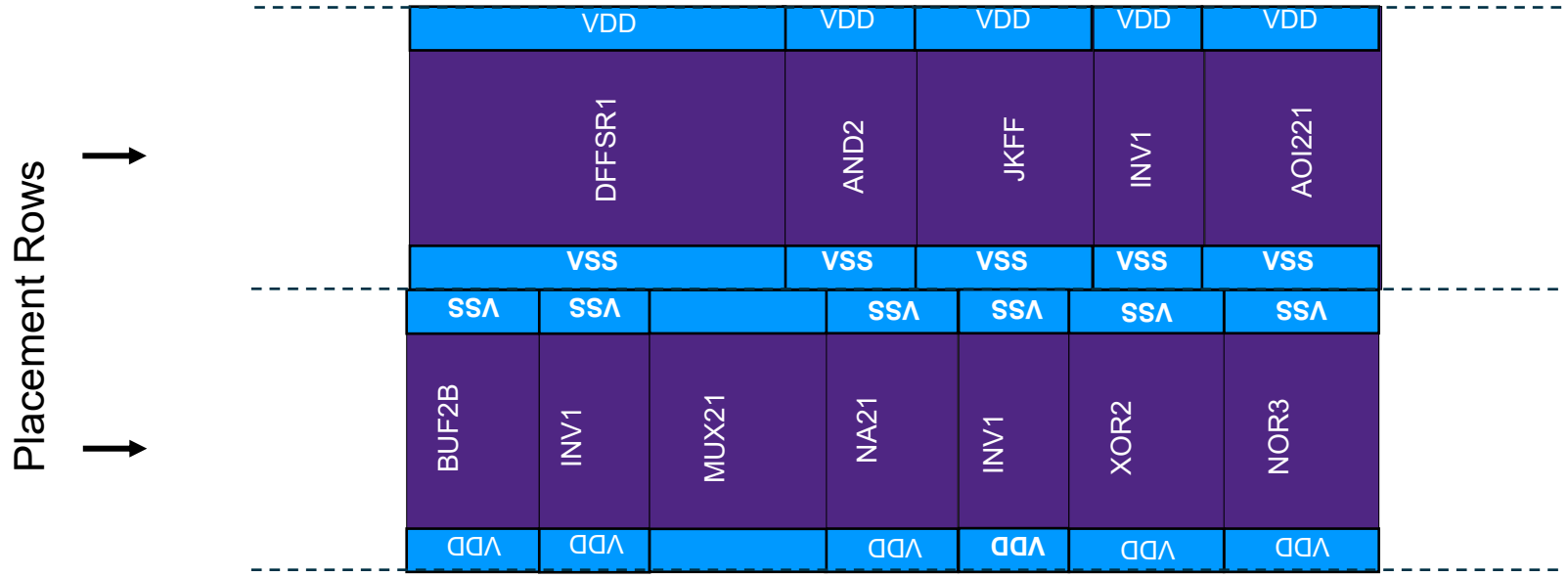
Physical Design: Site Rows and Tiles

- Placement uses grid in which cells are placed.
- Unit tile is used to build this grid.
  - Unit tile is defined by a library developer.
  - All the cells in the library are designed to be multiple of unit tile.

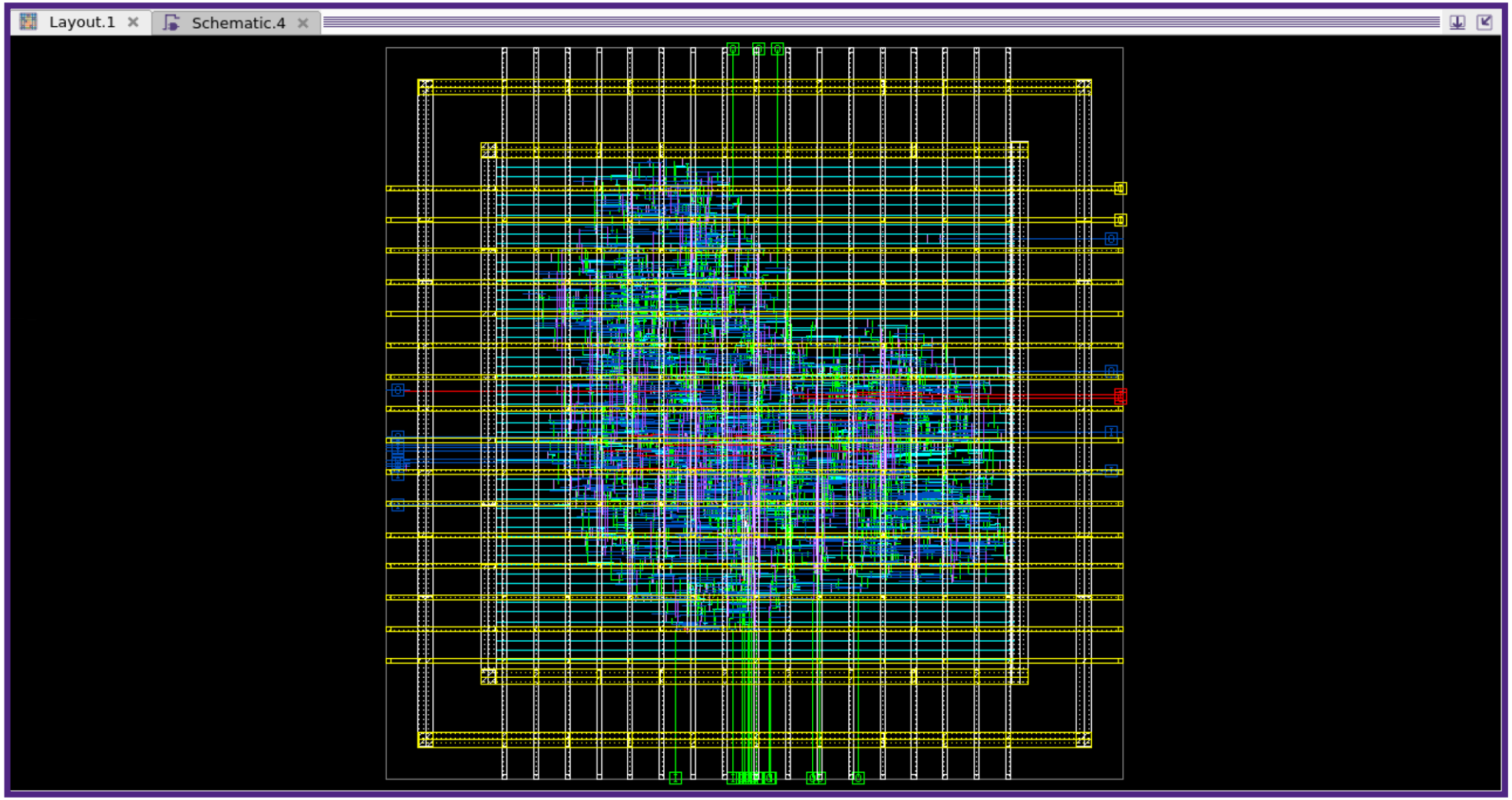


Physical Design: Standard cell rows

- Cells are placed in rows, next to each other.
- One cells structure continue previous one.
- Cells on neighbor rows are flipped so that they can share same supply.



Physical Design: Power Plan

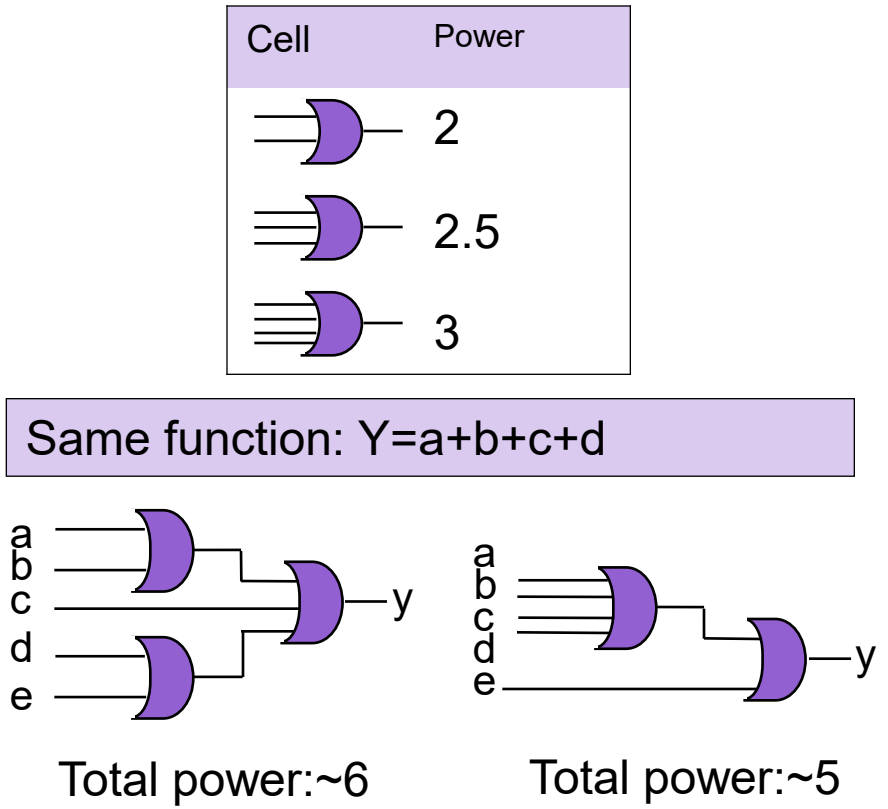


- Physical design consists of placed physical views of all standard cells, IP blocks, etc. and corresponding metal routing connections.

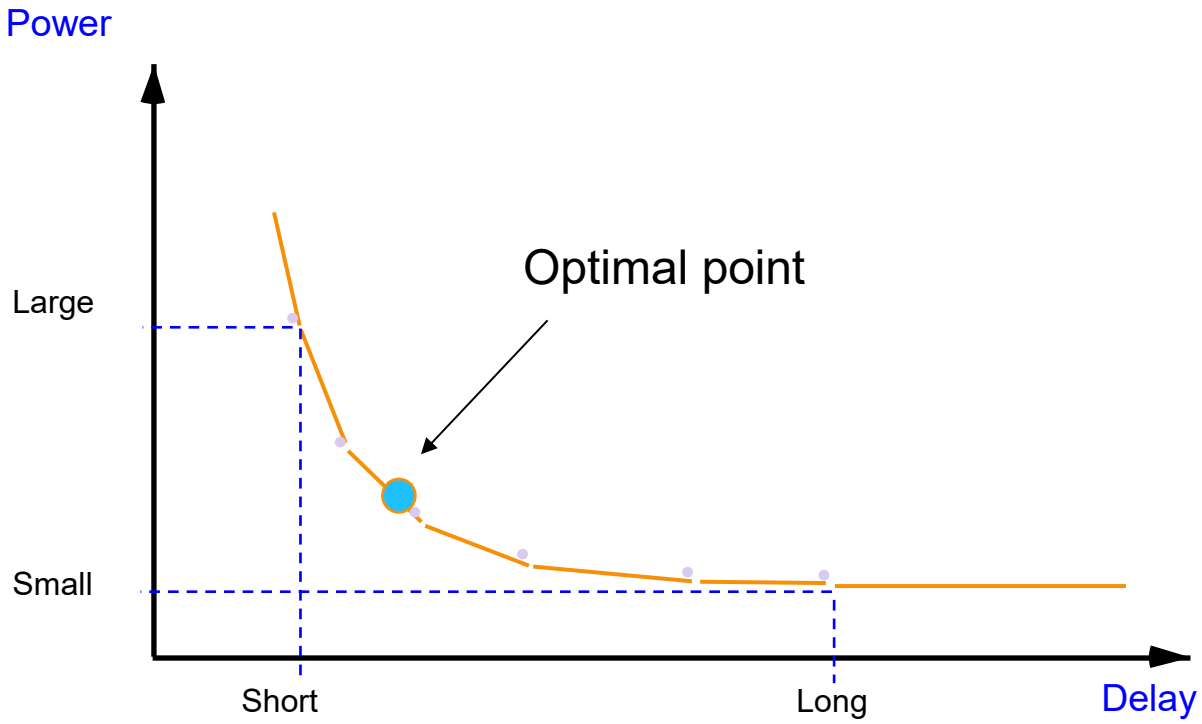
Main Optimization Trade-Offs

Circuit design is a trade-off of timing, power and area

- Timing optimization
  - Goal: small delays
- Power optimization
  - Goal: low power consumption
- Area optimization
  - Goal: small area



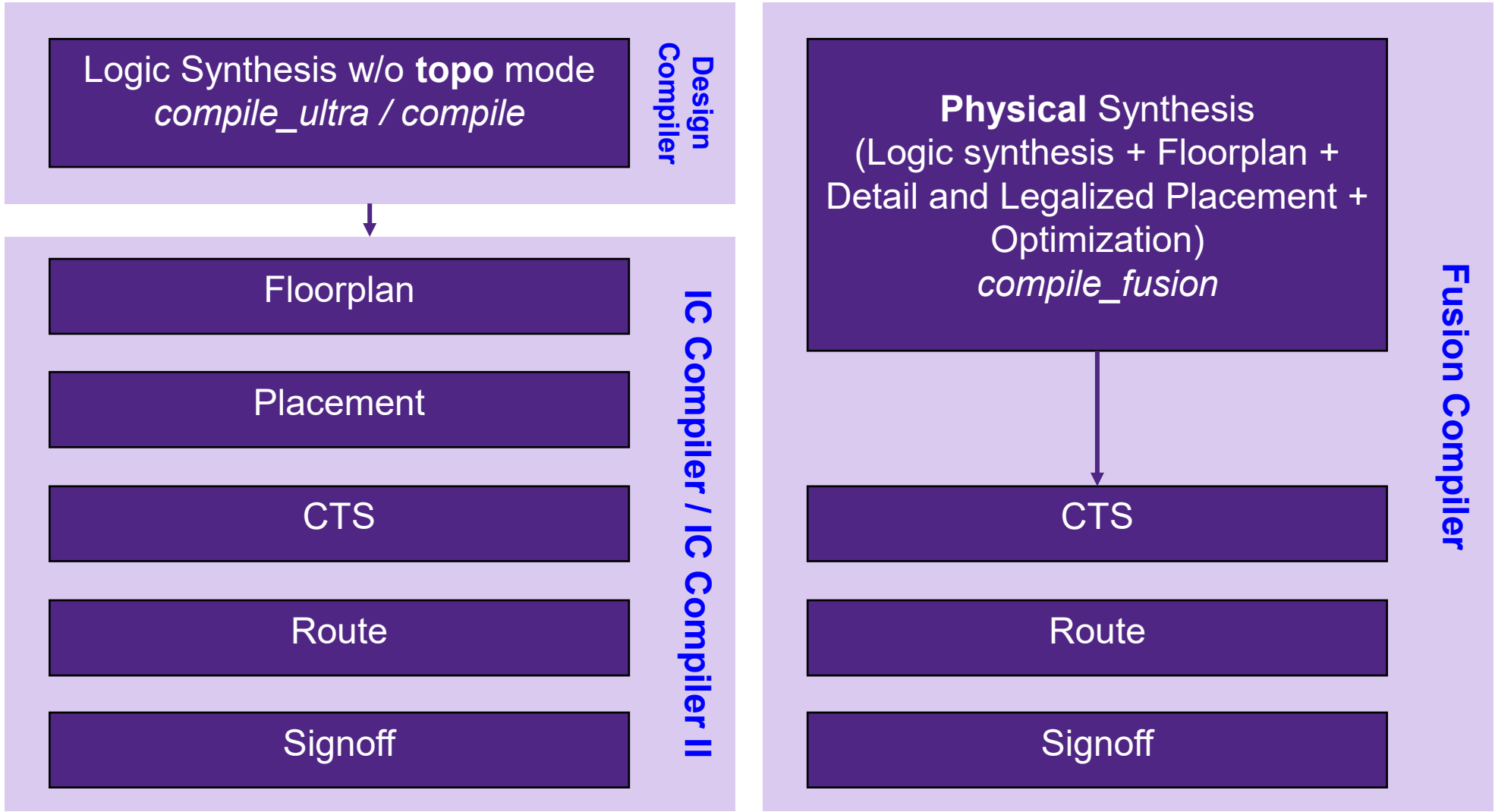
Design Constraints: Parameter Trade-off



Frequency is usually replaced by delay, as the lower the delay the larger is frequency

Power < 100 mW
Area < 2 um^2
Delay < 1ns

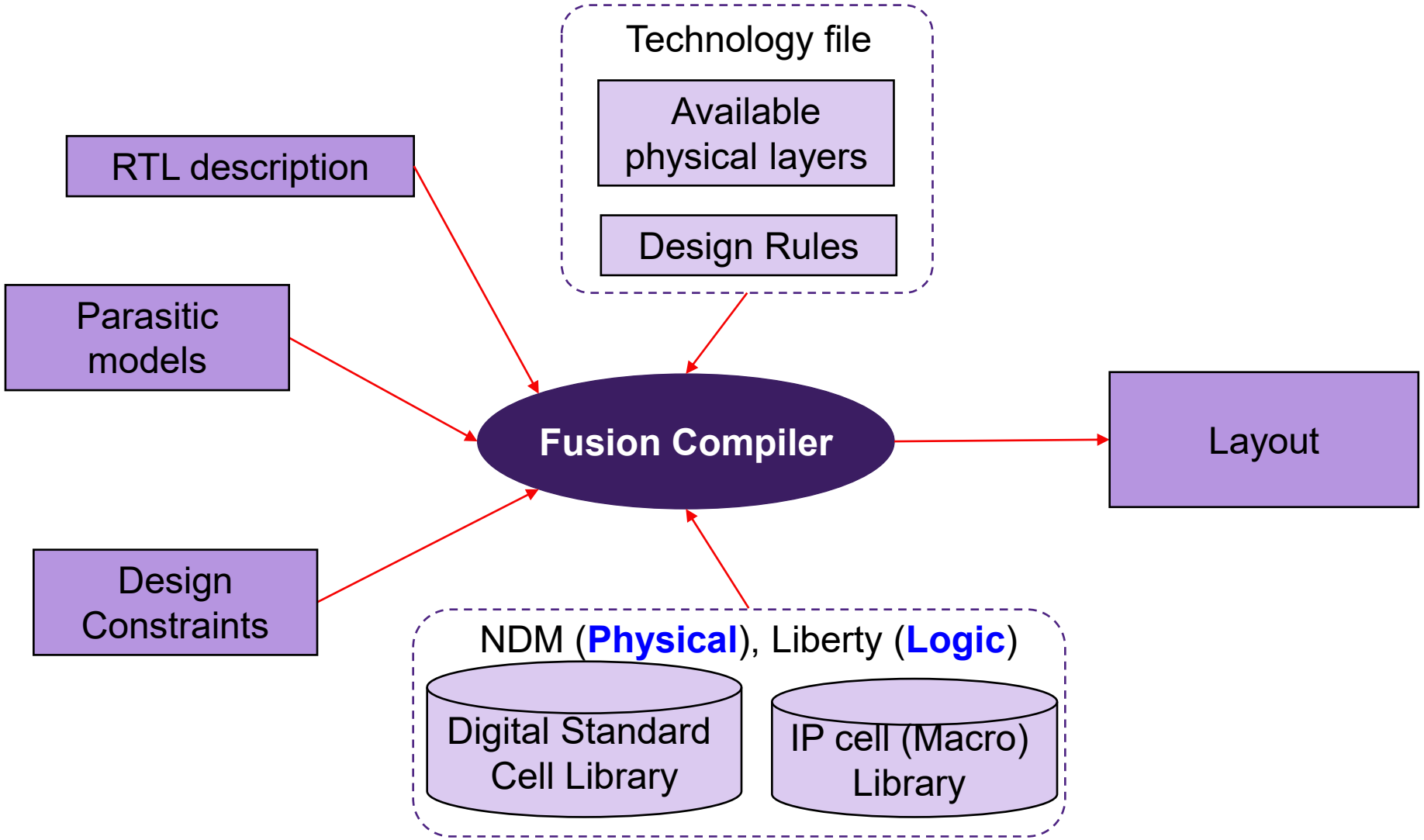
Design Flow with Fusion Compiler



Agenda

- IC Design Flow and Implementation Concepts
- ***Fusion Compiler Input Data***
- Starting Fusion Compiler
- Graphical User Interface

Main Inputs



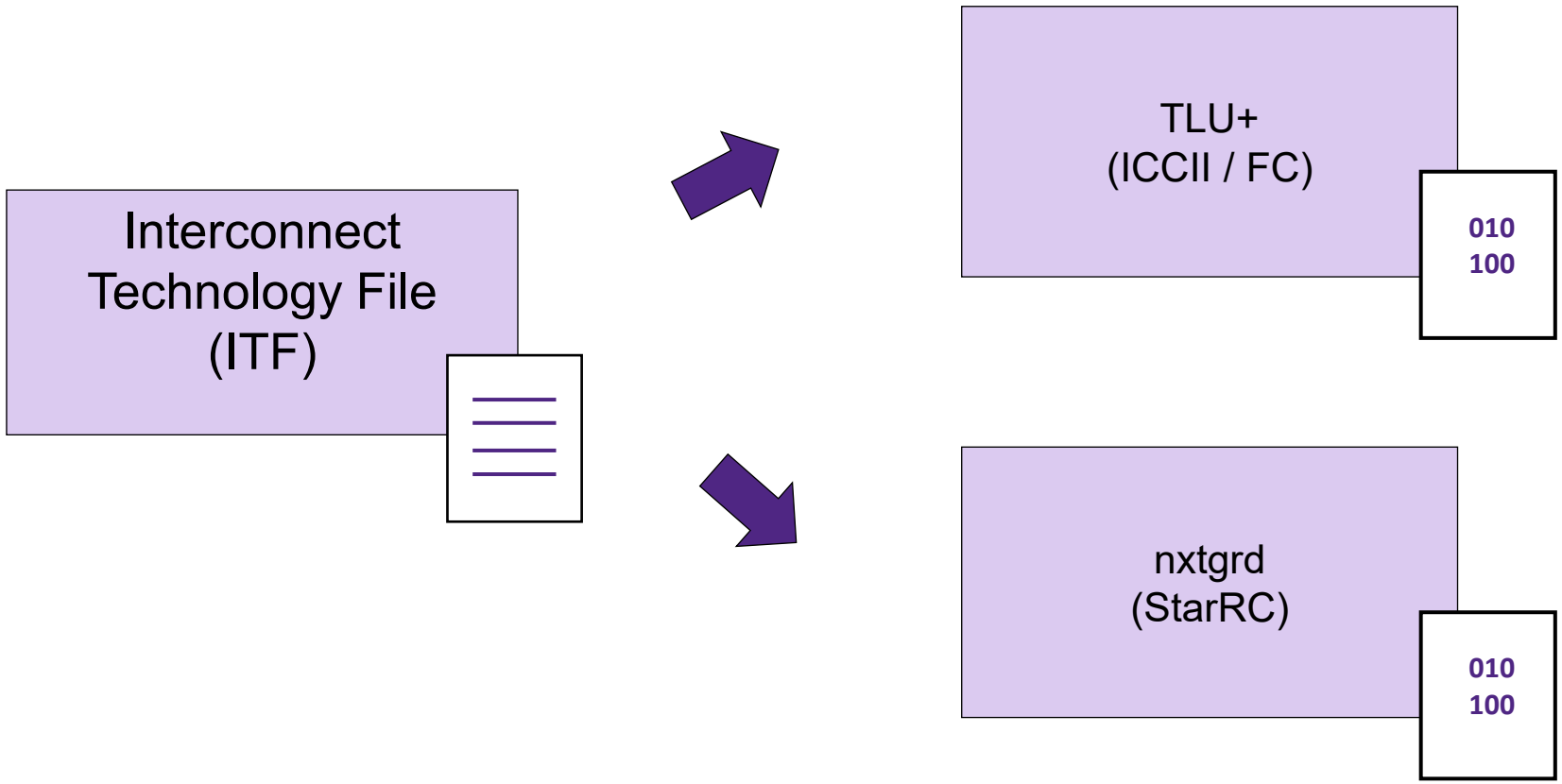
Technology File (.tf)

- Units and precision
- Layers
- Vias
- Design rules
  - For vias and layers
- Colors and patterns

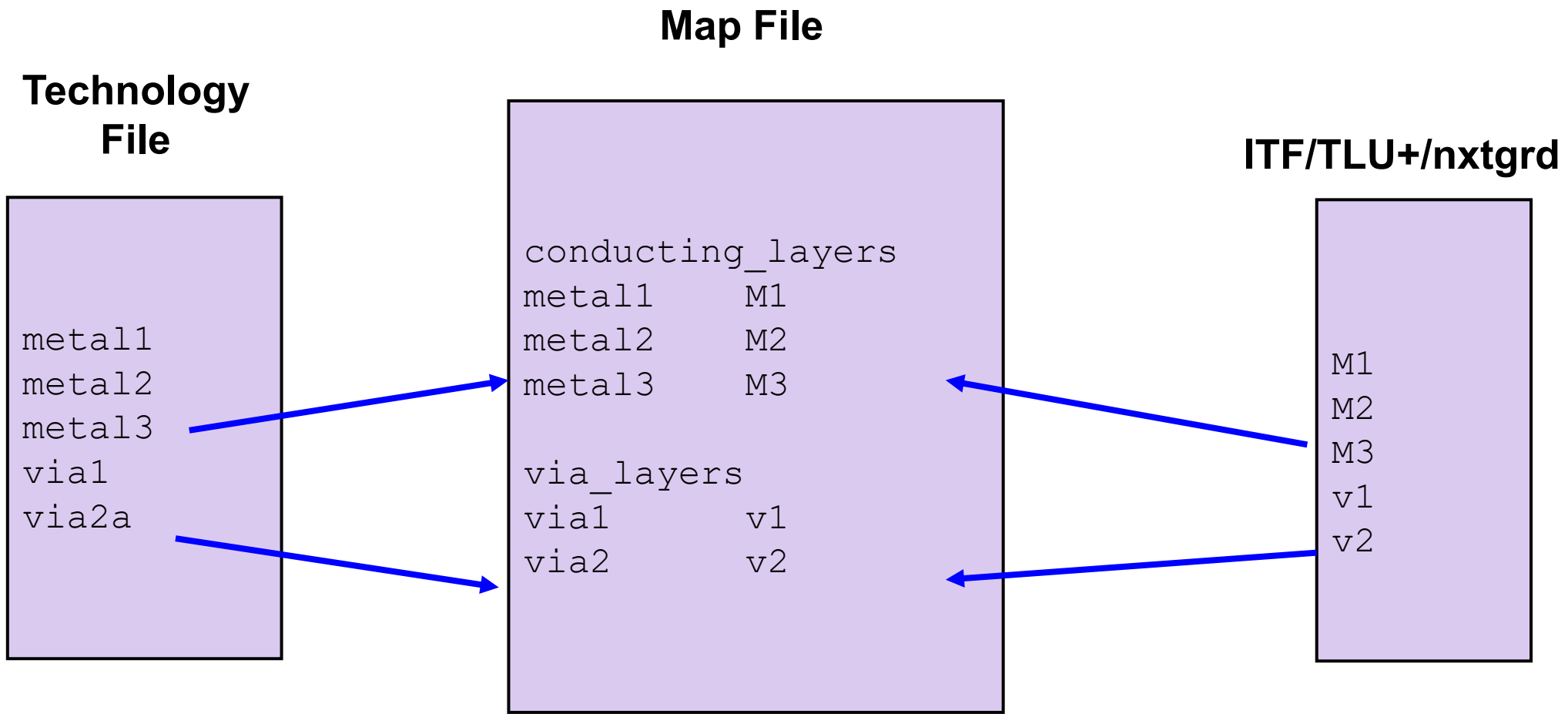
```
Technology  {
   unitTimeName = "ns"
   timePrecision = 1000
   unitLengthName = "micron"
     ...
}
Layer  "M1" {
    layerNumber = 16
    defaultWidth = 0.23
    minWidth = 0.23
     ...
}
ContactCode     "VIA_1_2" {
    contactCodeNumber  = 31
    cutLayer = "VIA1"
    lowerLayer  = "M1"
    upperLayer  = "M2"
     ...
}
```

Modeling Parasitics

- TLUPlus Models contain R/C look-up tables

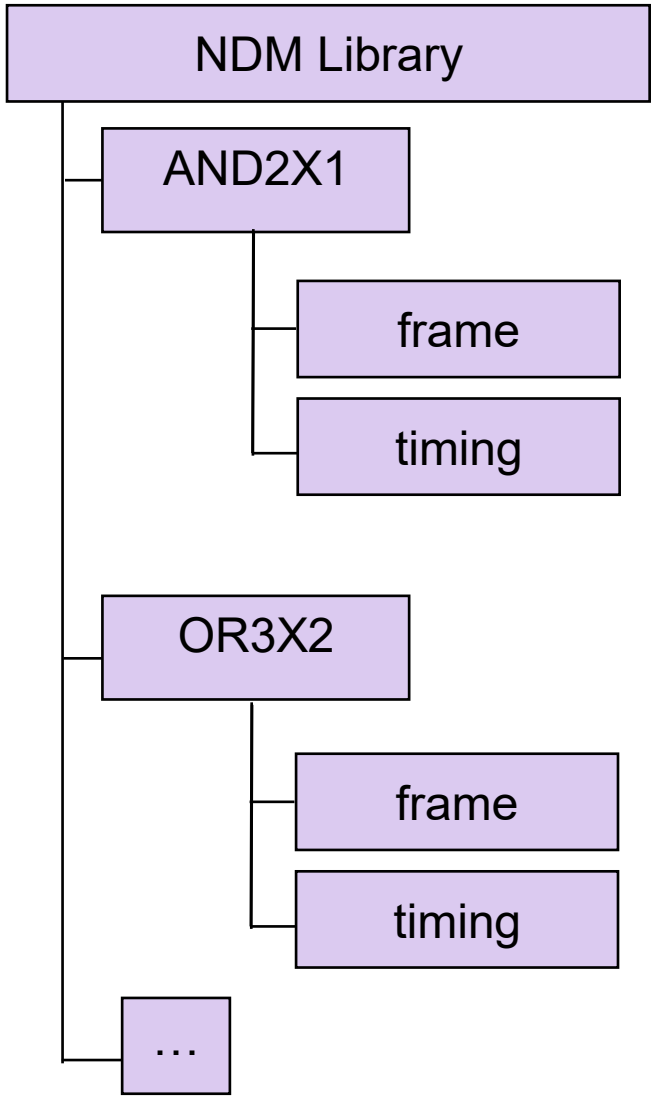


Map Files

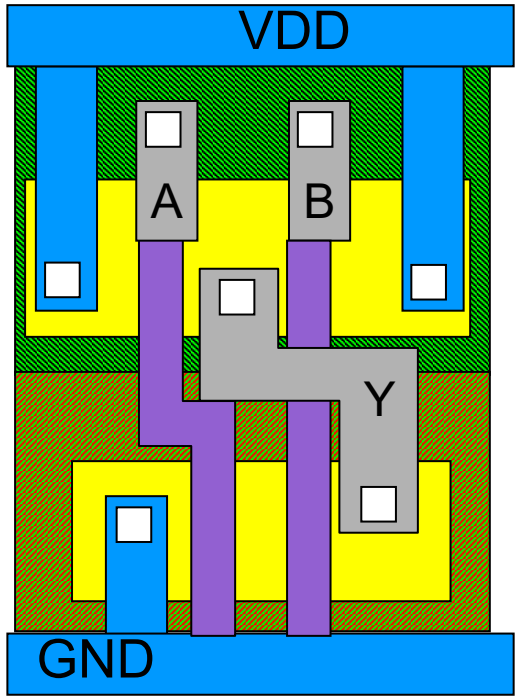


NDM Reference Library

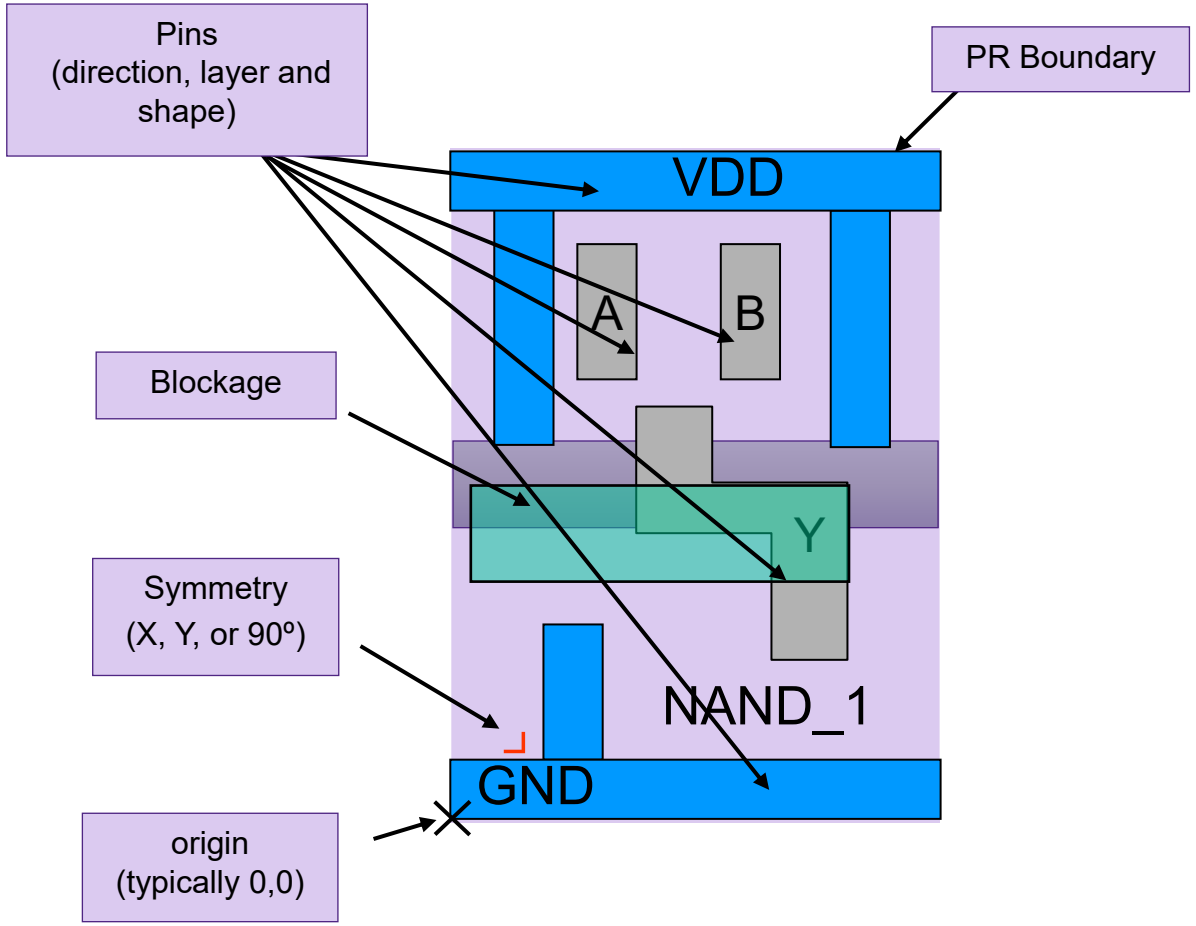
- Information is stored in “views”:
  - **frame view**: The abstract view used for place and route
  - **layout view**: The view which is equal to the cell’s layout
  - **timing view**: The abstract view which contains timing/power information
- Tools using NDM:
  - IC Compiler II
  - Fusion Compiler



Frame View Content



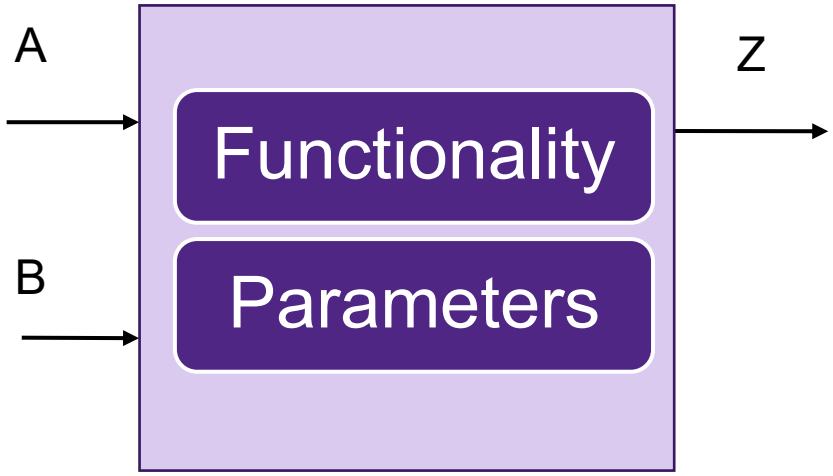
Layout View



Frame view

Cell Logic Model

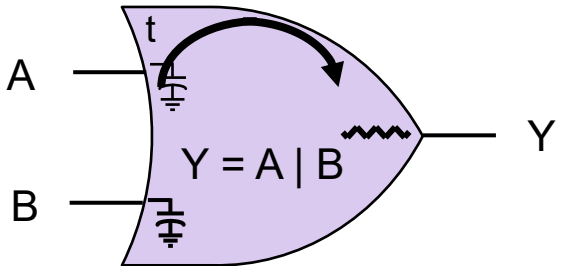
- Cell logic model is generated by characterization process
- Cell model contains:
  - Cell name, pins, pin directions
  - Functionality
  - Timing parameters
  - Power parameters
  - Other parameters if needed by EDA tool
    - Pin capacitances
    - …

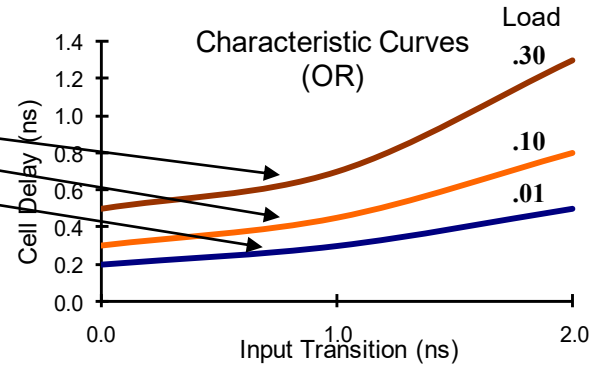


Cell Logic Model: Liberty

Liberty (*.lib) format

```
cell ( OR2_4x ) {                          <-- Cell name
    area : 8.000 ;                         <-- Cell Area
    pin ( Y ) {
        direction : 2;                     <-- 2 = Output; 1 = Input
        timing ( ) {
            related_pin : "A" ;
            timing_sense : positive_unate ;
            rise_propagation (drive_3_table_1) {
                values ("0.2616, 0.2711, 0.2831,..)
            }
            rise_transition (drive_3_table_2) {
              values ("0.0223, 0.0254, ...)
         . . . .
        function : "(A | B)";              <-- Pin Y Functionality
        max_capacitance :  1.14810 ;
        min_capacitance :  0.00220 ;       <-- Design Rules for Pin Y
    }
    pin ( A ) {
        direction : 1;
        capacitance : 0.012000;            <-- Electrical Characteristics of Pin A
         . . . .
```

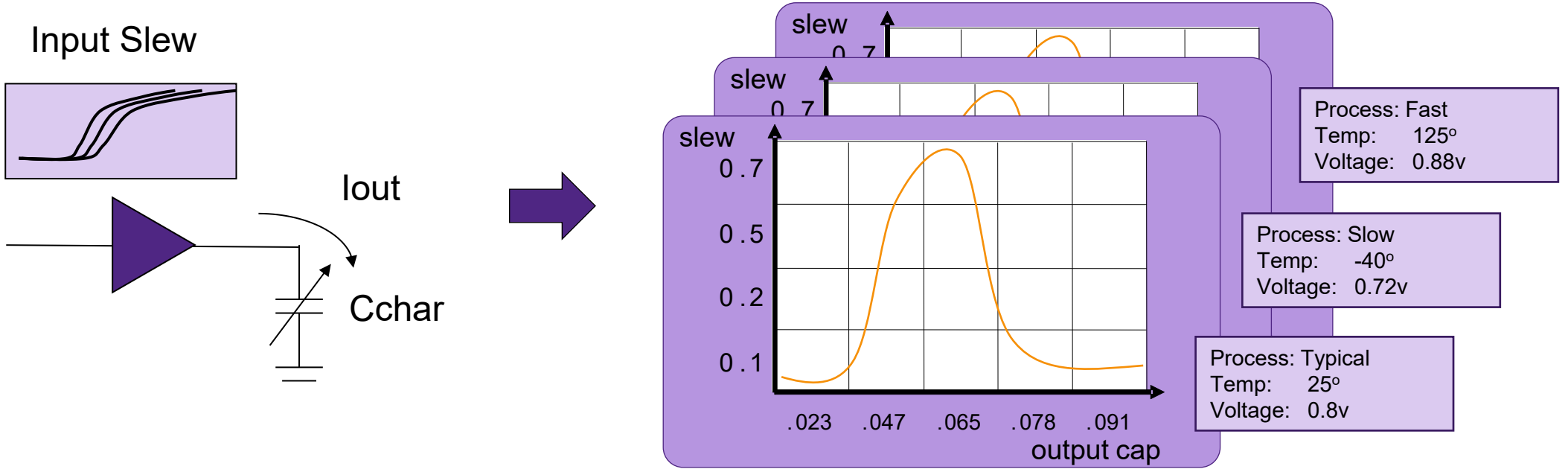




* Each liberty (.lib) has corresponding (.db) file

Characterization

- Characterization computes cell parameter (e.g. delay, output current) depending on input variables: output load, input slew, etc.
- Characterization is performed for various combinations of operating conditions: process, voltage, temperature (also called PVT corners).



Characterization Corners

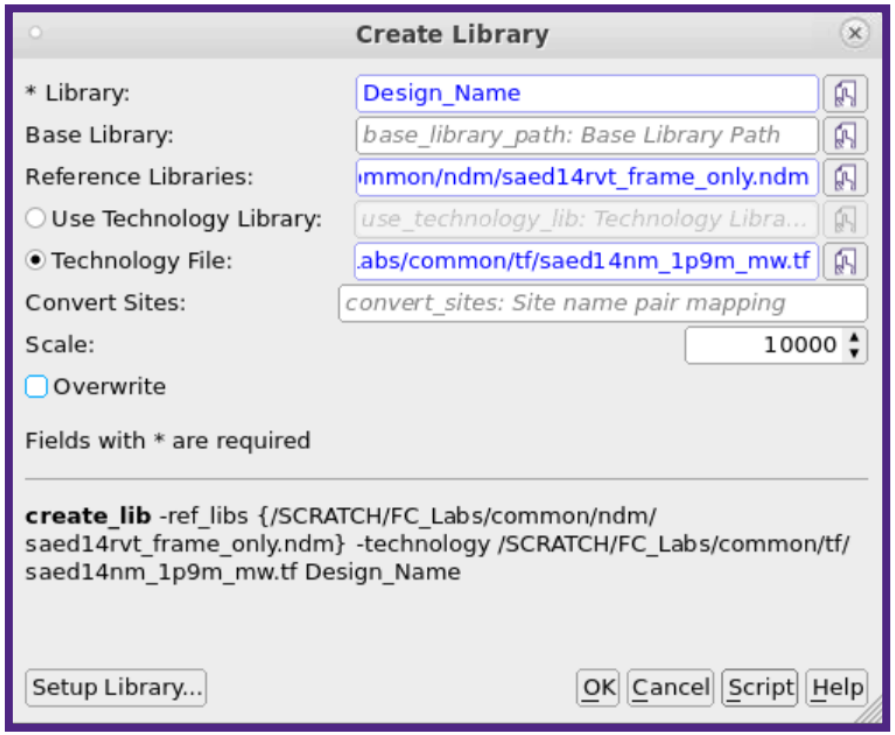
Corner Name	Process (NMOS proc. – PMOS proc.)	Power Supply (V)	Temperature (C)
tt0p8v25c	Typical - Typical	0.8	25
tt0p8v125c	Typical - Typical	0.8	125
tt0p8vm40c	Typical - Typical	0.8	-40
ss0p72v25c	Slow - Slow	0.72	25
ss0p72v125c	Slow - Slow	0.72	125
ss0p72vm40c	Slow - Slow	0.72	-40

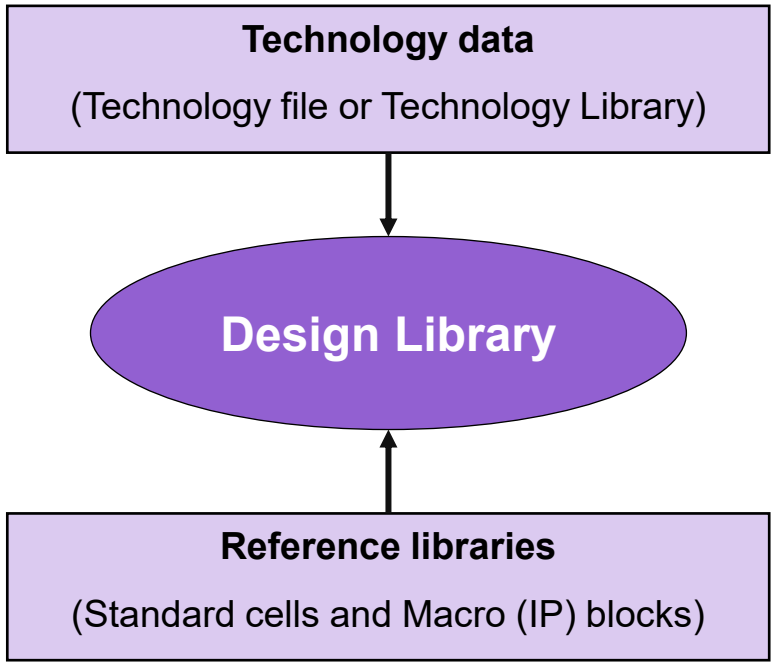
Agenda

- IC Design Flow and Implementation Concepts
- Fusion Compiler Input Data
- ***Starting Fusion Compiler***
- Graphical User Interface

Create Design Library (1)

File -> Create Library





- **create_lib** ${DESIGN_LIBRARY} -technology $TECH_FILE \
  -ref_libs ${REFERENCE_LIBRARY}

- **create_lib** ${DESIGN_LIBRARY} -use_technology_lib $TECH_NDM \
  -ref_libs ${REFERENCE_LIBRARY_WITH_TECH_LIB}

Create Design Library (2)

There are 2 possible scenarios for providing reference libraries:

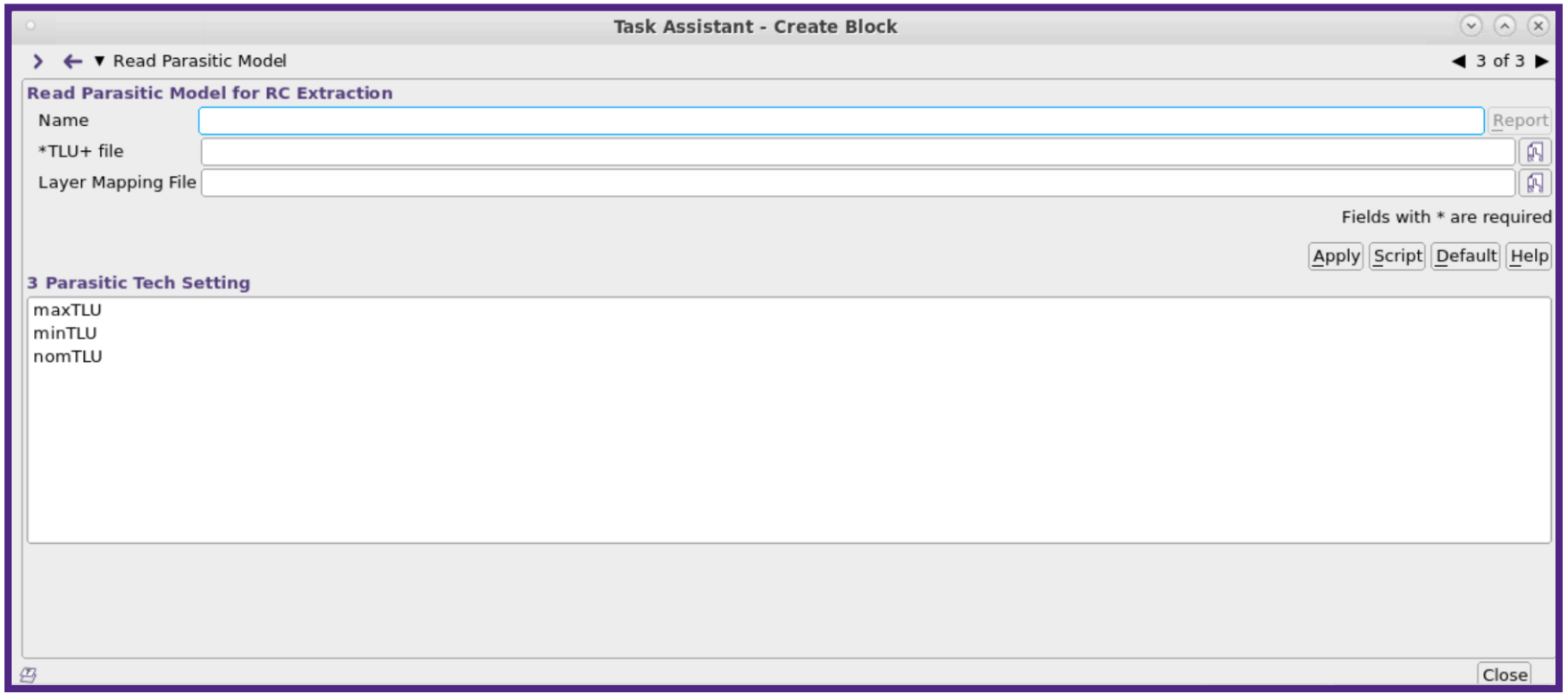
1. Provide **frame_only NDM + Liberty db** to generate **CLIBs.**
   - Library Manager tool is used to automatically generate CLIBs with shared logic and physical data.
   - In this scenario only Technology File and reference frame_only NDMs need to be provided in *create_lib* command.
   - Need to provide **Liberty db** before creating the design library:

```
set_app_var link_library "${DB_1} ${DB_2} ${DB_3} …"
```

2. Provide **Timing NDM** or already generated **CLIBs**.
   - No need in providing Liberty db.

Read Parasitic Models

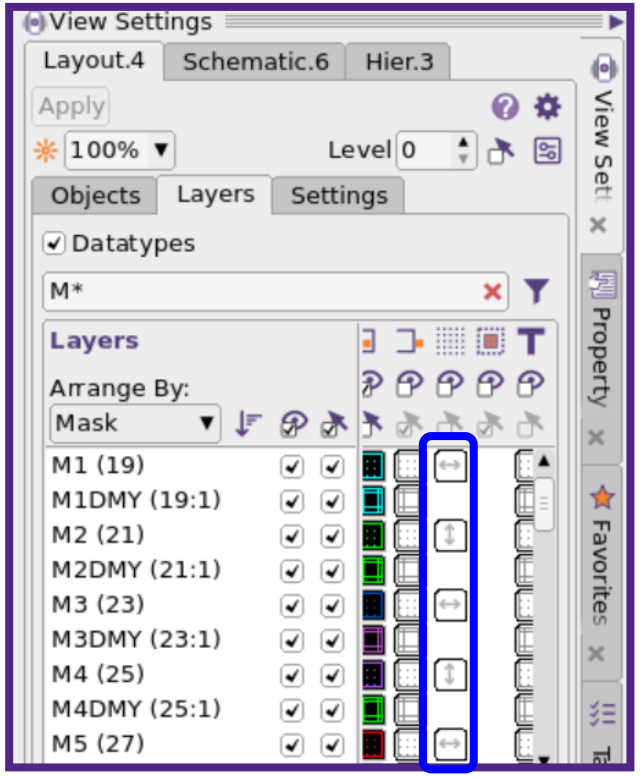
Task -> Create Block -> Read Parasitic Model




```
read_parasitic_tech -layermap ${LAYER_MAP_FILE} -tlup ${TLUP_MAX_FILE} -name maxTLU
read_parasitic_tech -layermap ${LAYER_MAP_FILE} -tlup ${TLUP_NOM_FILE} -name nomTLU
read_parasitic_tech -layermap ${LAYER_MAP_FILE} -tlup ${TLUP_MIN_FILE} -name minTLU
```

Setup Routing Directions

- Before starting the design implementation routing directions for available metal layers from Technology File must be defined.



- **set_attribute** [get_layers {${HORIZONTAL_LAYERS}}] routing_direction horizontal
- **set_attribute** [get_layers {${VERTICAL_LAYERS}}] routing_direction vertical

Read RTL

1. Analyze the RTL to check the syntax:

- **analyze** -format verilog RTL.v
- **analyze** -format sverilog RTL.sv
- **analyze** -format vhdl RTL.vhd

   Verilog, System Verilog and VHDL Hardware Description Languages are supported.

2. Elaborate the design to build technology independent circuit:

- **elaborate** ${DESIGN_NAME}

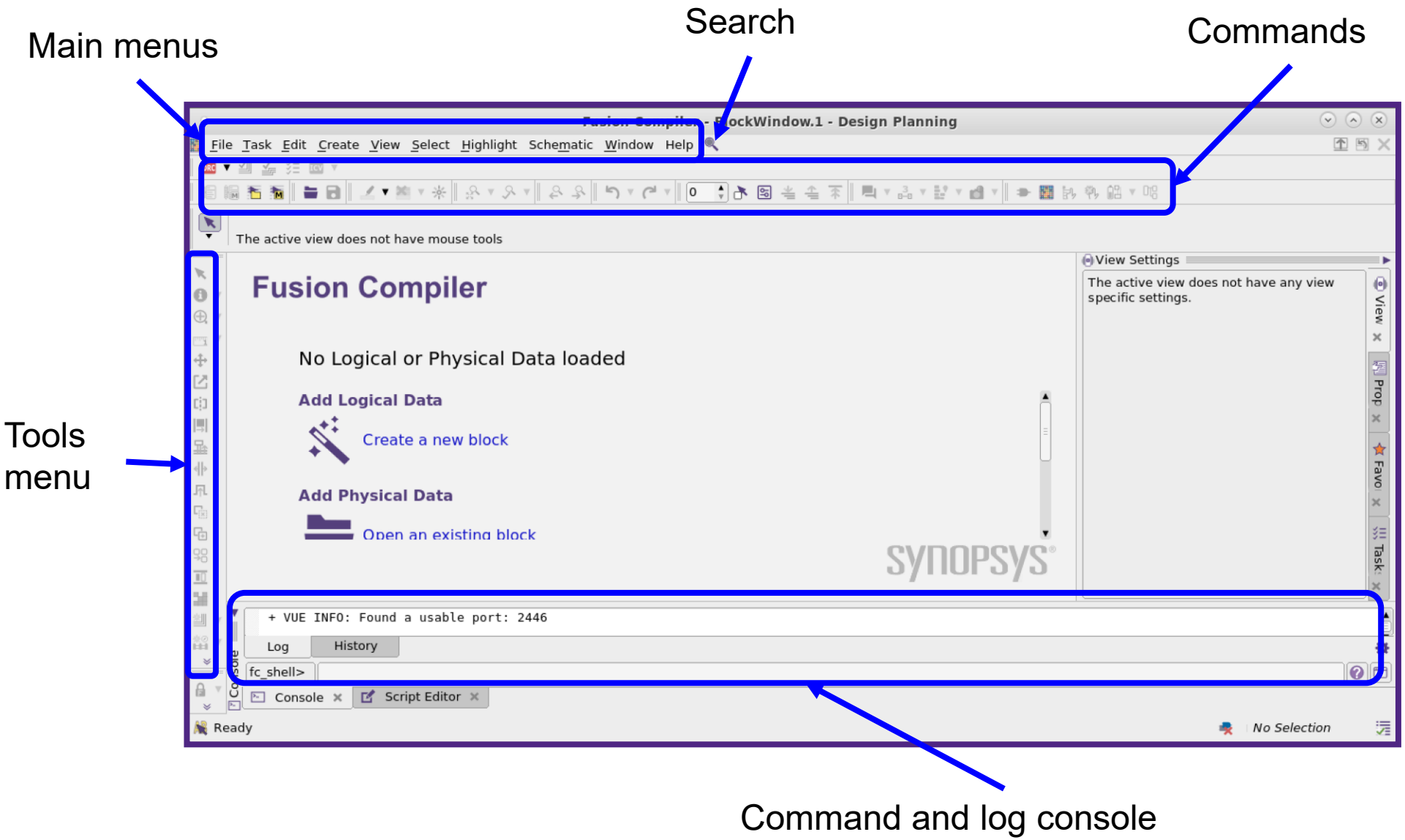
3. Set top module after which the design block in your design library will be created and design will be linked:

- **set_top_module** ${DESIGN_NAME}

Agenda

- IC Design Flow and Implementation Concepts
- Fusion Compiler Input Data
- ***Graphical User Interface***

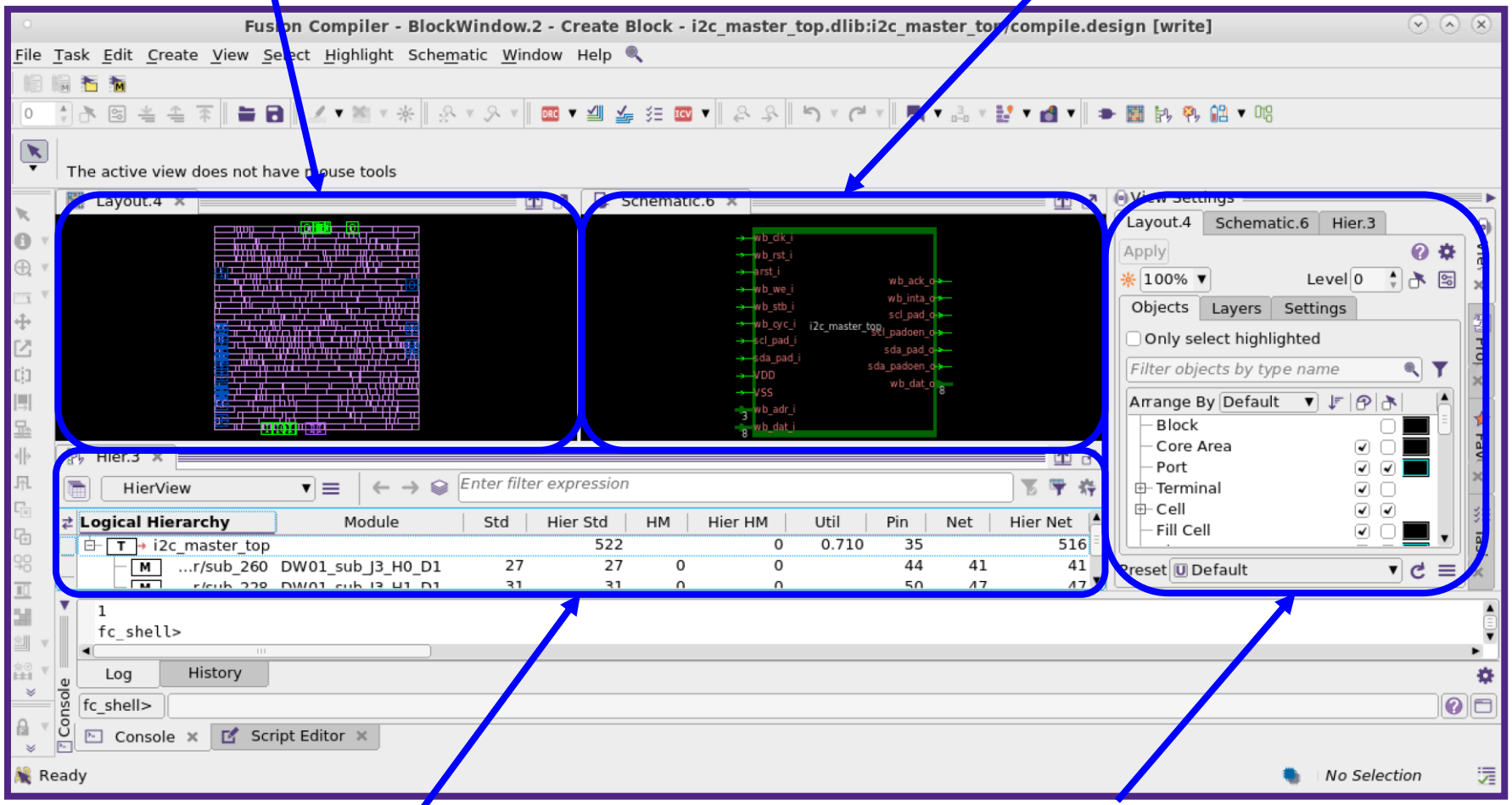
Main Window



Block Window

Layout view

Schematic view



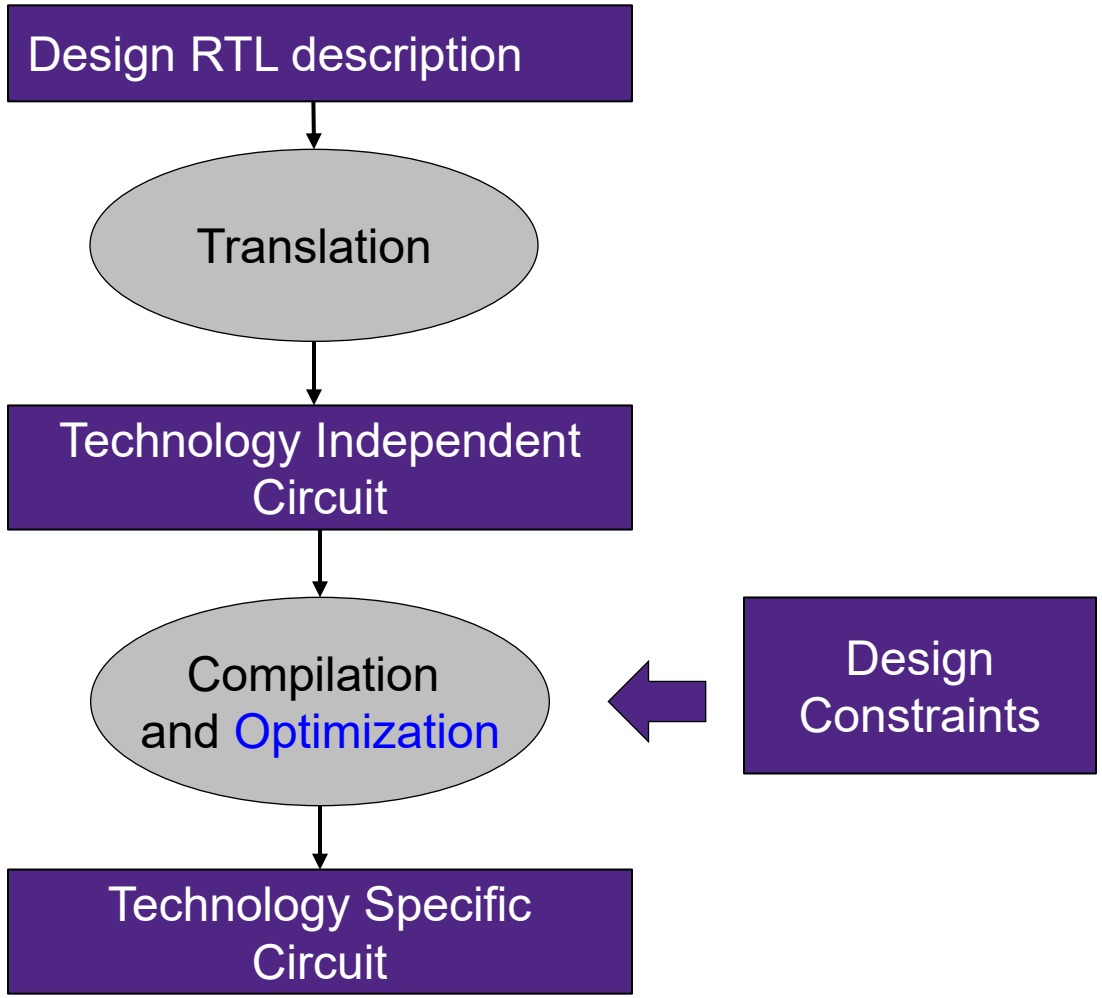
Hierarchy view

Visibility and object selection control

Course Overview

- Introduction
- Timing and Area Constraints
- Multi-Corner Multi-Mode and Mapping
- Application Options and Compile Flow
- Floorplan
- Placement
- Clock Tree Synthesis (not covered)
- Routing (not covered)
- Signoff (not covered)

Constraint-Driven Synthesis

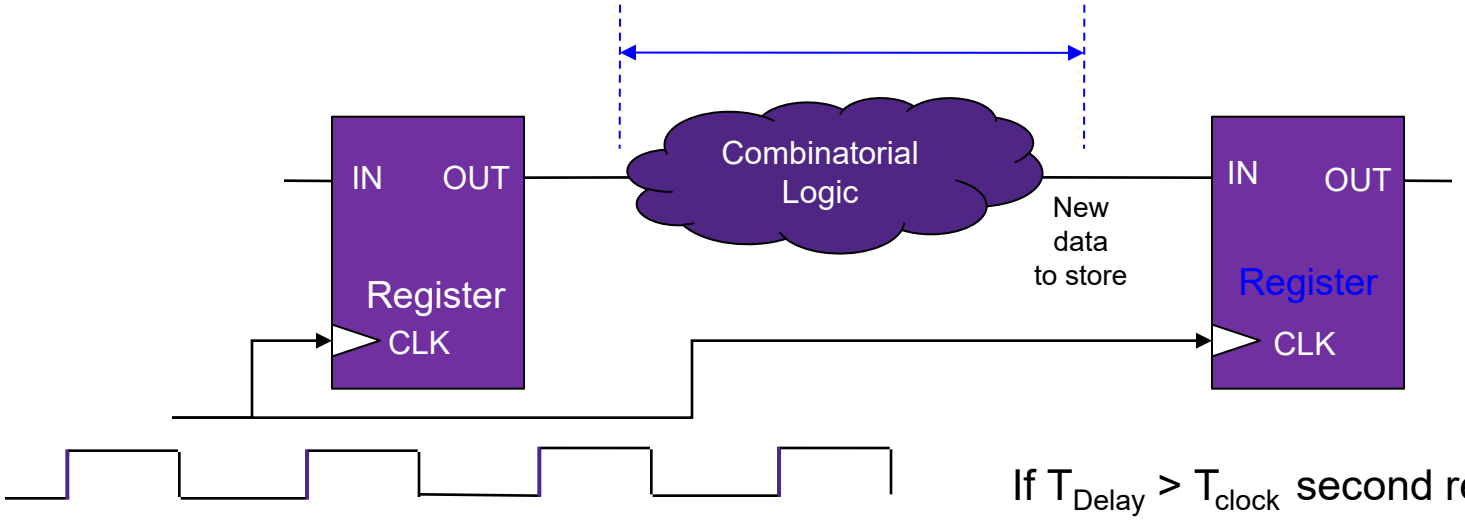


Area Constraints

- Area constraints in Fusion Compiler are given by limiting maximum area value;
- As timing has greater priority in Fusion Compiler it is used to set maximum area to zero, thus optimization achieves the best possible area with timings met;
- In Fusion Compiler area can be controlled also by floorplan parameters during Physical Synthesis.

Timing Closure and Constraints

- Problem
  - In clocked environment signals on the register inputs must arrive before next reading sequence (called arrival time)

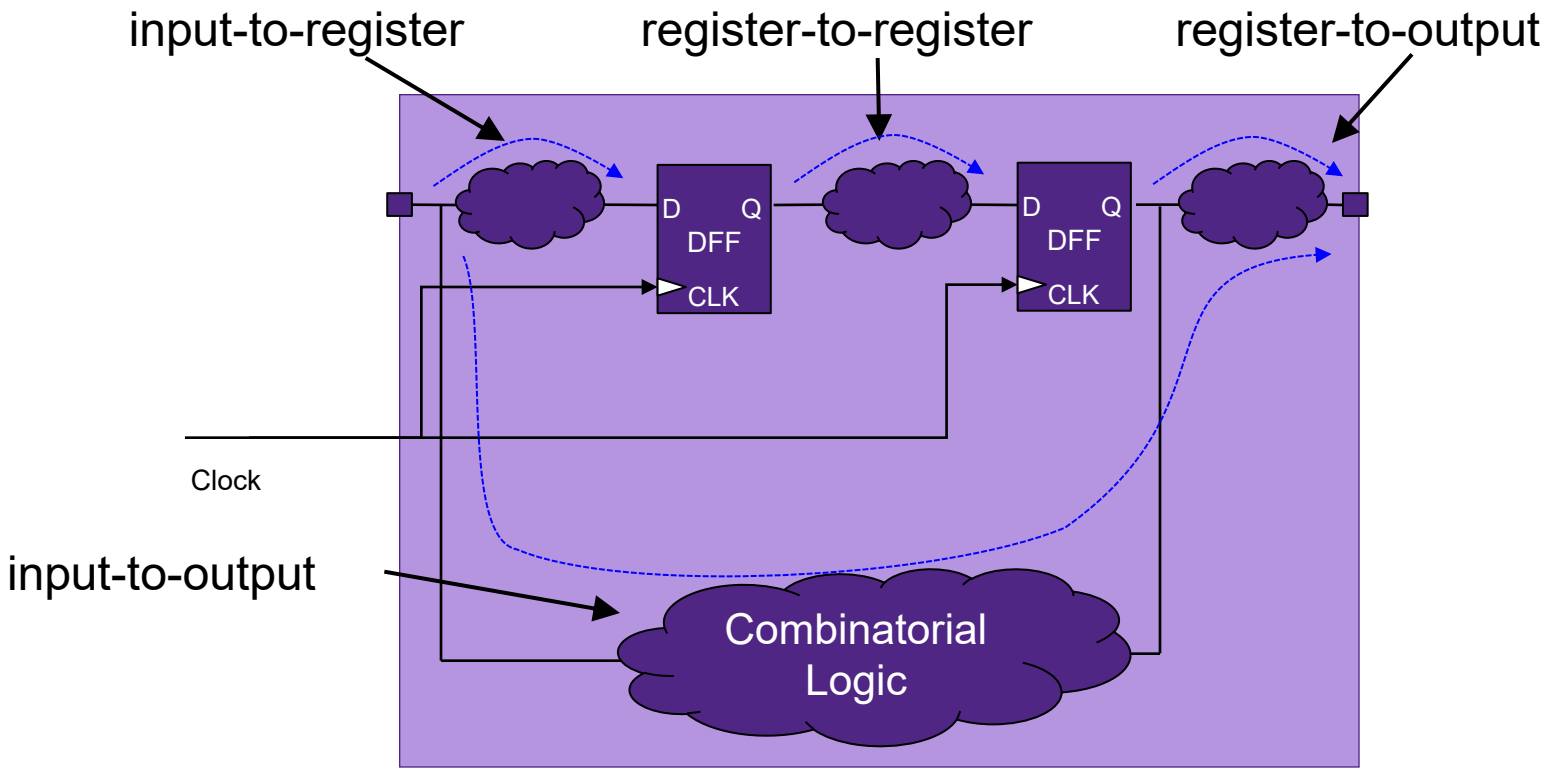


Clock: Both registers store new value at specific points of time (with some frequency, like 1GHz)

If $T_{Delay} > T_{clock}$ second register will not be able to store new data and it will be lost!

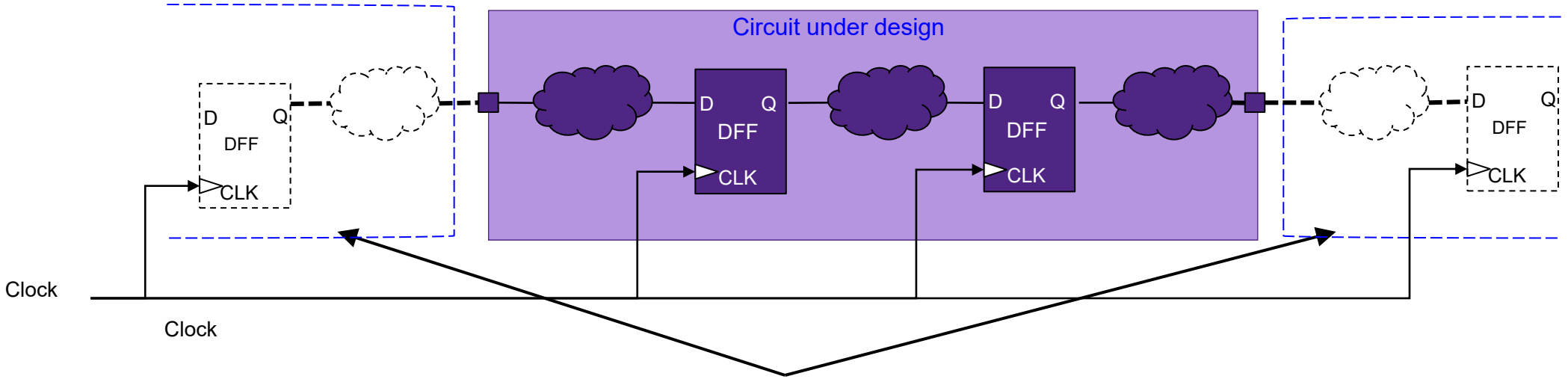
Path Types

- There are 4 types of paths in a synchronous circuit



Clocked Environment

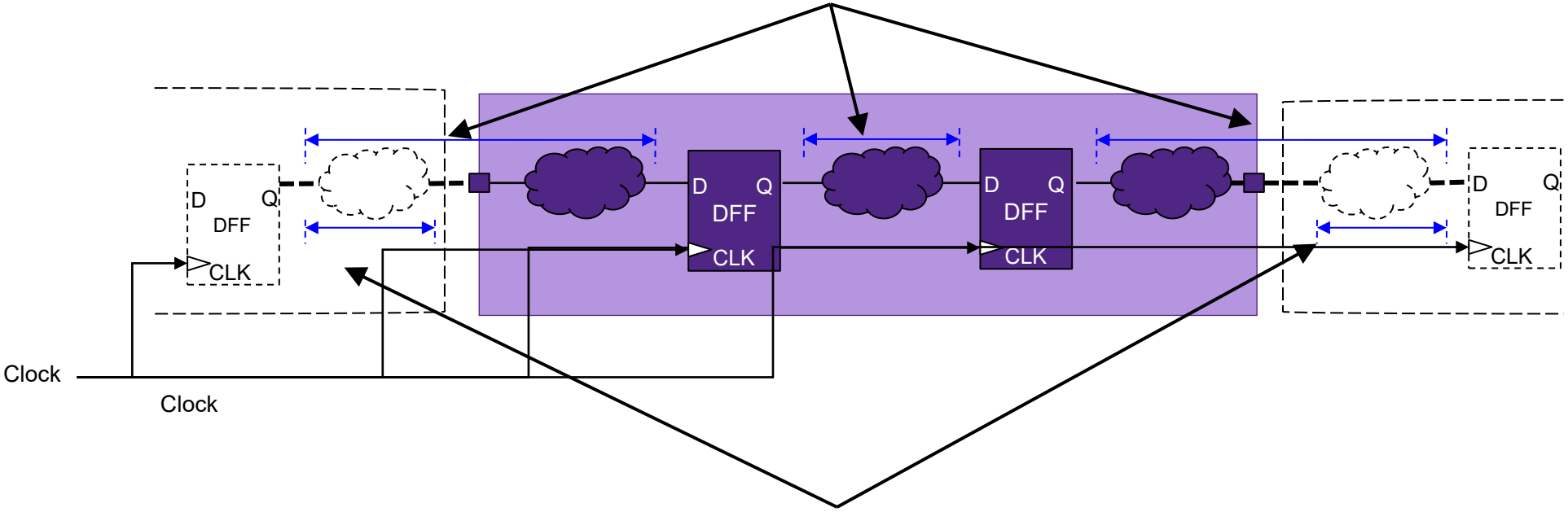
- For the ease of setting timing constraints the design is considered inside an environment clocked by the same clock signal.



- Virtual outer logic is considered during constraining to enable uniform constraints for all parts of a circuit.

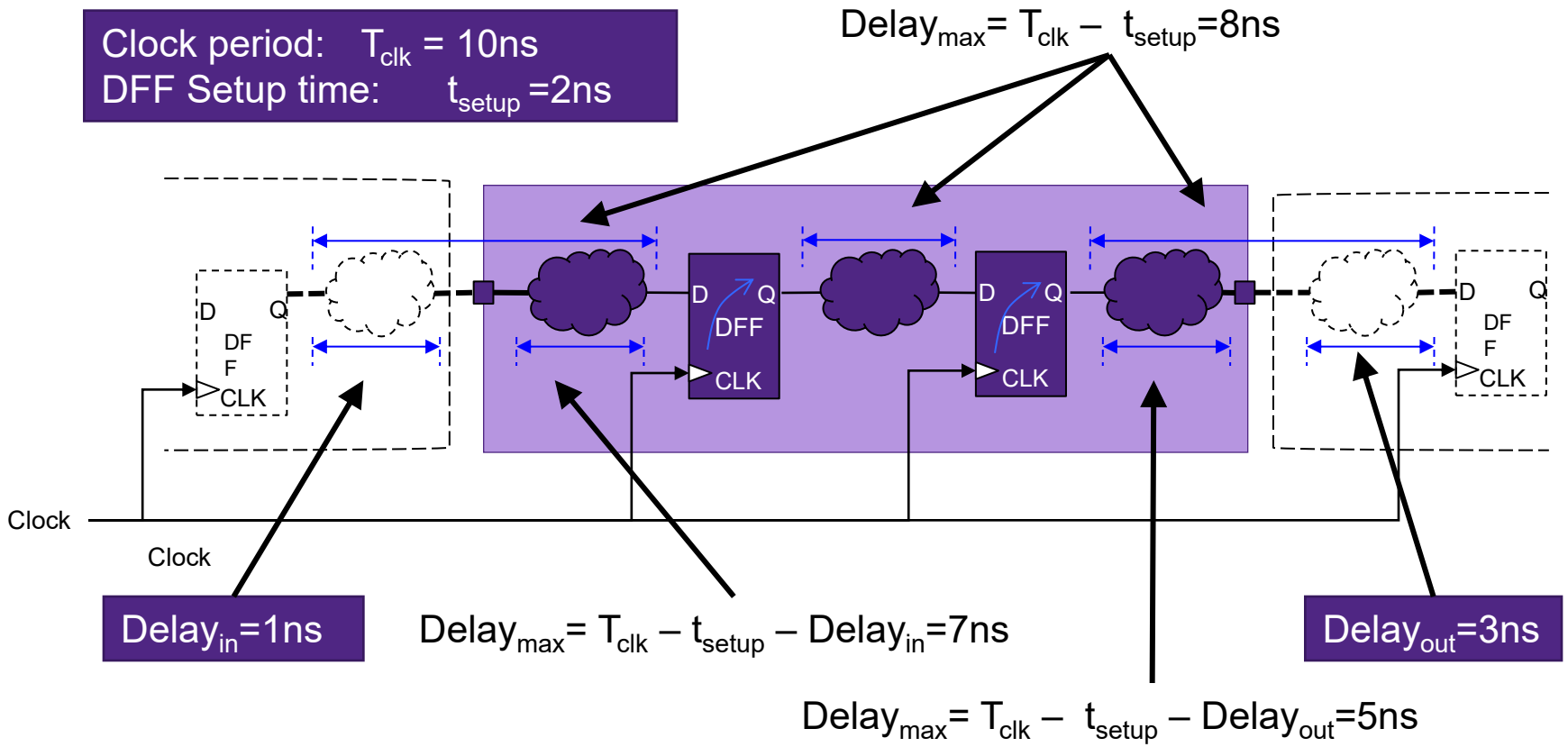
Constraining Timing: Setup/Hold

- All inter-register logic delays are constrained by clock period and register setup/hold requirements



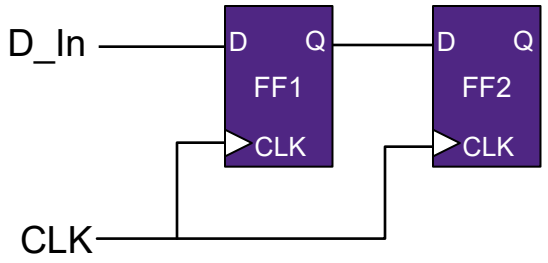
- To calculate delays of logic at the input/output ports the delays of outer virtual logic need to be provided.

Constraining Timing: Example

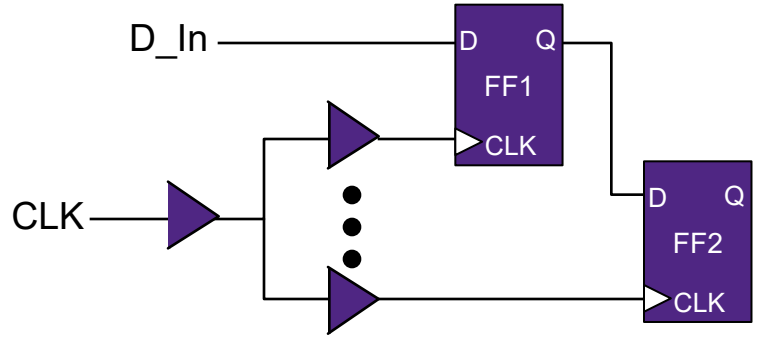


Constraining Timing: Modeling Clock

- In Synthesis , clock is treated as ideal and Clock tree is built at physical synthesis;
- To get realistic design clock skew effect needs to be modeled in logic synthesis to avoid optimistic design;
- With Fusion Compiler there is a possibility to mix Logic and Physical Synthesis.

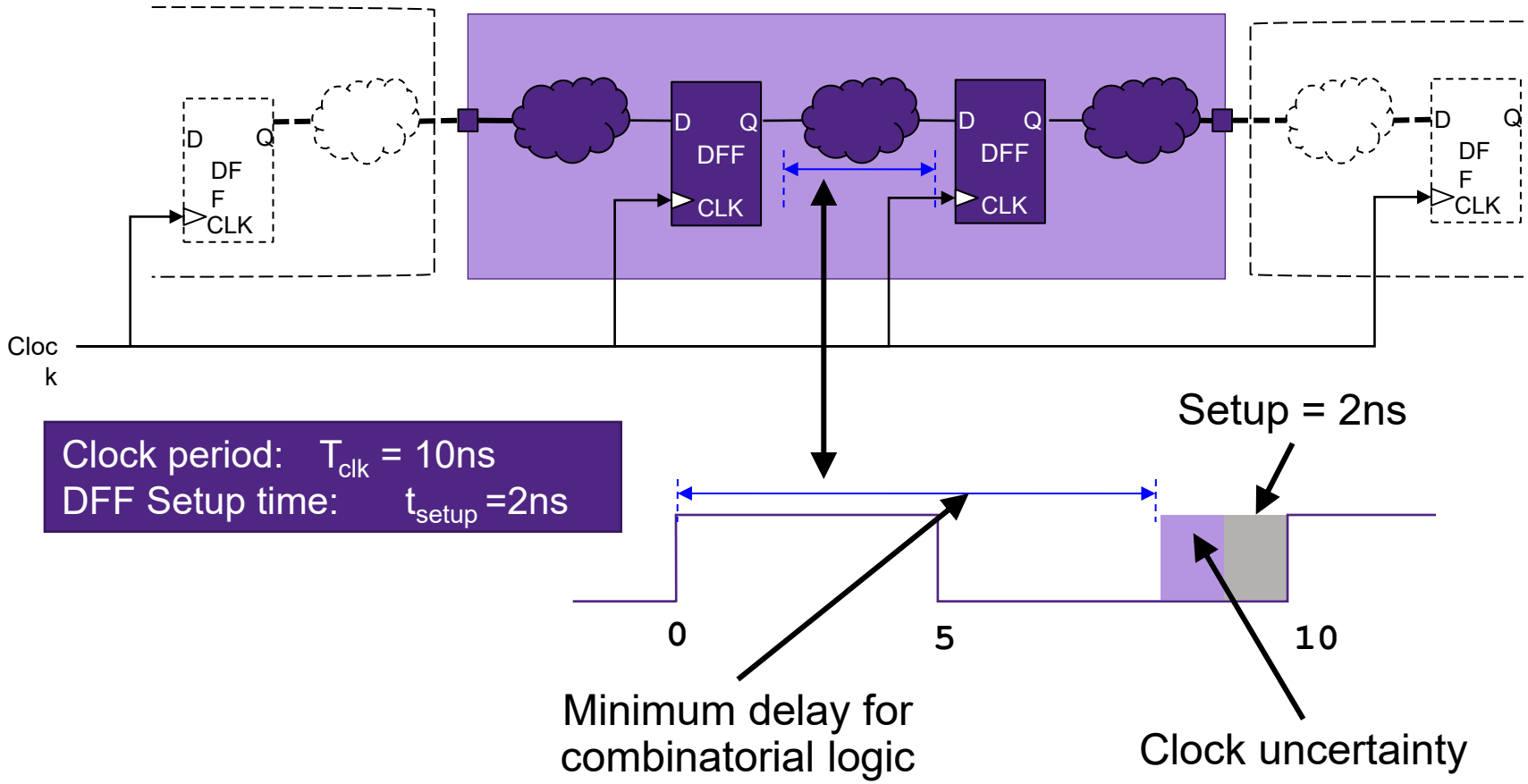


Logic Circuit



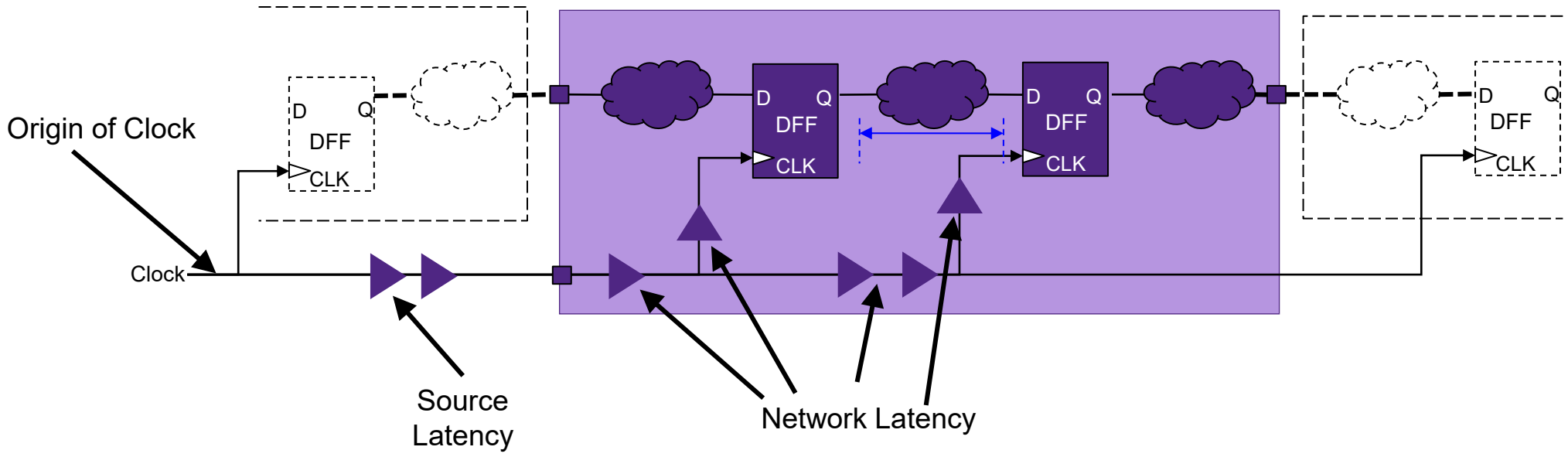
Post-Layout Circuit

Clock Uncertainty Modeling Example

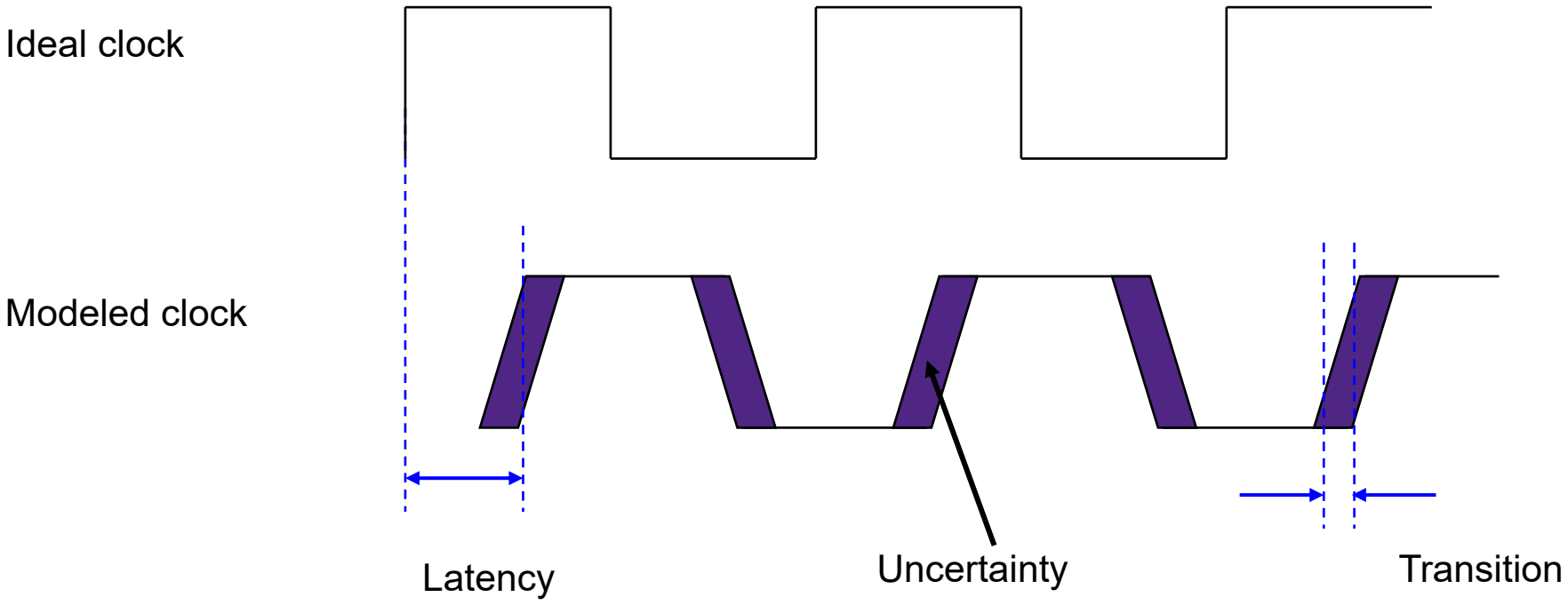


Modeling Clock Latency

- Clock latency can be of two types:
  - Source latency: the delay of clock signal from its source to designs input port (useful when clock generation circuitry is not part of the design)
  - Network latency: average propagation delay of clock signal inside the design itself



Clock Modeling Summary

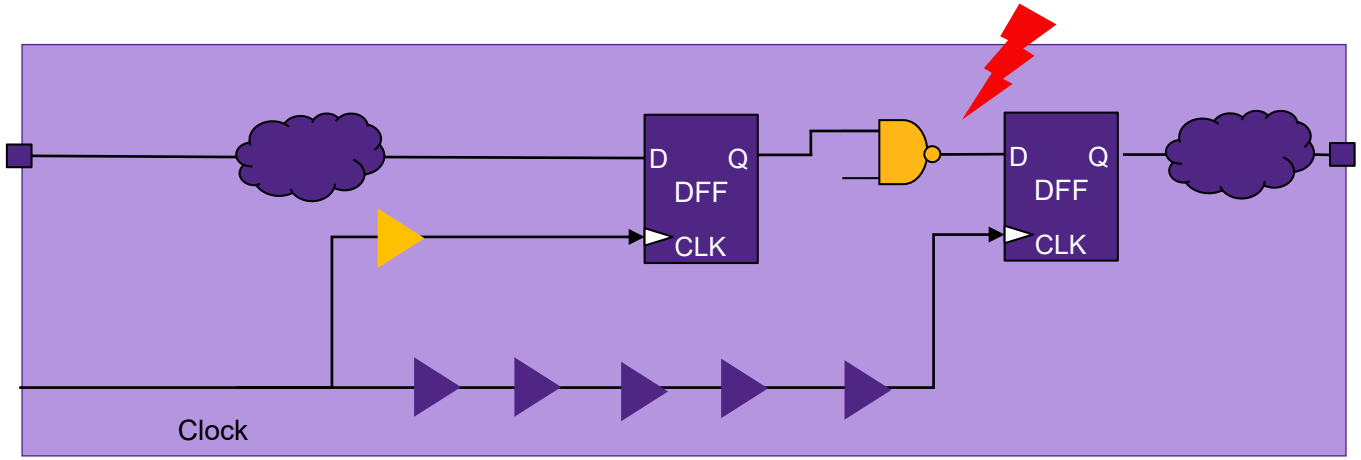


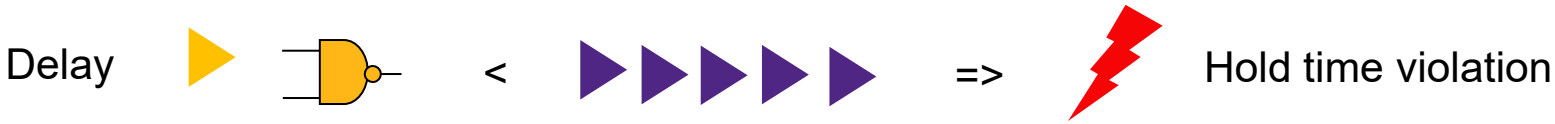
Uncertainty = jitter + skew + margin

Hold time

- Due to clock uncertainty, the clock signal reaches flipflops at different times
- For each register, hold time requirements need to be met to avoid that data gets lost

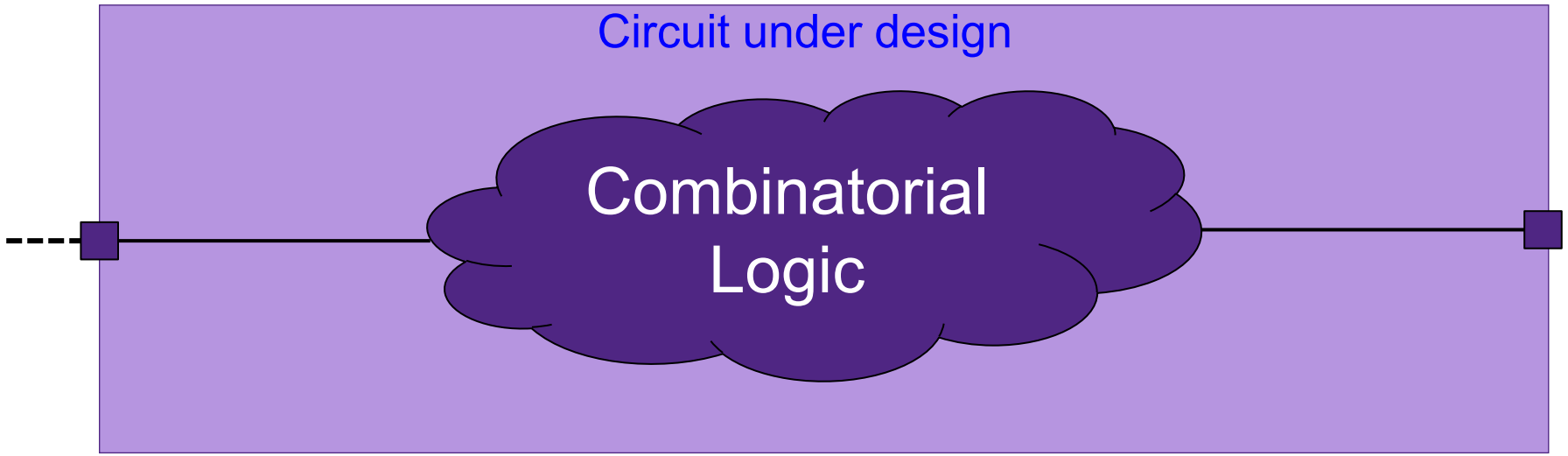
Hold time violation: Clock skew example





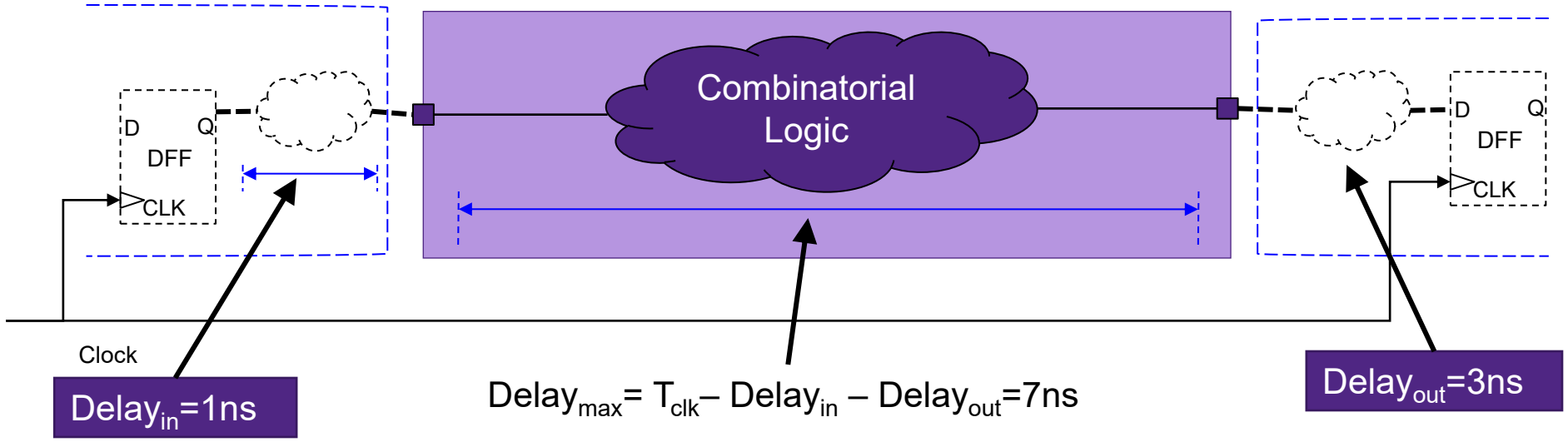
Combinatorial Designs

- Sometimes the design is a combinatorial circuit
- There is no clock to constraint timing

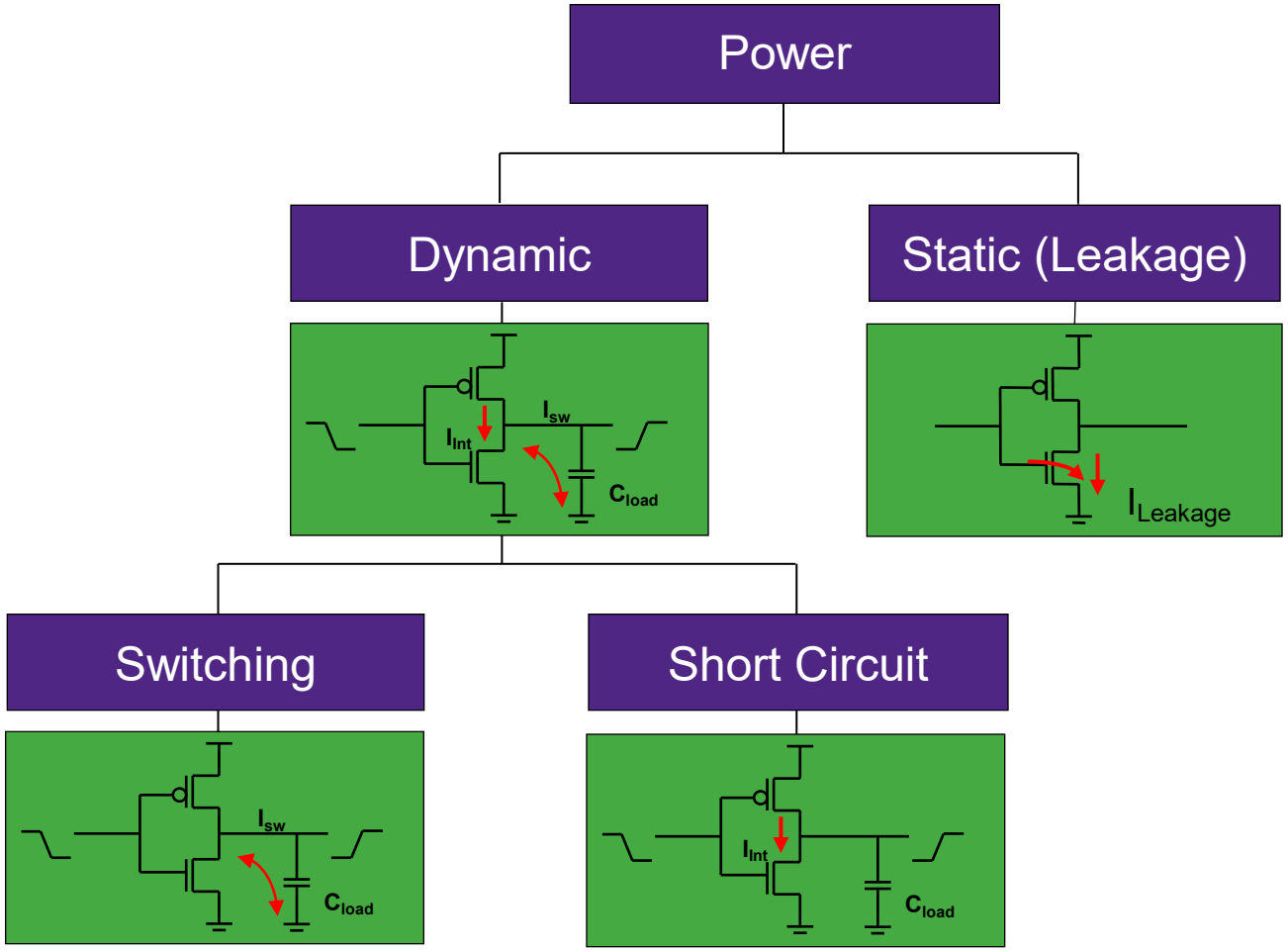


Combinatorial Designs (2)

- Combinatorial circuit is put in the same clocked environment as the clocked one
- An abstract clock called “Virtual clock” is defined for this environment
- By setting correct clock period, input/output delays, the delay of the combinatorial logic can be controlled



Power Consumption



Course Overview

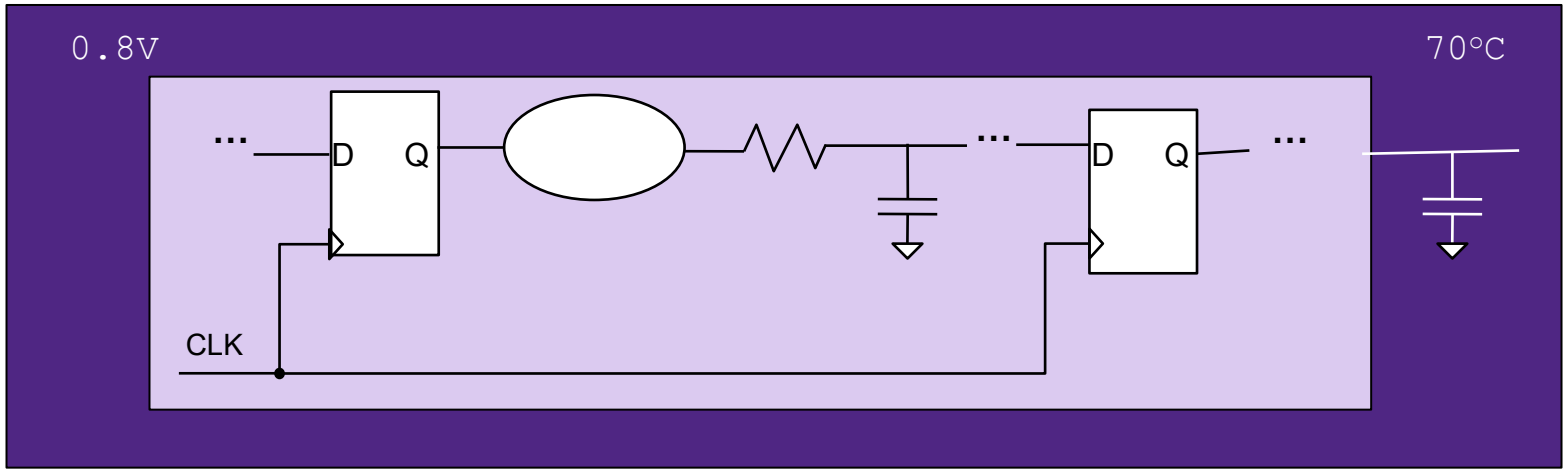
- Introduction
- Timing and Area Constraints
- Multi-Corner Multi-Mode and Mapping
- Application Options and Compile Flow
- Floorplan
- Placement
- Clock Tree Synthesis (not covered)
- Routing (not covered)
- Signoff (not covered)

Factors Affecting Timing

- Constraints describe delays available for register-to-register, input, output paths. But the conditions under which the delays must be met are not described by constraints.
- The path delays are affected by output capacitive loading, input transition times, operating conditions, interconnect parasitic RCs.

Factor Affecting Timing (2)

- Clock, input/output delay constraints are not sufficient for modeling and optimizing all logic path delays.
- It must be described:
  - Input transition times
  - Output loading
  - PVT conditions
  - Parasitic RCs



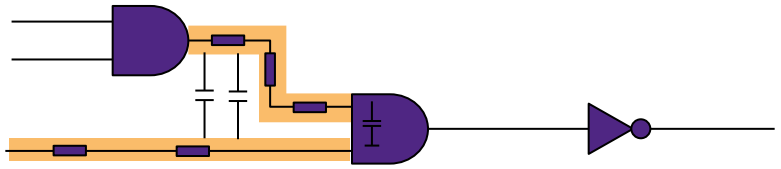
Wire Load Model (logical Synthesis)

- The wire delays must be estimated during the timing analysis before placement and routing in classic Digital Design Flow (*Logical Synthesis-> IC Compiler II*)
- Wire load model (WLM) is the simplest method of estimation
- WLM gets a rough value for the total wire capacitance based on the size of the chip and the fanout of the net.
- For each net wire load model obtains:
  - a parasitic resistance value
  - a capacitance value
- WLMs are not specific to each design and based on statistical averages.

*WLMs **are not** used in Fusion Compiler due to Physical Synthesis concept.*

WLM Parasitics estimation

Generalization

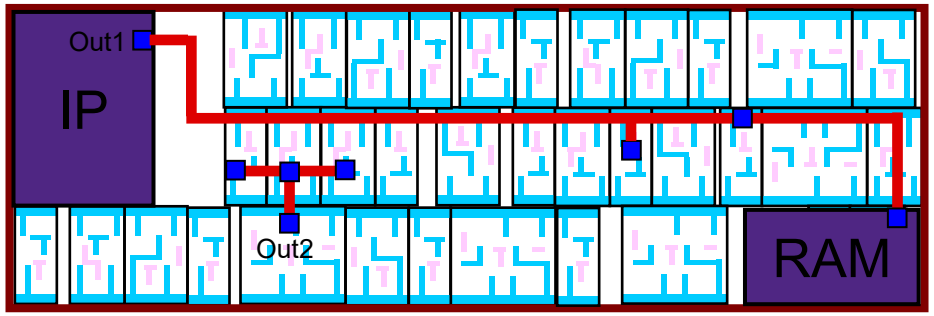


- All parasitics depend on interconnect length

$$R = length \cdot R_{unit\ length}$$

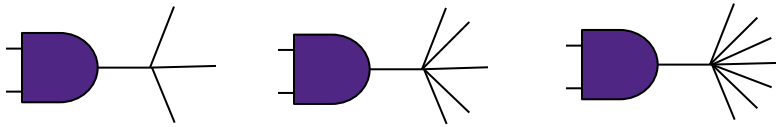
$$C = length \cdot C_{unit\ length}$$

$$Area = length \cdot Area_{unit\ length}$$

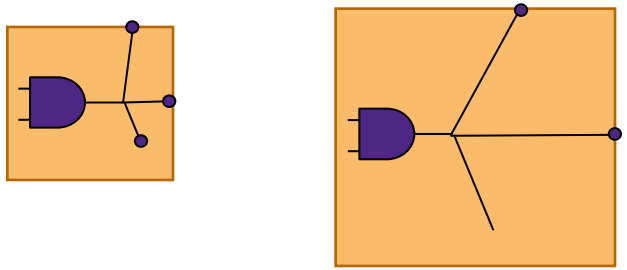


Length components

- The more the fanout (output connections) the larger the length



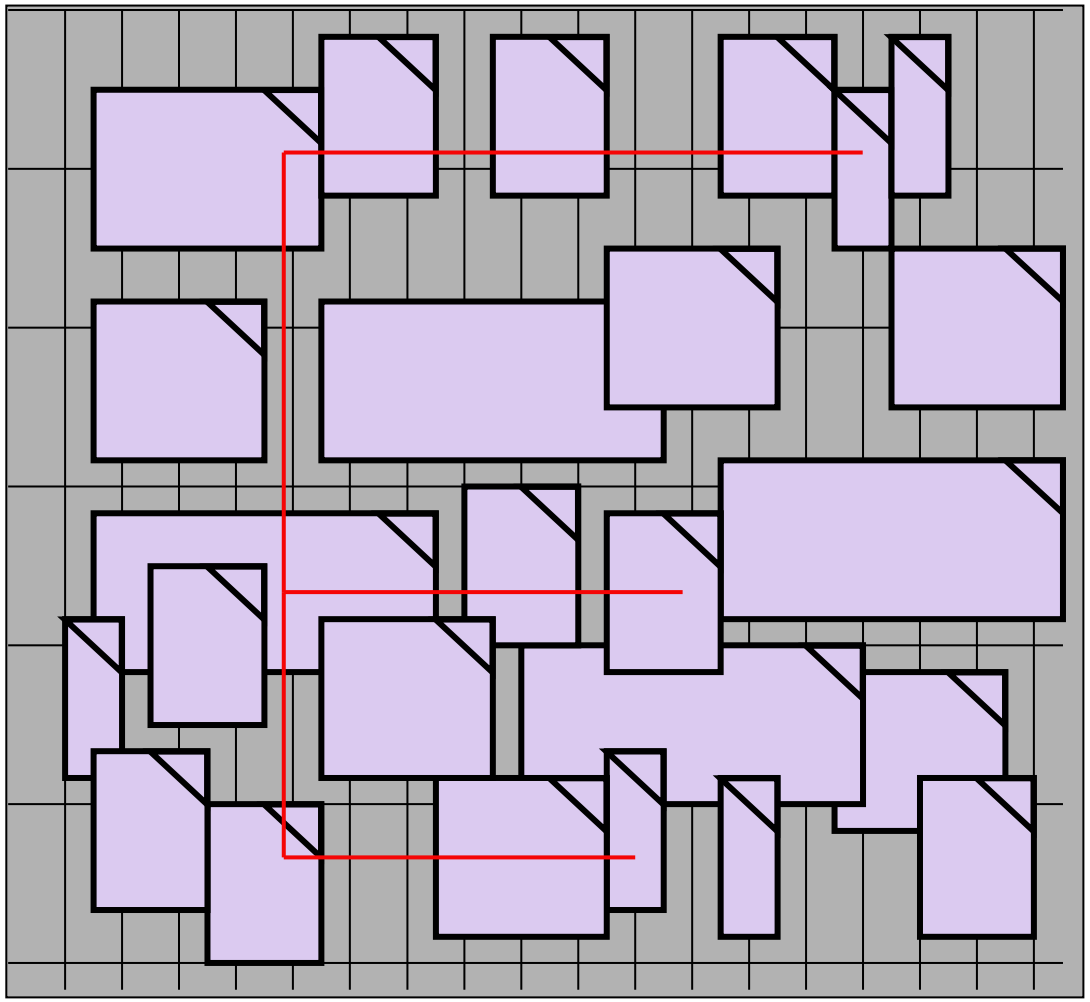
- The larger the chip (the more gates it has) the more the length



length = f (gate count, fanout)

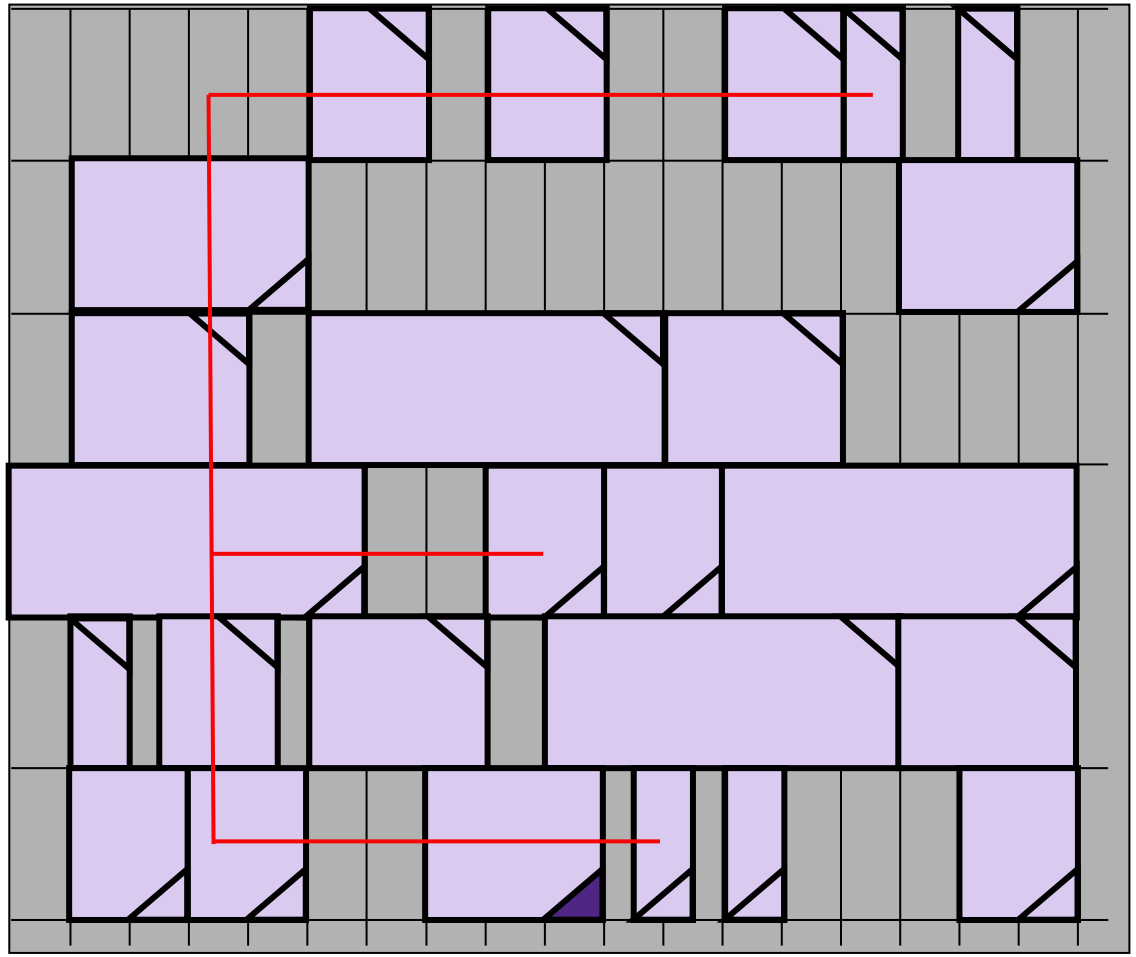
Fusion Compiler Parasitics estimation: Coarse Placement

- Early estimation based on Coarse Placement
  - All the cells are placed in the approximate locations but **they are not** legally placed.
  - Interconnect length can be estimated using Manhattan distance
  - RC parasitics calculated by routing direction and length based on TLU+ information



Fusion Compiler Parasitics estimation: Legalized Placement

- During legalization the cells are moved to their legal locations and flipped to legal orientation.
- Parasitics estimation will get more accurate in a legal design
- The router engine can be called to assign tracks and estimate RC based on actual routing metal layers, improving accuracy further



Placement-based parasitics estimation instead of wireload models is a key differentiator of physical synthesis over traditional synthesis.

Operating Conditions

- The operating conditions of a design include the following parameters:
  - Process
  - Voltage
  - Temperature
- The chip is intended to operate under this parameters.

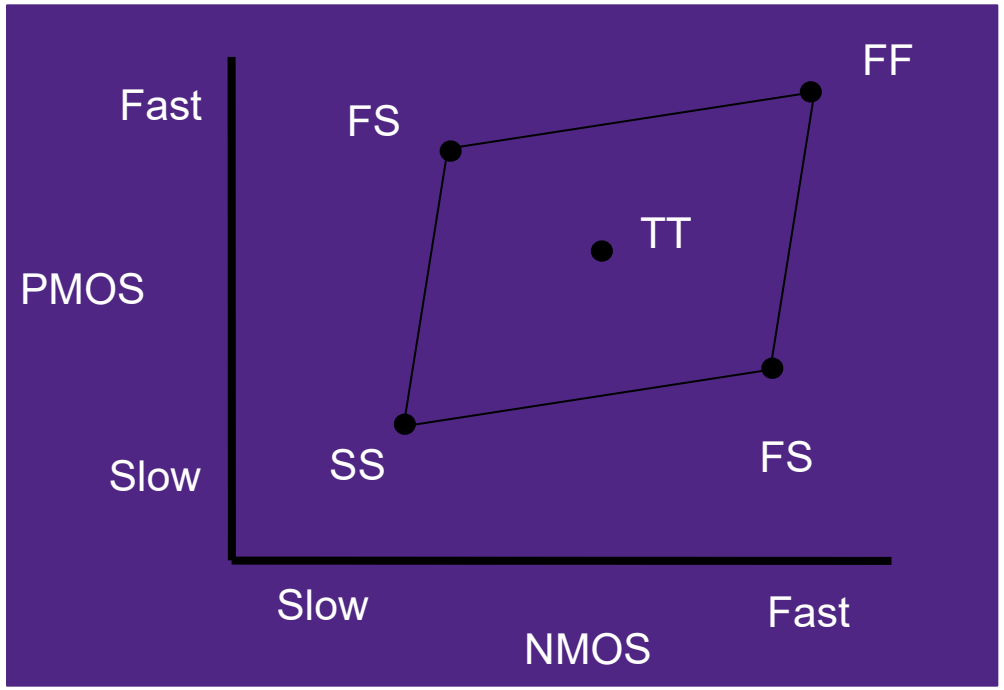
Operating Conditions (2)

- Process variation
  - Deviations in the semiconductor fabrication process
- Supply voltage variation
  - Design's supply voltage can vary from the established value during day-to-day operation.
- Operating temperature variation
  - Effects on performance caused by temperature fluctuations

- **set_operating_conditions** ${OP_COND}
- **set_process_number** ${PROCESS}
- **set_voltage** ${VOLTAGE}
- **set_temperature** ${TEMPERATURE}

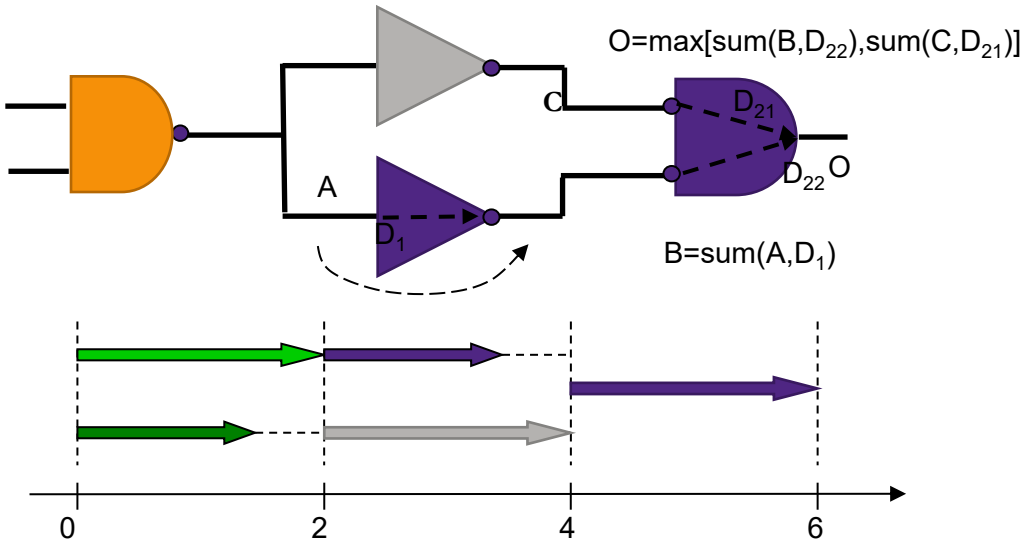
Process and Operating Conditions

- For being able to do MC analysis standard cell libraries are characterized for the following features
  - Supply voltage range
  - Parametric process variation
  - Temperature range

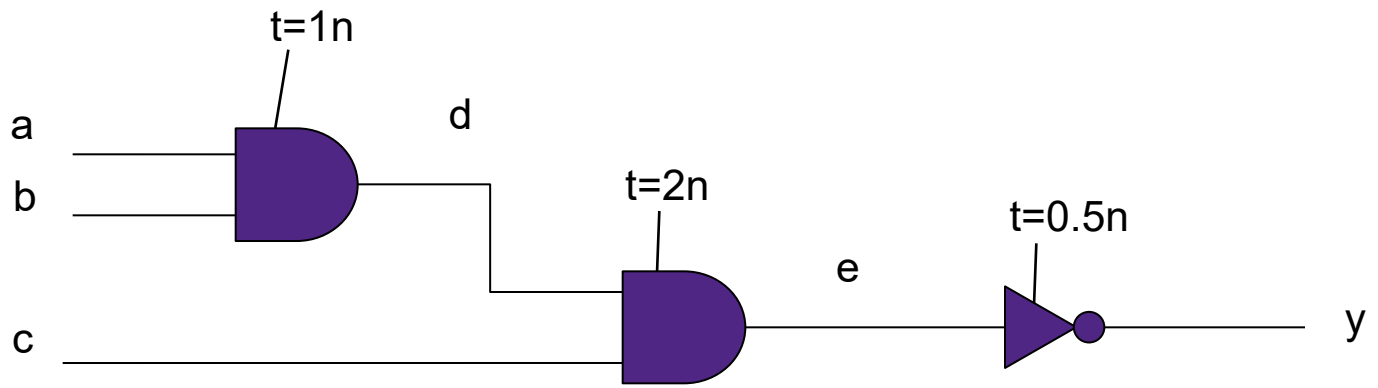


Static Analysis

- During synthesis path delays are calculated using static analysis
  - Static Timing Analysis (STA)
  - Static Power Analysis



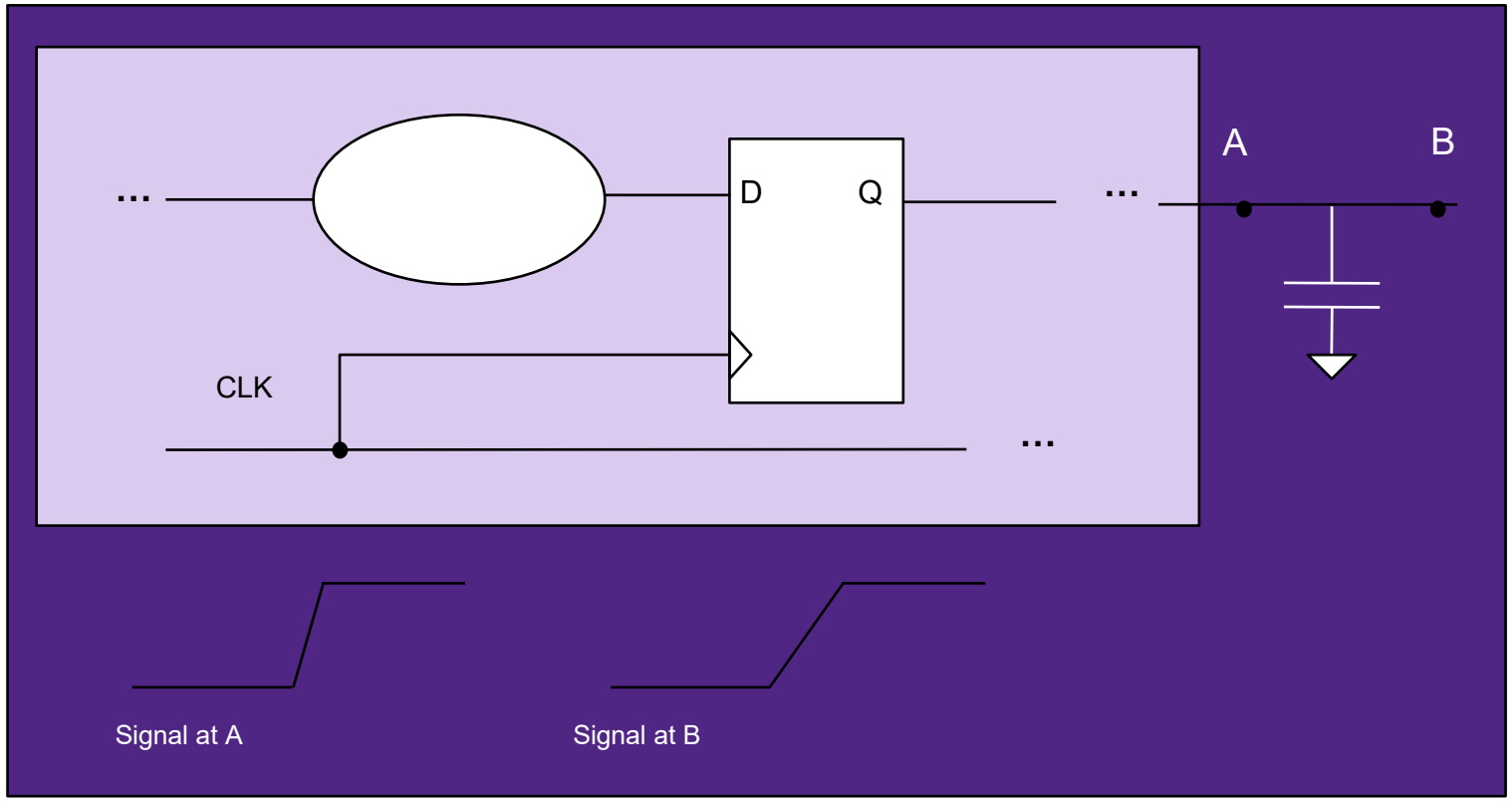
Path Delay



t_{delay}=3.5n

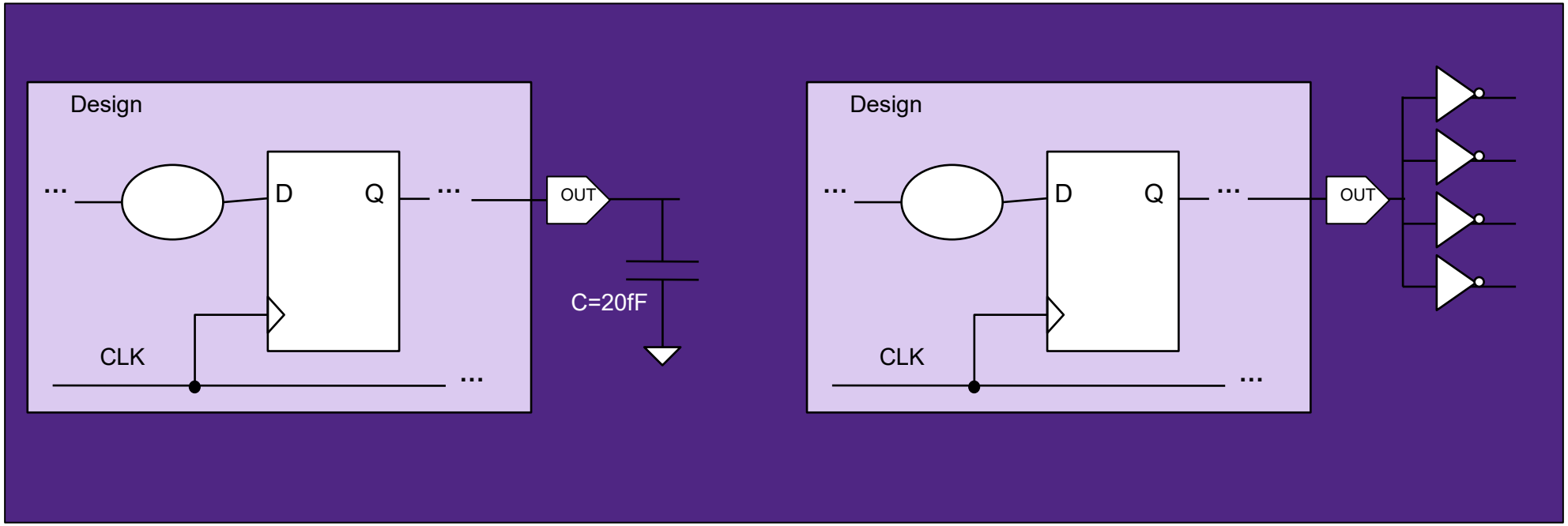
Effect of Output Capacitive Load

- Capacitive load on an output port affects the transition time, and thus the cell delay of the output driver.

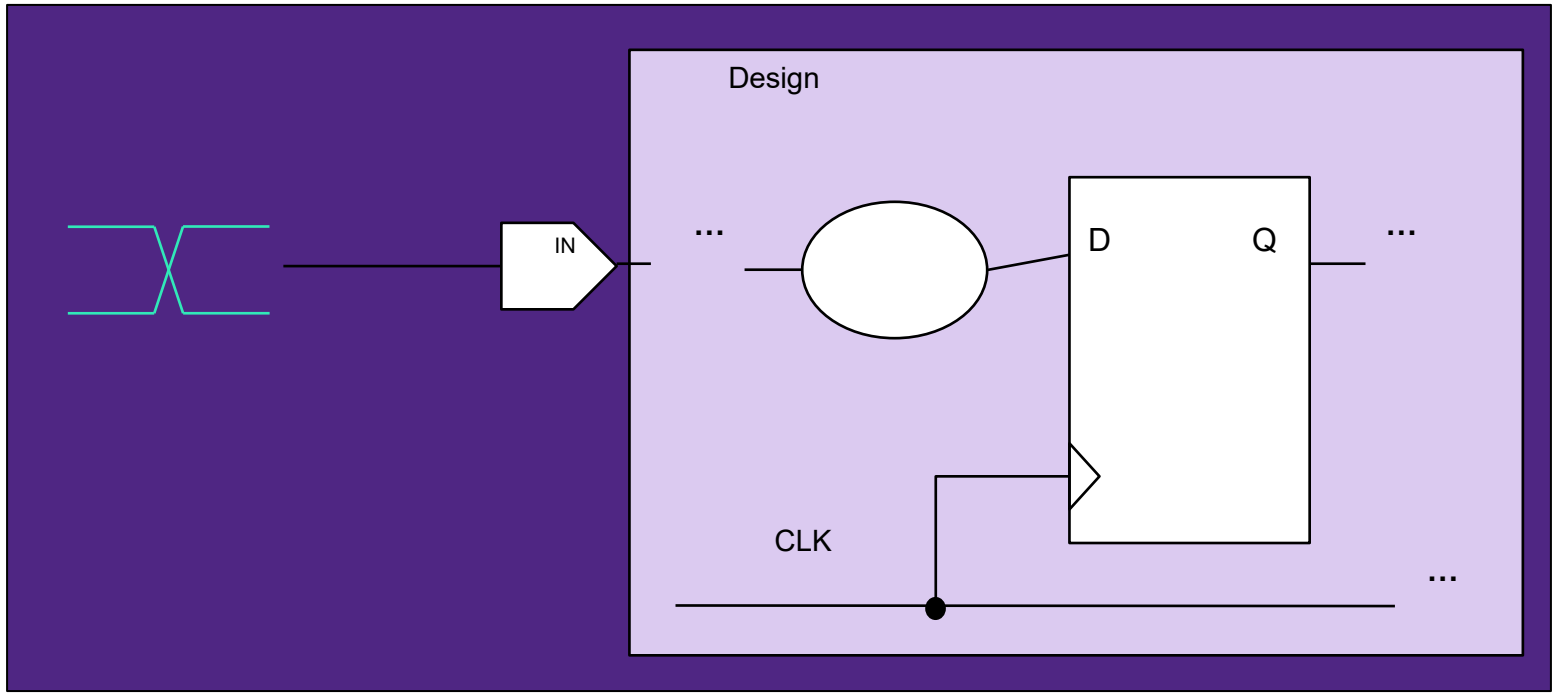


Modeling Output Capacitive Load

- For modeling output capacitive load a specific capacitive value or capacitance of a gate can be used.



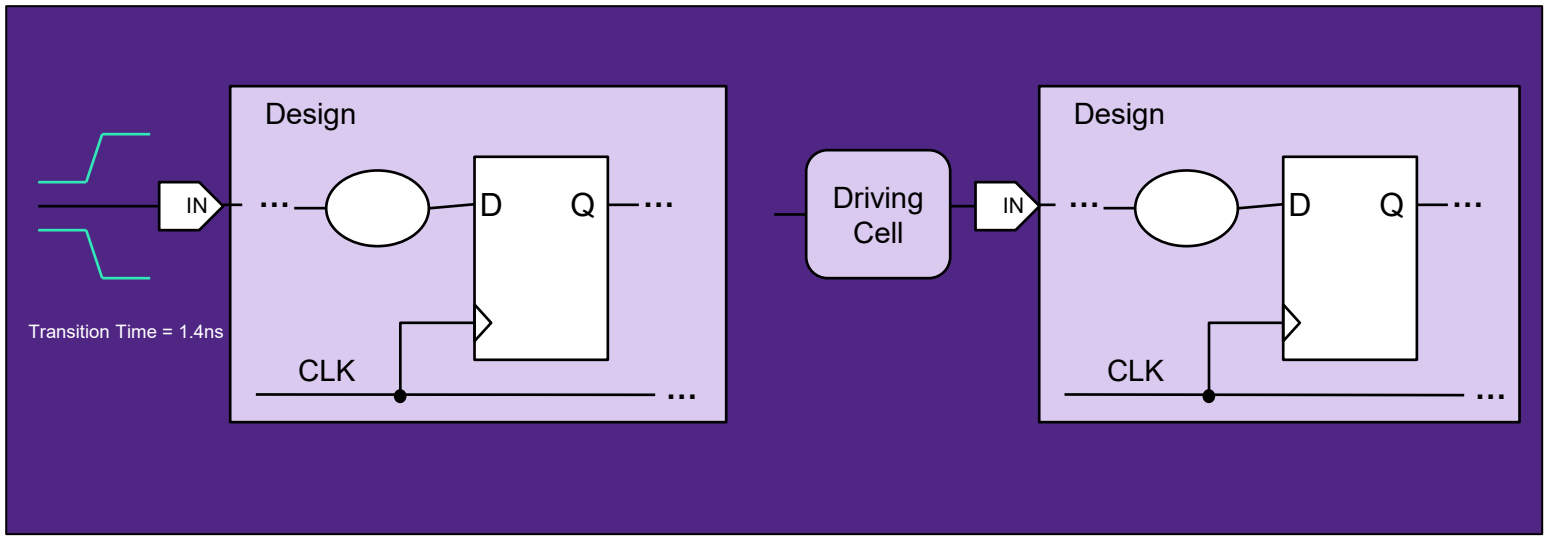
Effect of Input Transition Time



- Rise and fall transition times on an input port affect the cell delay of the input gate.
- It is therefore important to accurately model transition times on all inputs.

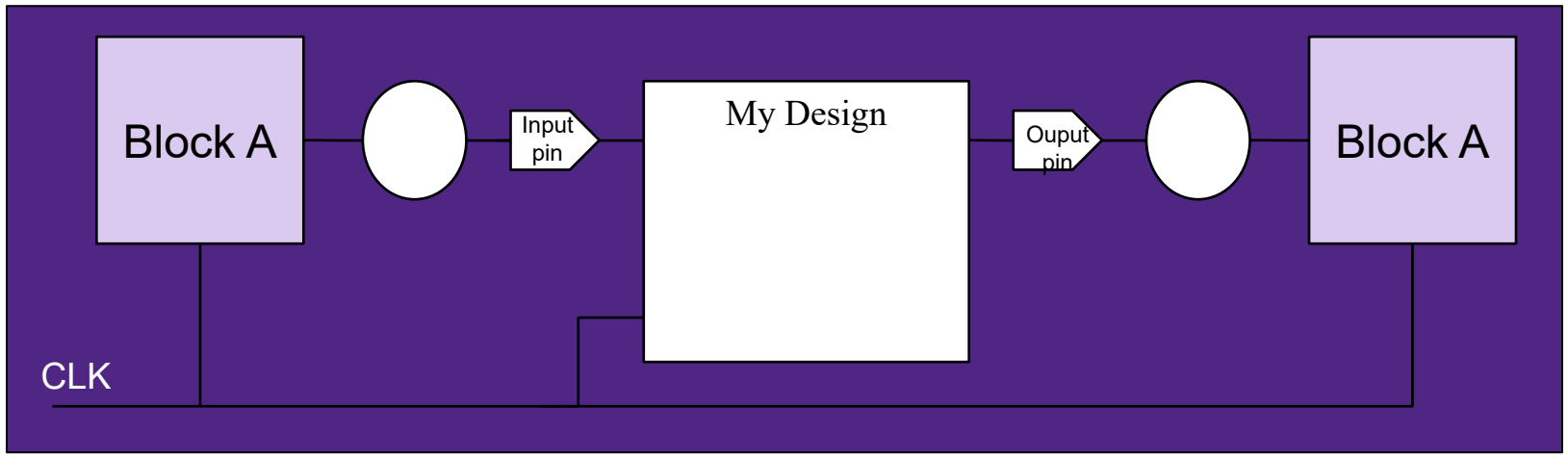
Modeling Input Transition

- For modeling input transition a specific transition time value or a driving cell on the input pin can be used.



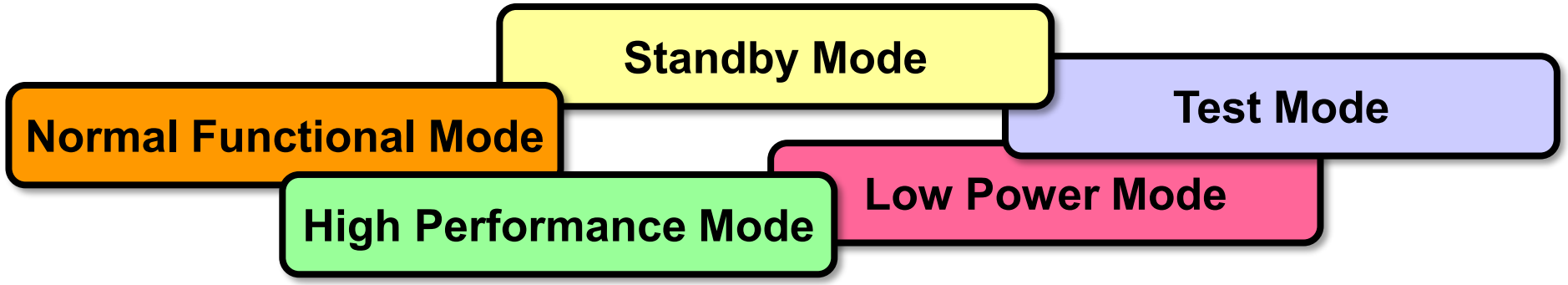
Load Budgeting

- When using top-level design planning and partitioning the design into multiple blocks, block interface conditions can be captured
- Timing budgets are created, sharing the available clock period across the system

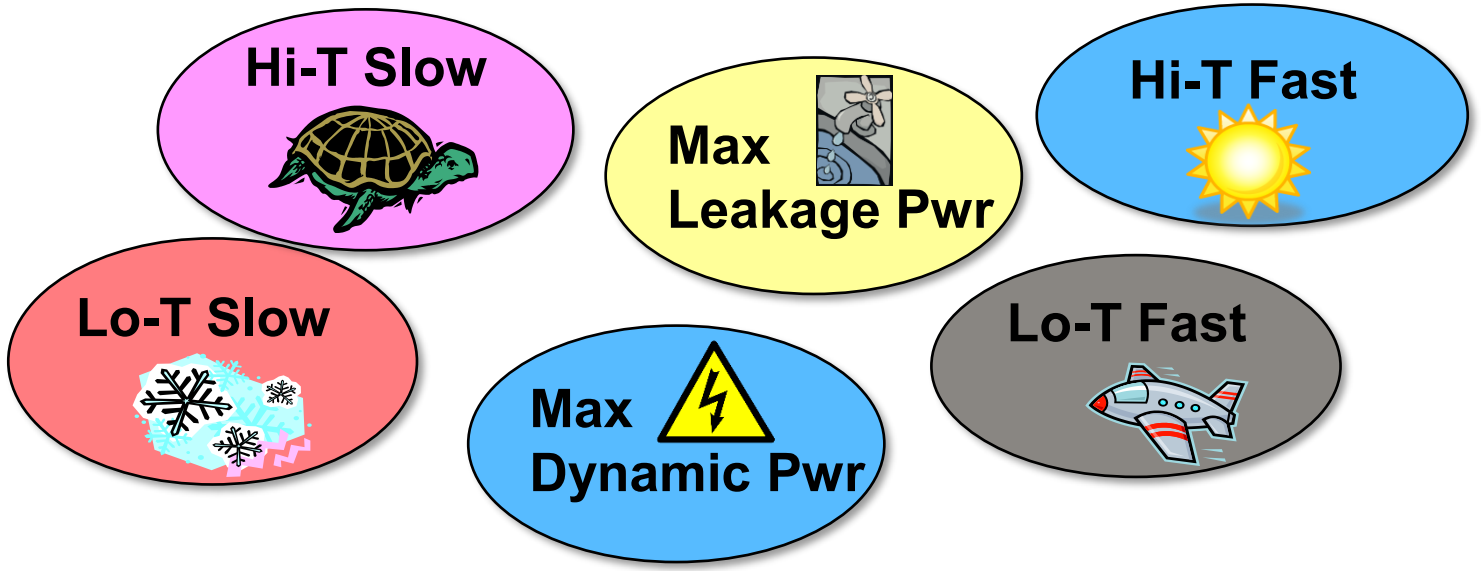


Multiple Modes and Corners

Today's chips must operate in multiple modes

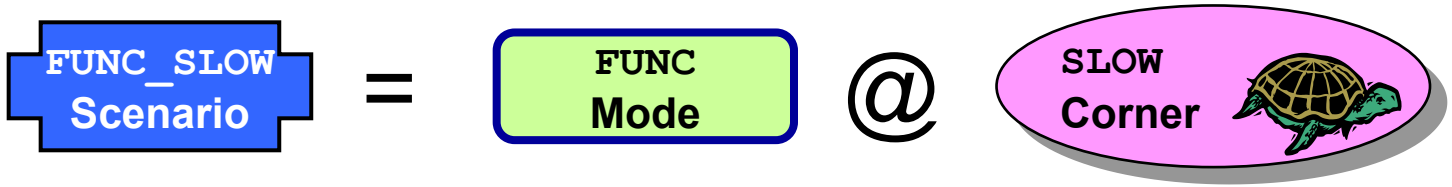


…and across multiple PVT corners

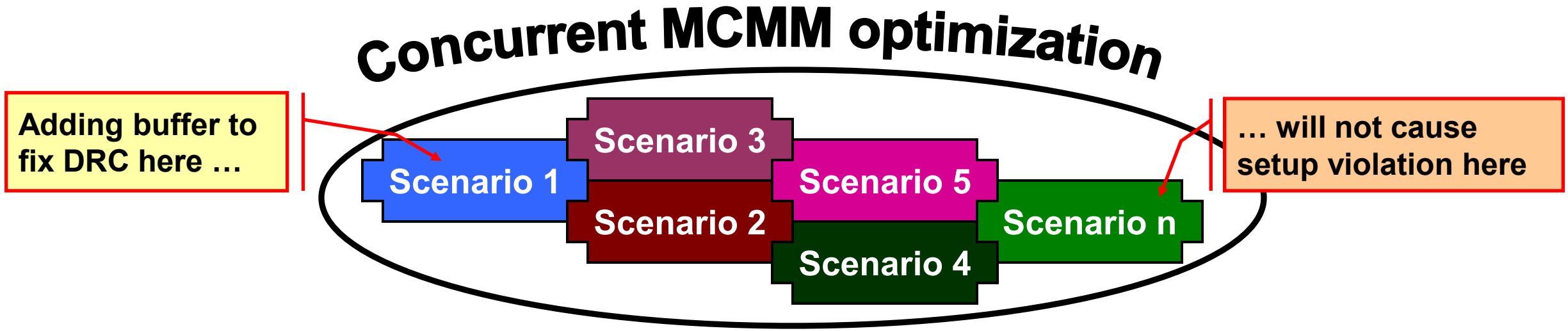


Concurrent MCMM Optimization

- Fusion Compiler performs concurrent optimization under multiple corner and mode combinations, called scenarios

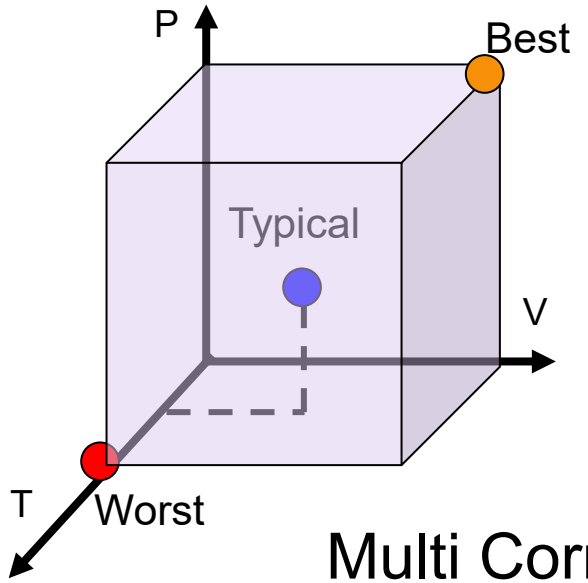


- Improves each violation in a scenario, while trying not to cause/increase a violation in another scenario



The Multiple Analysis Corners

- Fusion Compiler supports Multi-Corner Multi-Mode (MCMM) setup.
- Optimization is performed concurrently for different combinations of specified corners and modes, which are called scenarios.
- Concurrent optimization helps to prevent violations in other scenarios, while optimizing for one.



Multi Corner => P,V,T

Creation of Corners and Modes

1. Create corners for your design:

- **create_corner** ${CORNER_NAME}
- **…**

2. Create modes for your design:

- **create_mode** ${MODE_NAME}
- **…**

- Some useful commands:
  - **get_corners**
  - **get_modes**
  - **report_corners**
  - **report_modes**
  - **current_corner**
  - **current_mode**

Creation of Scenarios

Create scenarios for your design:

```
create_scenario -mode ${MODE_NAME} -corner ${CORNER_NAME} \
  -name ${SCENARIO_NAME}
...
```

Scenarios are created for appropriate modes and corners combinations.

Without specifying **–name** option, created scenario will be named like: ${MODE_NAME}::${CORNER_NAME}

- Some useful commands:

```
get_scenarios
report_scenarios
current_scenario
```

Scenarios' Configuration

- Created scenarios can be configured for different analysis types:
  - Timing (Setup and Hold)
  - Power (Leakage and Dynamic)
  - Design Rules (Max Transition, Max and Min Capacitance)

```
set_scenario_status ${SCENARIO_1} -setup false -hold true \
  -leakage_power false -dynamic_power true  -max_transition false \
  -max_capacitance true  -min_capacitance true -active true
set_scenario_status ${SCENARIO_2} -all -active true
set_scenario_status ${SCENARIO_3} -all -active false
```

- Analysis and optimization during the flow will be performed for active scenarios only.

Load the Constraints

- For each corner, mode and scenario user can load its specific constraints:

```
current_scenario ${SCENARIO_1}
read_sdc ${CORNER_1_CONSTRAINTS}.sdc
read_sdc ${MODE_1_CONSTRAINTS}.sdc
read_sdc ${SCENARIO_1_CONSTRAINTS}.sdc
current_scenario ${SCENARIO_2}
read_sdc ${CORNER_2_CONSTRAINTS}.sdc
read_sdc ${MODE_2_CONSTRAINTS}.sdc
read_sdc ${SCENARIO_2_CONSTRAINTS}.sdc
```

```
read_sdc ${GLOBAL_CONSTRAINTS}.sdc
```

- Also, user can load global constraints, which will be shared.

Type of constraints

```
create_mode func

create_corner ss0p95v125c
create_corner ss0p95vn40c

create_scenario -mode func -corner ss0p95v125c -name func_ss0p95v125c
create_scenario -mode func -corner ss0p95vn40c -name func_ss0p95vn40c

report_scenarios

######## The current_mode is the last one that was created, i.e. func
######## The current_corner is the last one created, i.e. ss0p95vn40c
######## The current_scenario is the last one created, i.e. func_ss0p95vn40c

# Populate corner object specific constraints

current_corner ss0p95v125c ;# This changes the current corner from ss0p95vn40c to ss0p95v125c
set_voltage 0.95
set_temperature 125
set_parasitic_parameters -library saed32nm_1p9m_tech -late_spec maxTLU -late_temperature 95
set_load 0.0005 [all_outputs]

current_corner ss0p95vn40c ;# This changes the current corner from ss0p95vn40c to ss0p95n40c
set_voltage 0.95
set_temperature -40
set_parasitic_parameters -library saed32nm_1p9m_tech -late_spec maxTLU -late_temperature -40
set_load 0.0005 [all_outputs]

# Populate mode object specific constraints
###### The current_mode is func, so these are loaded into func
create_clock -period 2.0 -name Clk [get_port Clk]
create_clock -period 2.0 -name VClk

# Populate scenario object specific constraints

####### If these scenario specific constraints did not have the -scenarios option they would be loaded into just the current_scenario,
func_ss0p95vn40c. However, the -scenarios option means they are loaded into both scenarios.
set_input_delay  -scenarios {func_ss0p95v125c func_ss0p95vn40c} \
  -clock VClk 1.0 [remove_from_collection [all_inputs] [get_port Clk]]
set_driving_cell -scenarios {func_ss0p95v125c func_ss0p95vn40c} \
  -input_transition_rise 0.2 -input_transition_fall 0.2 -lib_cell IBUFFX4_HVT -pin Y [all_inputs]
set_output_delay -scenarios {func_ss0p95v125c func_ss0p95vn40c} \
  -clock VClk 1.0 [all_outputs]
set_clock_latency -scenarios {func_ss0p95v125c func_ss0p95vn40c} \
  0.5 [get_clock Clk]
set_clock_latency -scenarios {func_ss0p95v125c func_ss0p95vn40c} \
  0.5 [get_clock VClk]
```

Mode-Specific	Corner-Specific	Scenario-Specific	Global
Constraints that define or modify the topology of the timing graph	Constraints that modify the calculated delay on an object	Constraints that have a time, load, resistance, or drive value, and can refer to a modal object (e.g. -clock)	Constraints that apply to all corners, modes and scenarios of a given netlist
create_clock create_generated_clock group_path set_case_analysis set_cell_mode set_clock_gating_check set_clock_groups set_clock_sense set_data_check set_disable_timing set_false_path set_latch_loop_breaker set_max_delay set_min_delay set_multicycle_path set_propagated_clock set_sense	set_aocvm_coefficient set_extraction_options set_load set_operating_conditions set_parasitic_parameters set_process_label set_process_number set_temperature set_timing_derate set_voltage	read_sdc set_annotated_arrival set_annotated_delay set_annotated_required set_annotated_transition set_awp_thresholds set_clock_latency set_clock_transition set_clock_uncertainty set_drive set_drive_resistance set_driving_cell set_fanout_load set_ideal_latency set_ideal_transition set_input_delay set_input_transition set_max_capacitance set_max_time_borrow set_max_transition set_min_capacitance set_min_pulse_width set_output_delay set_path_margin set_switching_activity	set_disable_clock_gating_check set_dont_touch set_dont_touch_network* set_ideal_network set_power_clock_scaling

Analyze The Design for Different Scenarios

- User can specify for which scenario, corner or mode to collect the reports in different ways:

```
current_scenario ${SCENARIO_1}
report_qor
```

```
report_qor -modes ${MODES} -corners ${CORNERS} -scenarios ${SCENARIOS}
```

- Command examples for collecting reports:

```
report_power
report_timing
report_area
report_design
…
```

Course Overview

- Introduction
- Timing and Area Constraints
- Multi-Corner Multi-Mode and Mapping
- Application Options and Compile Flow
- Floorplan
- Placement
- Clock Tree Synthesis (not covered)
- Routing (not covered)
- Signoff (not covered)

Agenda

- ***Application Options***
- compile_fusion flow
- Attributes

Application Options

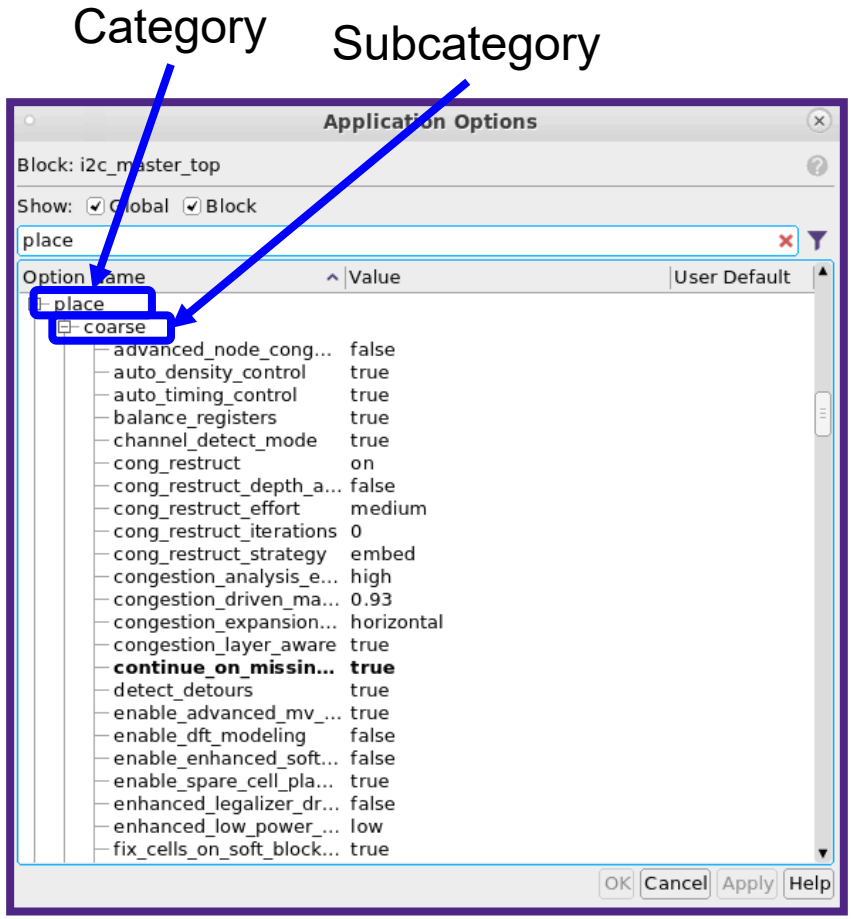
- Application options in Fusion Compiler are used to control the tool behavior in different stages as in IC Compiler II.
- Syntax:

  - **set_app_options -name** {category.option_name} **-value** {option_value}
  - **set_app_options -name** {category.subcategory.option_name} **-value** {option_value}

- Application options can be:
  - Block-scoped: Application options are applied only to the block, with which user is currently working on. They can be saved in the design library.
  - Global-scoped: Application options are applied to all blocks in the design library. They are set only during current working session and are not saved in the design library.

Examples of Application Options

- Application options help to control different stages of the design process.
- Examples:
  - hdlin.*
  - compile.*
  - place.*
  - clock_opt.*
  - route.*
  - opt.*
  - …



File -> Application Options

Commands for Interaction

- Setting Application options:

  - **set_app_options** **-name** {category.option_name} **-value** {option_value}
  - **set_app_options** **-name** {category.subcategory.option_name} **-value** {option_value}

- Get list of available Application options:

  - **get_app_options**
  - **get_app_options** compile.*
  - **get_app_options** **-block** [current_block]
  - **get_app_options** **-non_default**

- Report Application options:

  - **report_app_options**
  - **...**

Agenda

- Application Options
- ***compile_fusion flow***
- Attributes

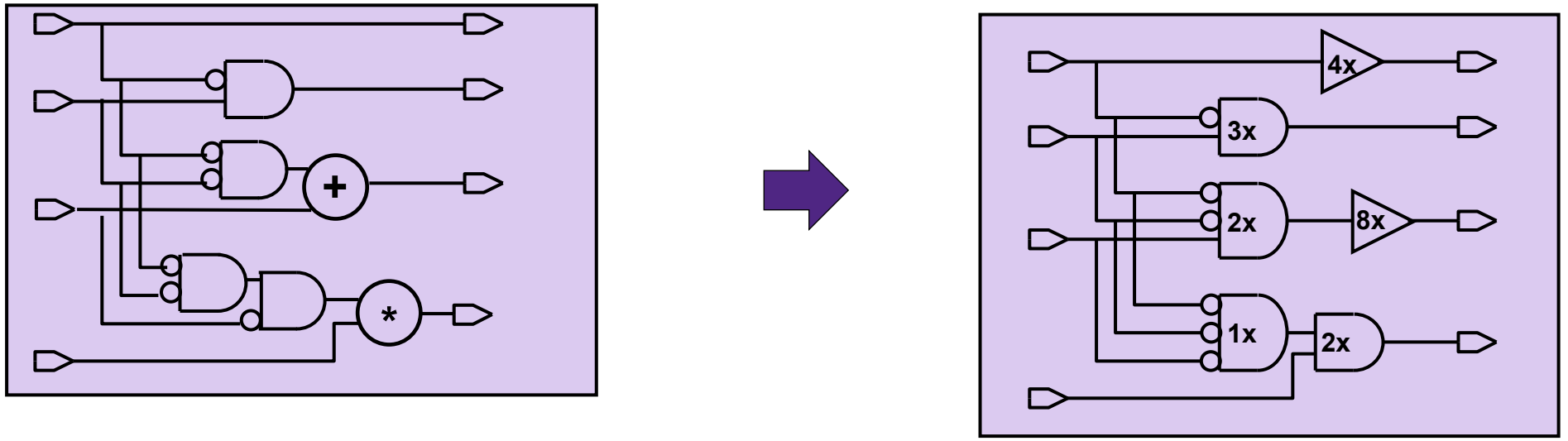
compile_fusion

- compile_fusion consists of the below stages:

1. **initial_map:**
   Design mapping
2. **logic_opto:**
   Logic-based delay optimization. ***Automatic floorplan creation*.**
3. **initial_place:**
   Coarse placement.
4. **initial_drc:**
   Buffer tree creation for high-fanout nets and design rule violation fixing.
5. **initial_opto:**
   Incremental placement and optimization.
6. **final_place:**
   Final placement to improve timing and congestion.
7. **final_opto:**
   Final optimization and legalization.

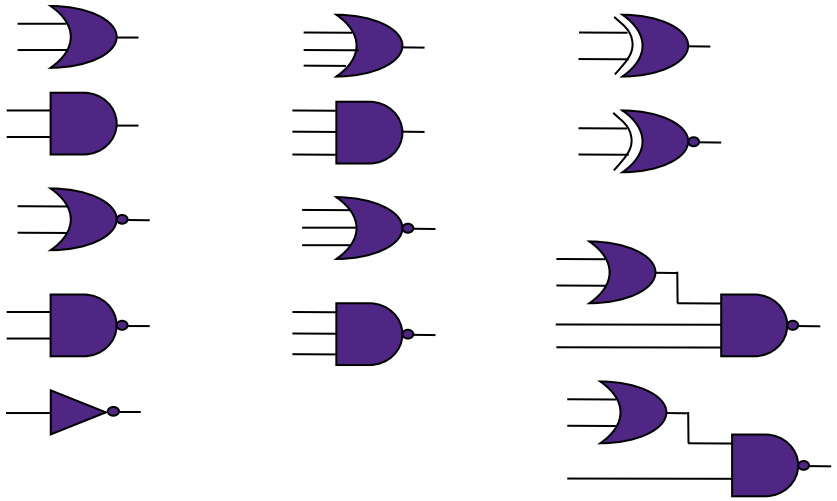
initial_map step

- Map technology-independent representation of circuit to technology library
- maps generic logic representation (wvgtech) to target library cells and performs area optimization
- Also inserts ICGs and MV cells



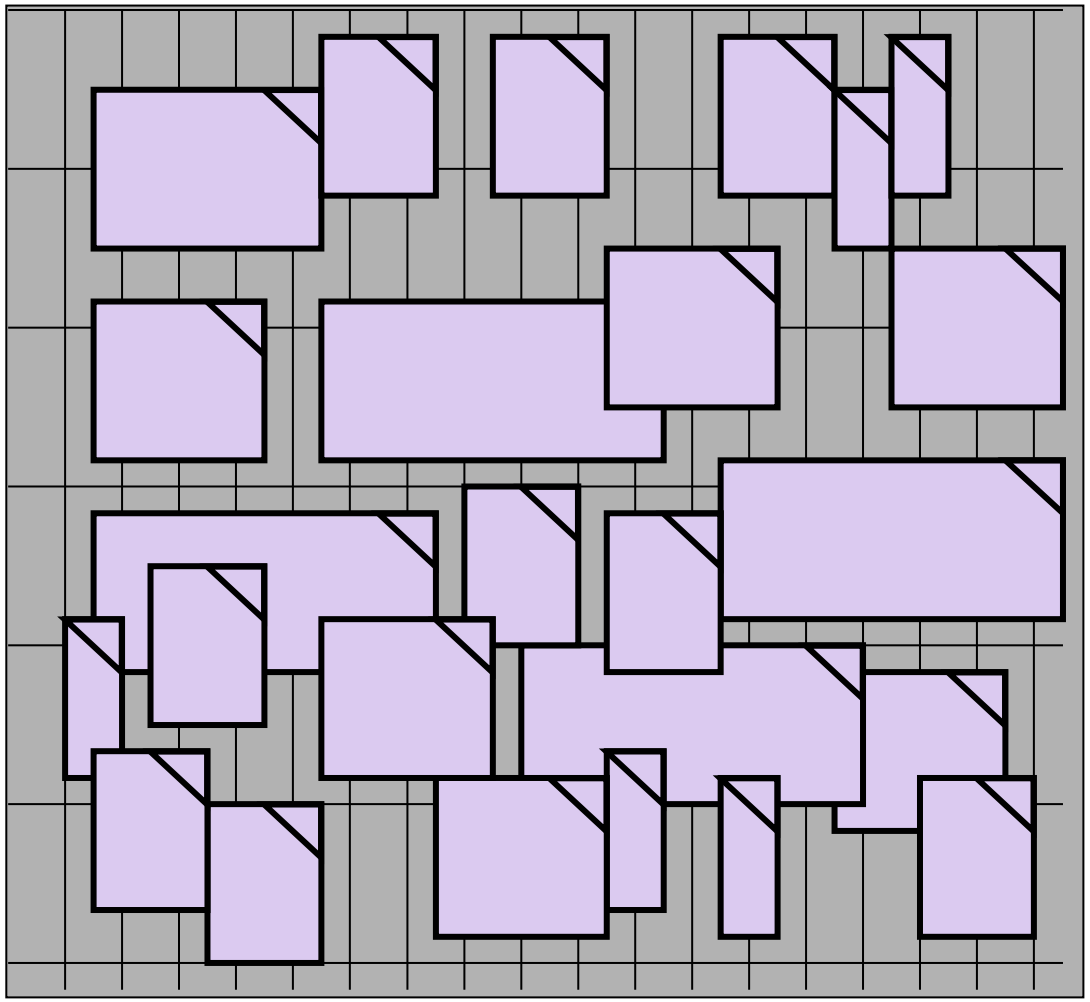
logic_opto step (1): Cell Library

- Implement the optimized logic network using a set of gates which form a library
- Performs logic delay optimization and wire-length driven placement
- Also performs auto-floorplanning if enabled
- After this stage, you could run DFT insertion
- Each gate has a cost (area, delay, etc.)



initial_place step:

- After optimizing the logic structure, initial coarse placement allows for a first accurate estimation of RC parasitics and congestion for further optimization steps
- Performs buffering aware timing driven placement (congestion not considered)

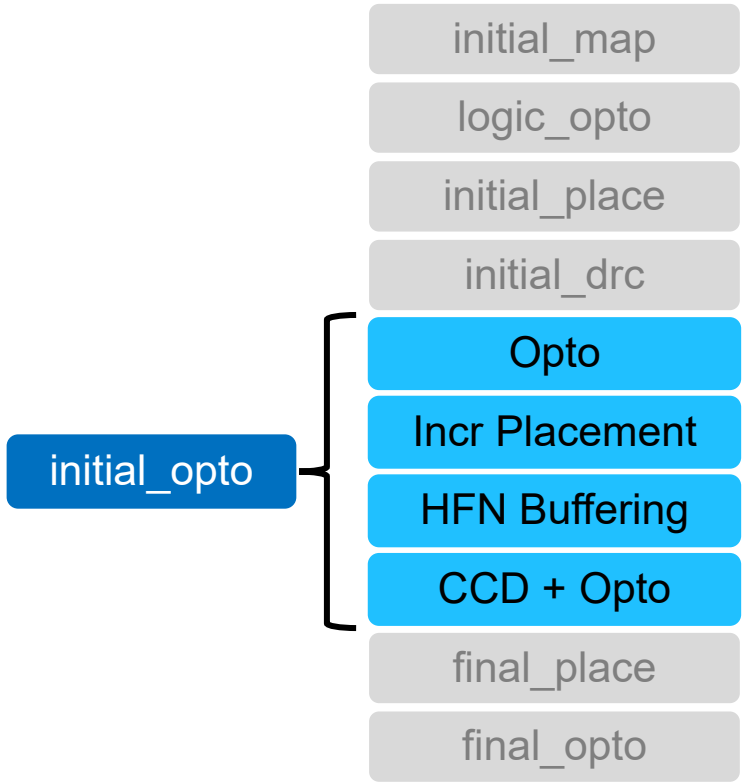


Initial_drc step

- builds the high-fanout buffer trees
  - Includes buffering of low fanout nets with large DRC violations

initial_opto step: Coarse Placement

- performs:
  - Optimization of the placed and HFN-inserted netlist
  - Incremental placement as well as incremental HFN buffering + DRC fixing
  - Another round of optimization
  - CCD optimization
  - ICGs optimized to meet constraints
  - **End result:** Placed, not fully legalized



Final_place / final_opto steps

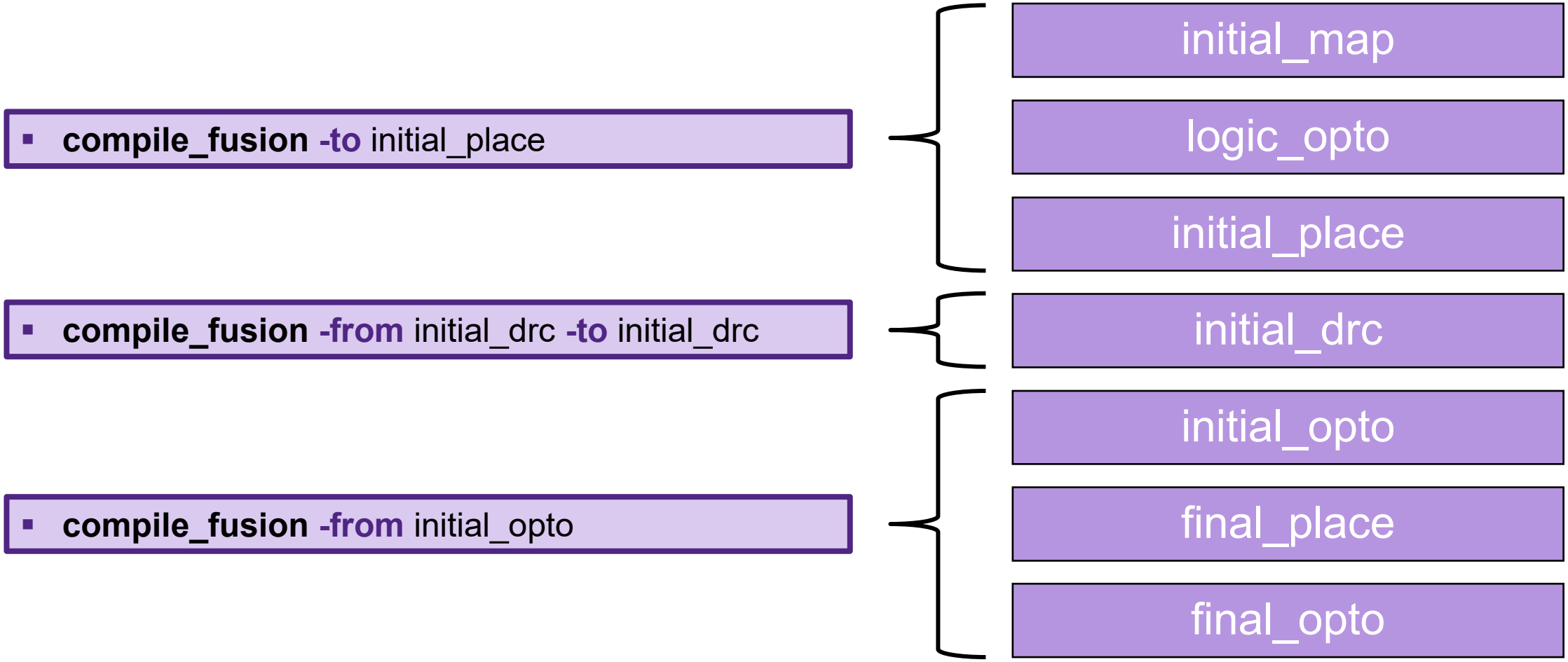
- *final_place* performs final, incremental timing- and congestion-driven placement

- *final_opto* performs several rounds of optimization on all metrics (timing, power, logical DRC), including CCD optimization
  - The end result is a fully placed and legalized design, ready for clock tree synthesis

- At this point, if using the Unified Flow, continue the flow by running CTS using `clock_opt`

Controlling compile_fusion stages

- After compile_fusion implementation there will be a legally placed and optimized design.



Agenda

- Application Options
- compile_fusion flow
- ***Attributes***

Attributes

Each object has its own attributes.

- To set attribute use:
  - **set_attribute ${object}** –name ${attribute_name} –value ${attribute_value}
- To get attribute value use:
  - **get_attribute ${object}** ${attribute_name}
- To remove attribute use:
  - **remove_attribute ${object}** ${attribute_name}

dont_touch Attribute

- To restrict an object in the design from being modified or optimized use dont_touch attribute.
  - **set_dont_touch** ${object} true
- Supported design objects:
  - Cells
  - Nets
  - Modules and cells (also hierarchical)
- To remove dont_touch attribute from the object:
  - **set_dont_touch** ${object} false
  - **remove_attribute** ${object} dont_touch

Restricting Network Optimization

- To restrict networks, like clock trees, from optimization use set_dont_touch_network command.
  - **set_dont_touch_network** [**get_clocks** ${clock}]
- To remove dont_touch attributes from the network's objects, like pins, nets, use:
  - **set_dont_touch_network** [**get_clocks** ${clock}] **-clear**

size_only Attribute

- To allow only sizing of cells during optimization use size_only attribute.
  - **set_size_only** [**get_cells** ${cell}] true
- To get size_only attribute's value on the cell:
  - **get_attributes** [**get_cells** ${cell}] size_only
- To remove size_only attribute from the cell:
  - **set_size_only** [**get_cells** ${cell}] false

Restricting Library Cell Usage

- To control cell usage during optimization use set_lib_cell_purpose command.
- To restrict library cell usage during design process, use:
  - **set_lib_cell_purpose -include** none [**get_lib_cells** ${tech_library}/${library_cell}]
- To allow the usage of library cell during hold fixing use:
  - **set_lib_cell_purpose -include** hold [**get_lib_cells** ${tech_library}/${library_cell}]
- To disallow the usage of library cell during CTS use:
  - **set_lib_cell_purpose -exclude** cts [**get_lib_cells** ${tech_library}/${library_cell}]

Ungrouping

- Timing and area optimization achieved by reducing the levels of logic and sharing logic;
- By default, the compile_fusion command automatically ungroups logical hierarchies;
- To disable automatic ungrouping for module DUT:
  - **set_ungroup** [**get_modules** {DUT}] false
- To enable automatic ungrouping for module DUT:
  - **set_ungroup** [**get_modules** {DUT}] true

Controlling Effort levels

- Timing effort can be controlled by the following commands:
  - **set_qor_strategy -metric** timing **-stage** synthesis
- Area effort can be controlled by:
  - **set_qor_strategy -metric** area **-stage** synthesis
- Power optimization can be controlled by:
  - **set_qor_strategy -metric** total_power **-stage** synthesis
  - **set_qor_strategy -metric** leakage_power **-stage** synthesis
- Congestion optimization can be controlled by:
  - **set_qor_strategy -metric** congestion **-stage** synthesis

Path Grouping

- Paths are groups based on the clock controlling the endpoint
- By default only on the worst path in each group is optimized
  - Delay optimization phase halts when most critical endpoint cannot be further improved
- By creating path groups user can control optimization priorities (weights)

Example: Request Fusion Compiler to spend less time on I/O paths (because of unknown constraints) creating separate path groups for the I/Os leaves the register-to-register paths in the original path groups

- **group_path** -**name** inpaths **-critical_range** 0.0 **-from** [**all_inputs**]
- **group_path** -**name** outpaths **-critical_range** 0.0 **-to** [**all_outputs**]

Course Overview

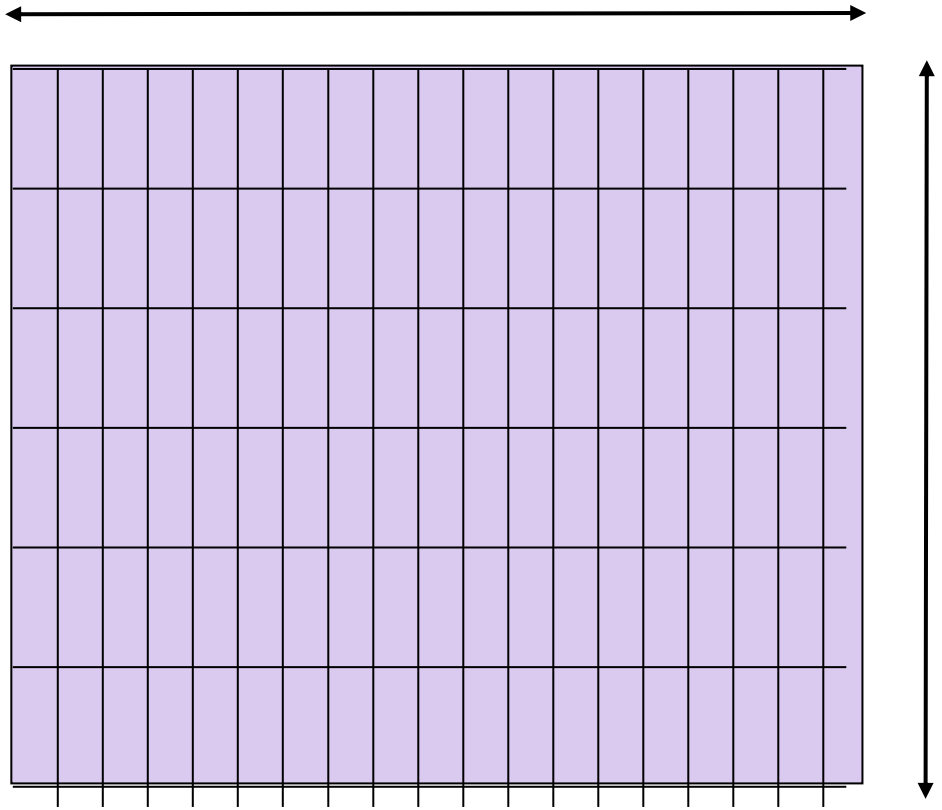
- Introduction
- Timing and Area Constraints
- Multi-Corner Multi-Mode and Mapping
- Application Options and Compile Flow
- Floorplan
- Placement
- Clock Tree Synthesis (not covered)
- Routing (not covered)
- Signoff (not covered)

Agenda

- ***Floorplan Overview***
- Floorplan Creation
- Power/Ground Network Creation
- IO cell placement

Floorplanning

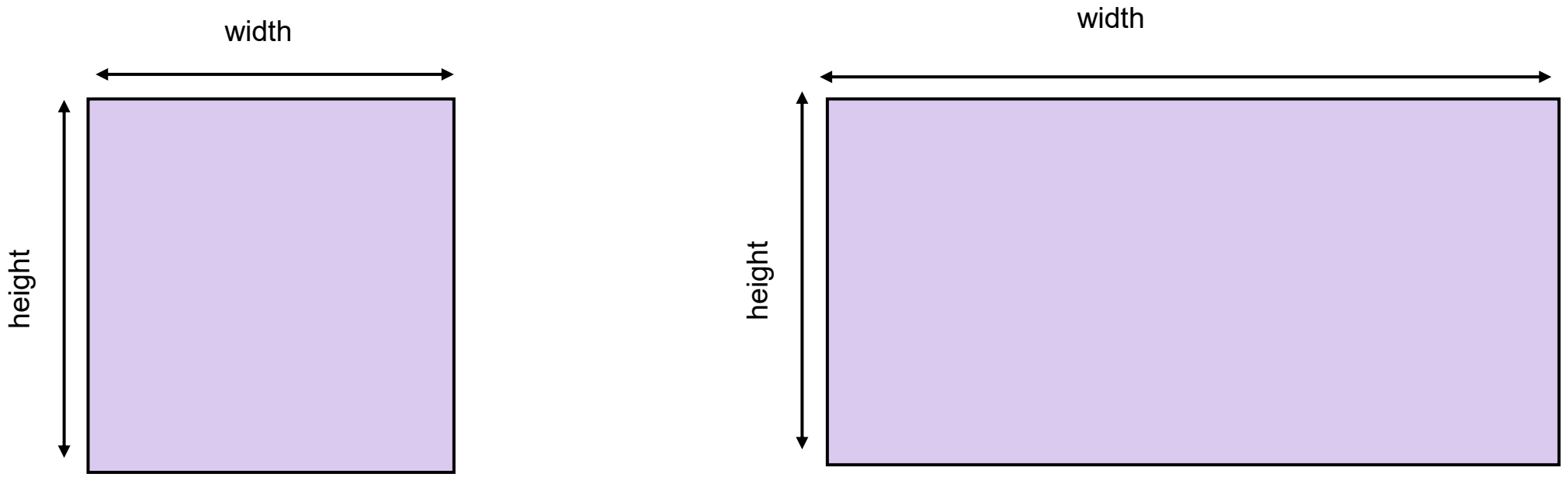
- During the floorplanning step the overall block is defined, including block size, shape, supply network, etc.



Floorplan

Floorplanning: Aspect Ratio

- Aspect ratio is the height to width ratio of a block
  - Defines the block shape
  - Default aspect ratio is 1

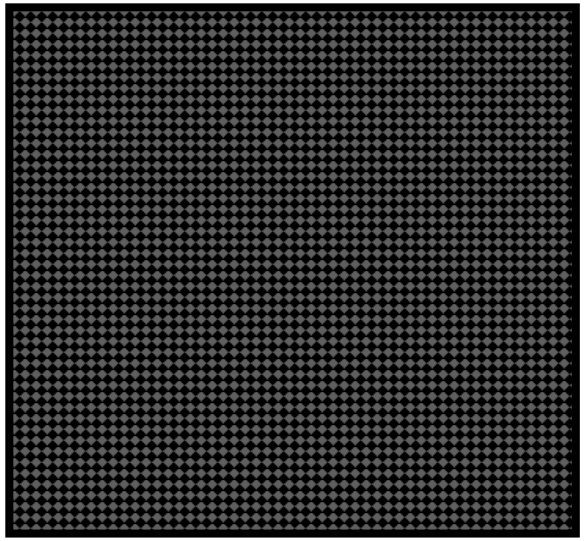


Floorplanning: Area Utilization

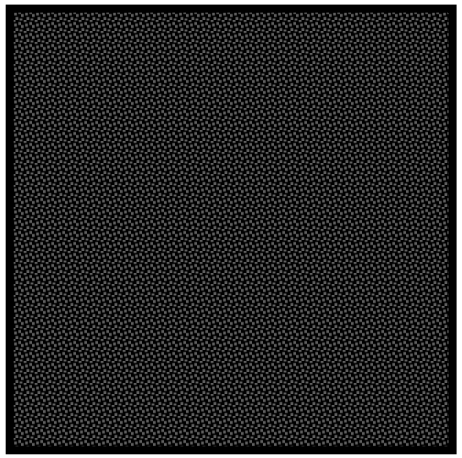
- Utilization dictates how densely user wants cells to be placed within the block
  - The remaining space is needed for ease of routing and newly inserted cells during optimization.

Increasing utilization will reduce the core area
(Default Utilization is 0.7)

$$Utilization = \frac{\sum cellArea}{CoreArea}$$



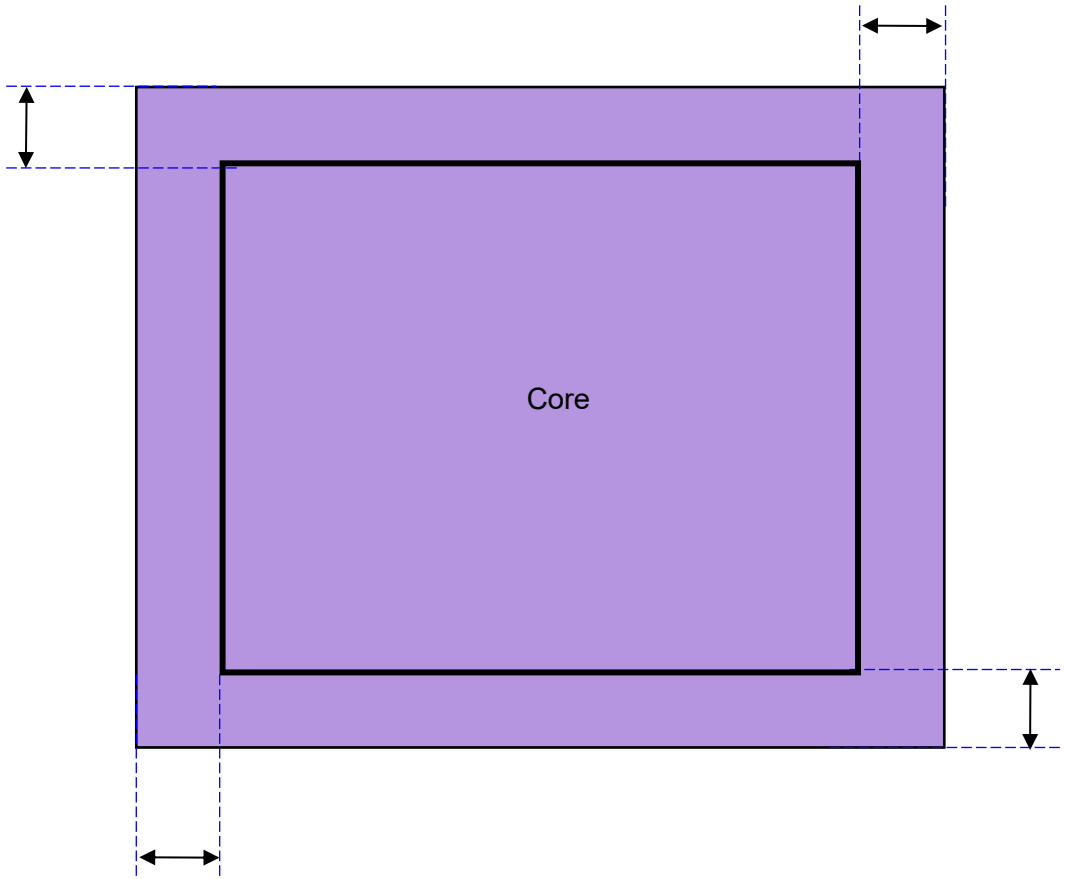
core_utilization 0.6



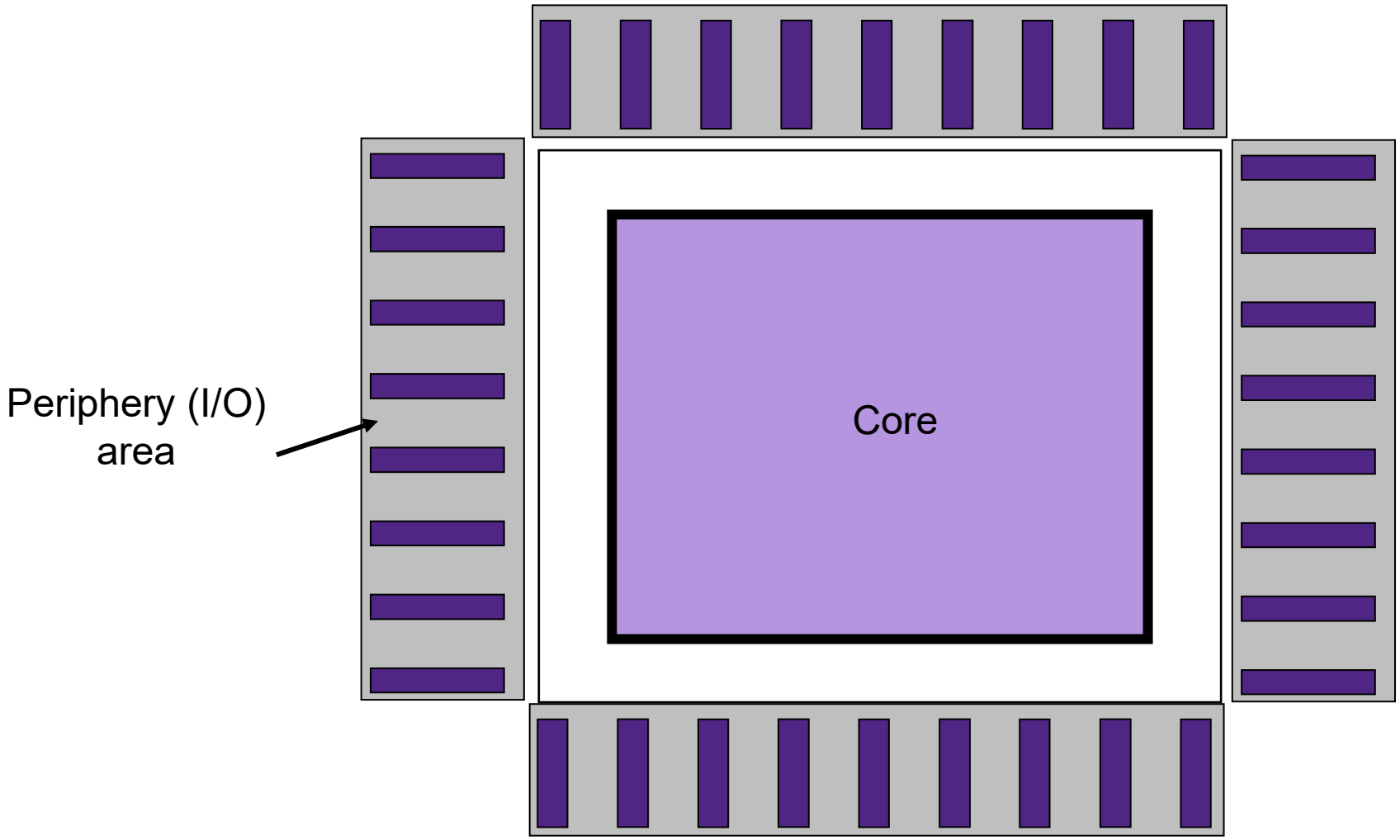
core_utilization 0.85

Floorplanning: Space for Power Rings

- A fixed space is required to be available around core for power supply rings

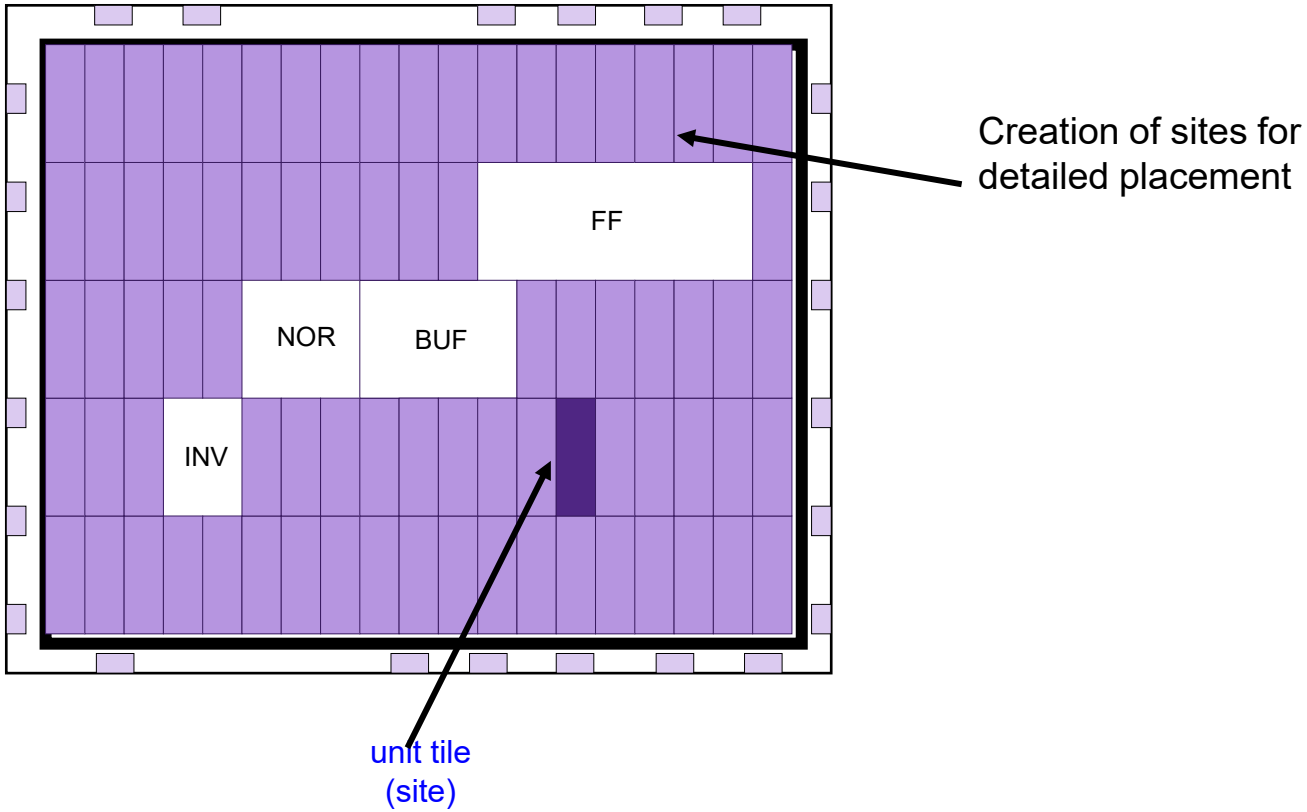


Floorplanning: I/O Placement

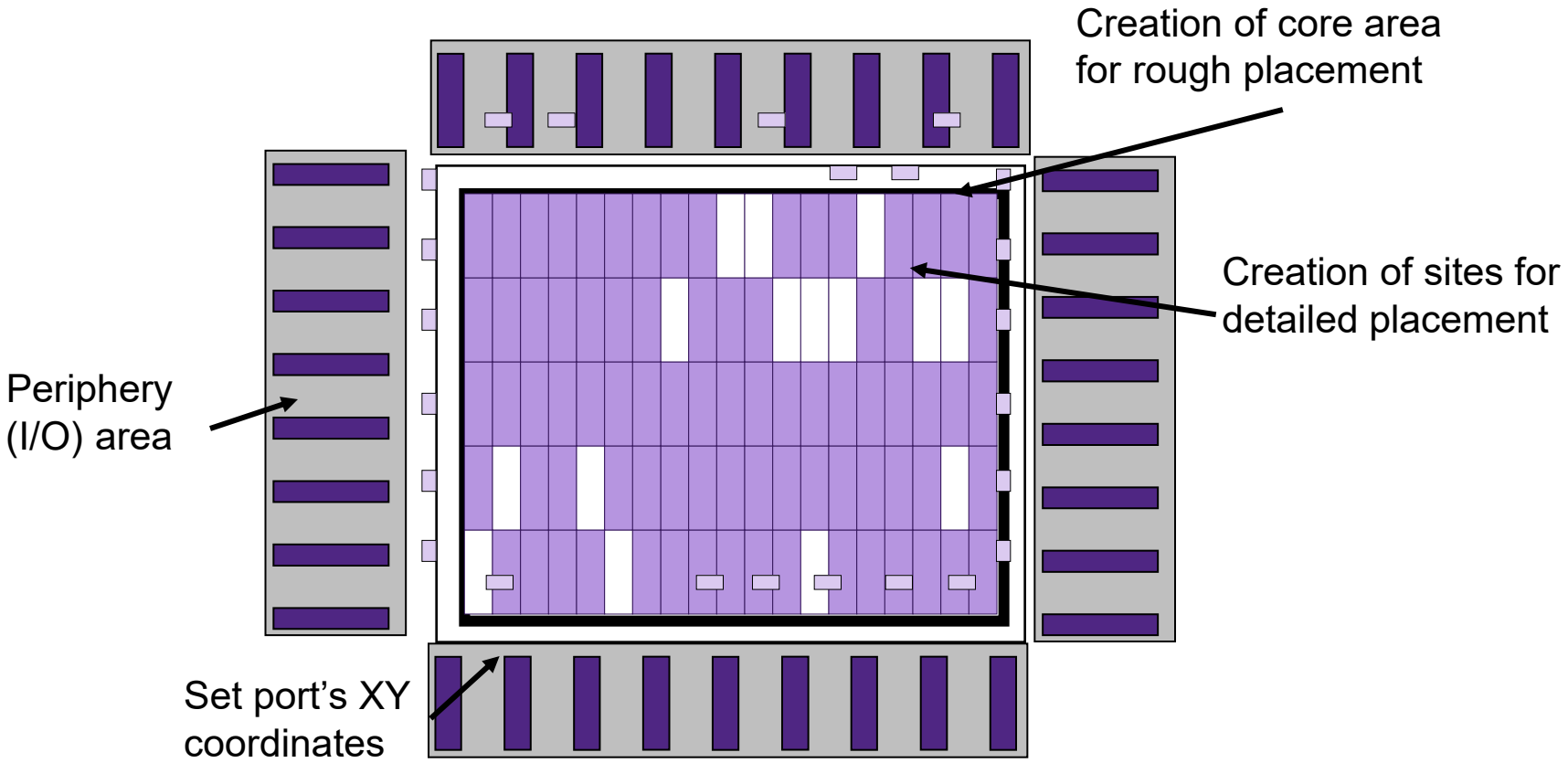


Floorplanning: Creation of Site Rows

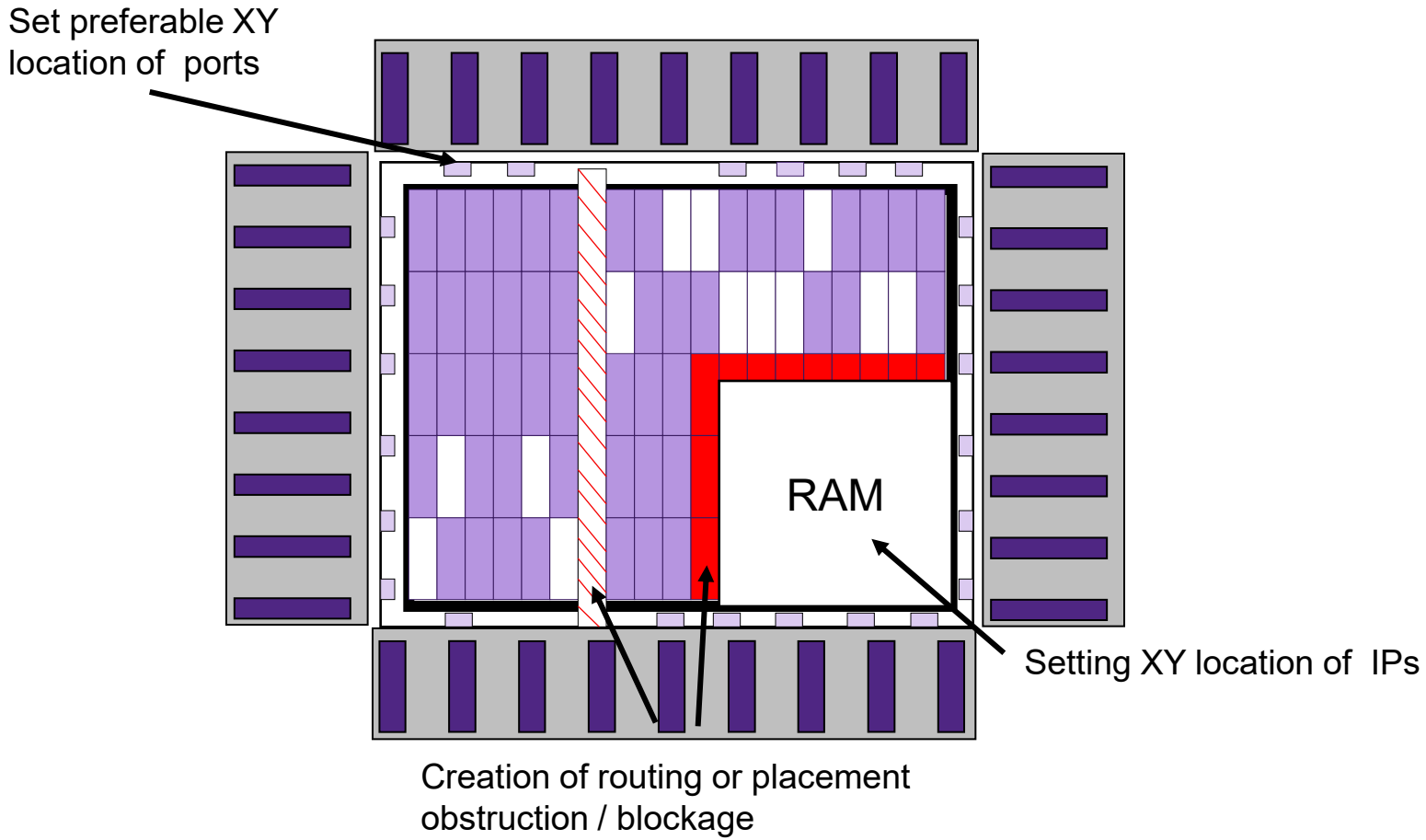
- Placement requires grid in which cells will be placed
- Floorplanning uses ‘unit tile’ cell to build this grid
  - It is defined by a library developer and library cells are designed to be multiple of unit tile



Floorplanning: Required Actions



Floorplanning: Possibilities



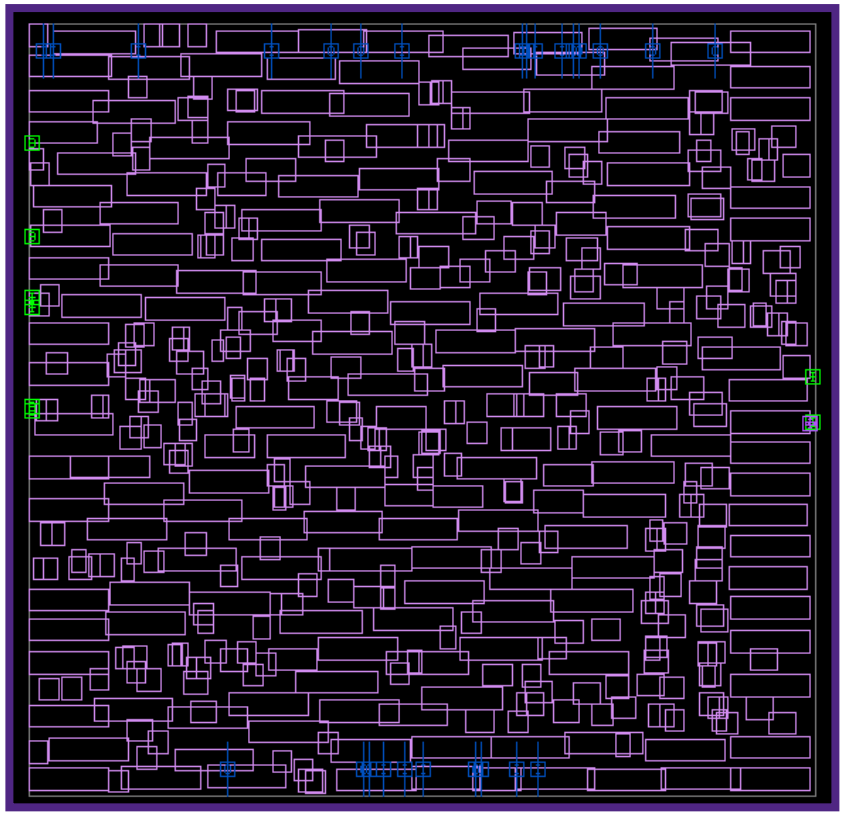
Agenda

- Floorplan Overview
- ***Floorplan Creation***
- Power/Ground Network Creation
- IO cell placement

Automatic Floorplan

- If there is no available floorplan, it will be created automatically as physical synthesis includes placement, which requires floorplan.
- Automatic floorplan is created during compile_fusion flow at logic_opto stage. Default block utilization is 0.7.

  - Automatic floorplan performs:
    - Creation of design's die area, site rows and routing tracks.
    - Macro blocks' placement.
    - Block's shaping.
    - Pins' and I/O cells' placement.



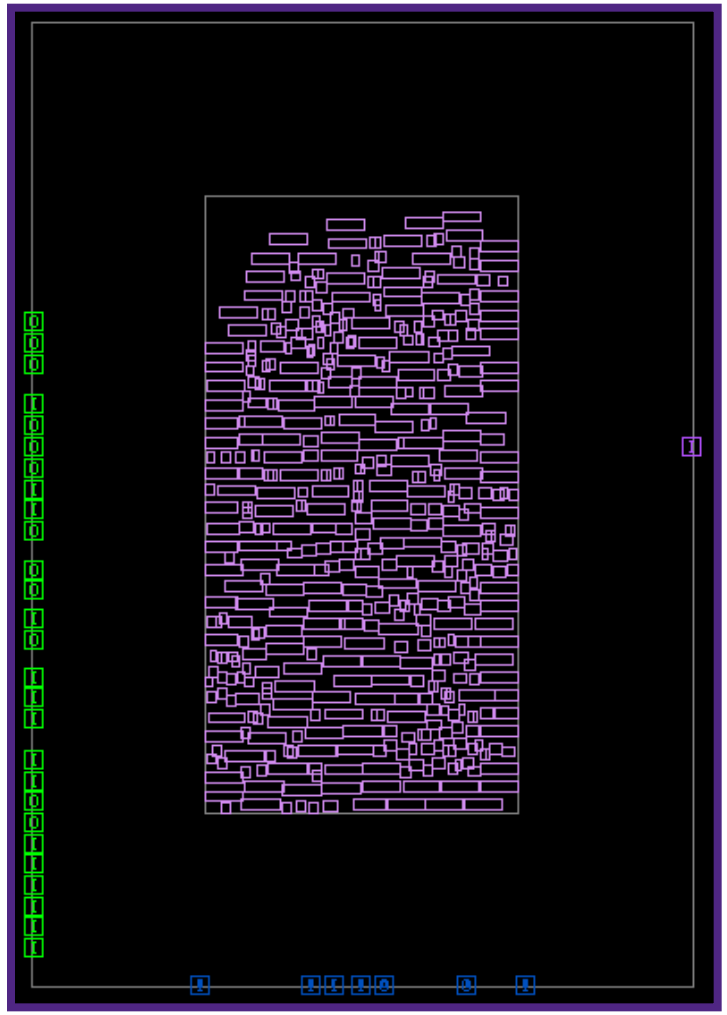
Constraints for Automatic Floorplan

- Constraints for automatic floorplan creation can be defined by using tool's commands and application options.

```
set_auto_floorplan_constraints \
  -core_utilization ${core_util} \
  -core_offset ${core_offset} \
  -shape R \
  -side_ratio {1 2} \
        …
```

- To report constraints for automatic floorplan use:

```
report_auto_floorplan_constraints
```



Constraints for Pin Placement

Pin placement during automatic floorplan creation also can be controlled.

- To set pin constraints use:
  - **set_block_pin_constraints** **-self** **-allowed_layers** {M2 M3} **-sides** {1 4} …
- To set individual pin constraints use:
  - **set_individual_pin_constraints** **-ports** [**get_ports** ${port_name}] **-allowed_layers** M4
- To report pin constraints use:
  - **report_block_pin_constraints** **-self**

Manual Floorplan Creation

- Floorplan can be created from scratch after logic_opto stage to better fit to engineer's needs. To create floorplan from scratch use:
  - **initialize_floorplan -core_utilization** ${core_util} **-core_offset** ${core_offset} …
- Constraints for pin, I/O cells, Macro blocks placement also can be defined for automatic placement.
- Physical only cells, like TAP or Boundary cells, can be inserted after block shape/size is defined.
- Created floorplan can be saved for later use.

GUI Floorplan Initialization Window

Block shape control

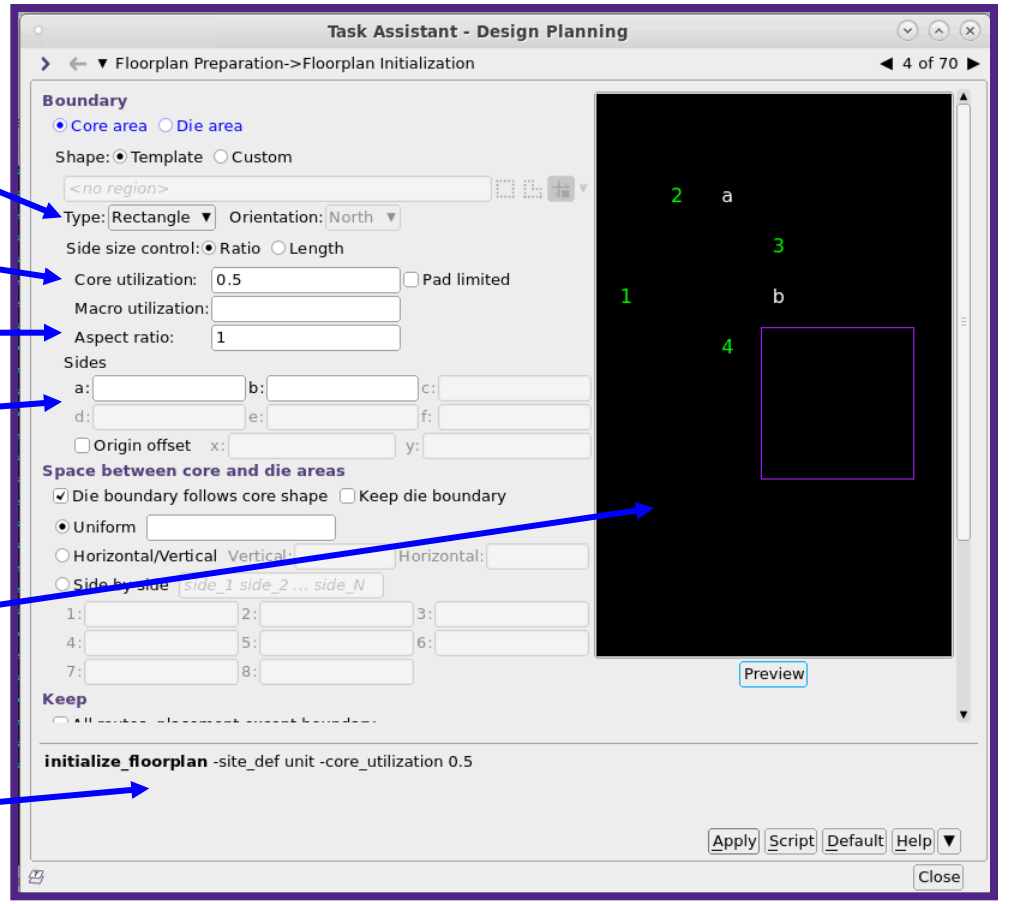
Core utilization control

Aspect ratio control

Sides' size control

Block's preview

Script with configured parameters



Task -> Design Planning -> Floorplan Initialization

Saving and Reading the Floorplan

- You can save the created floorplan in script:
  - **write_floorplan**

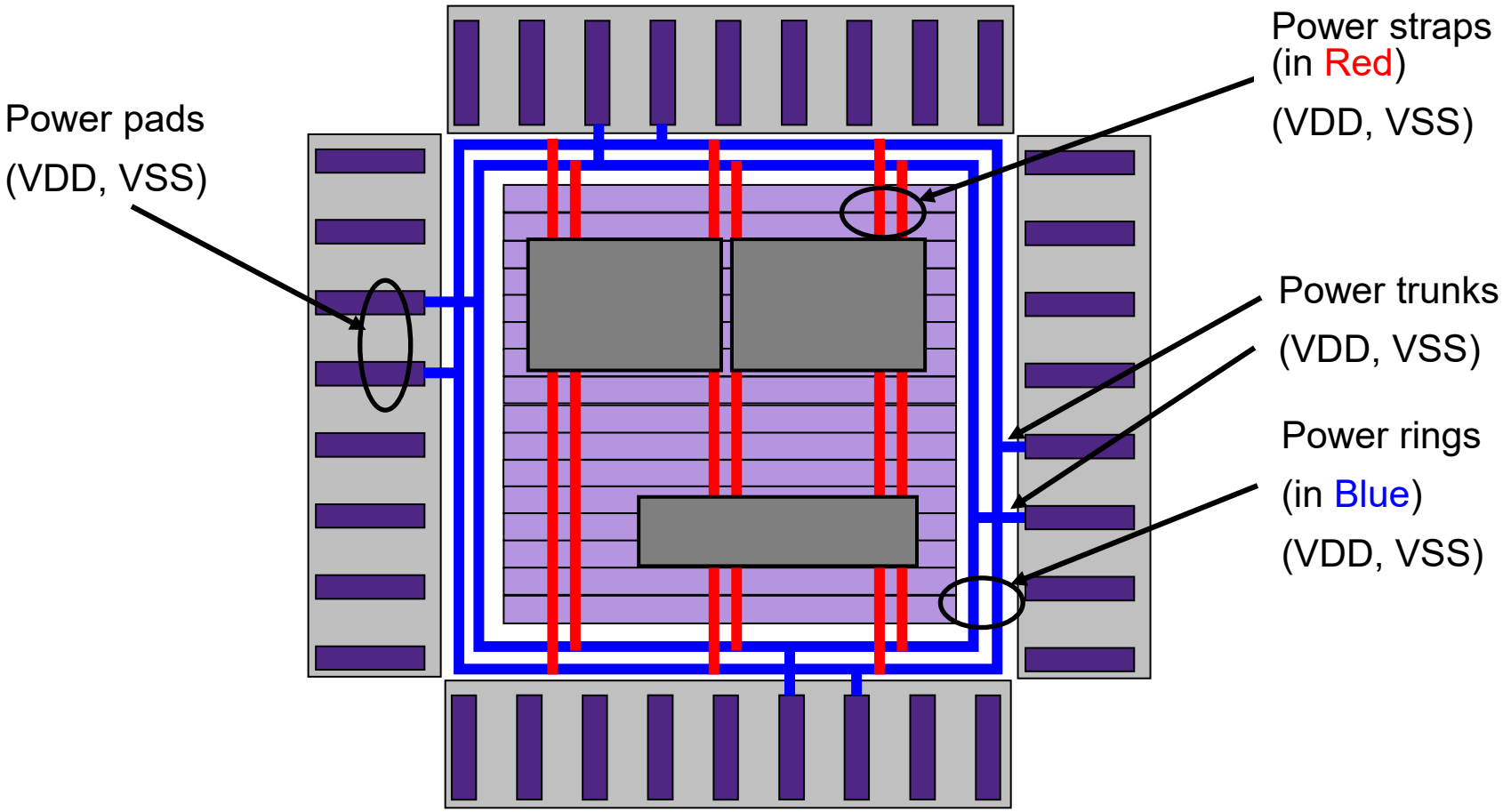
  or in Design Exchange Format (DEF) file.
  - **write_def** floorplan.def
- Created floorplan can be read in the tool by sourcing the script:
  - **source** floorplan.tcl

  or by reading in the DEF file:
  - **read_def** floorplan.def

Agenda

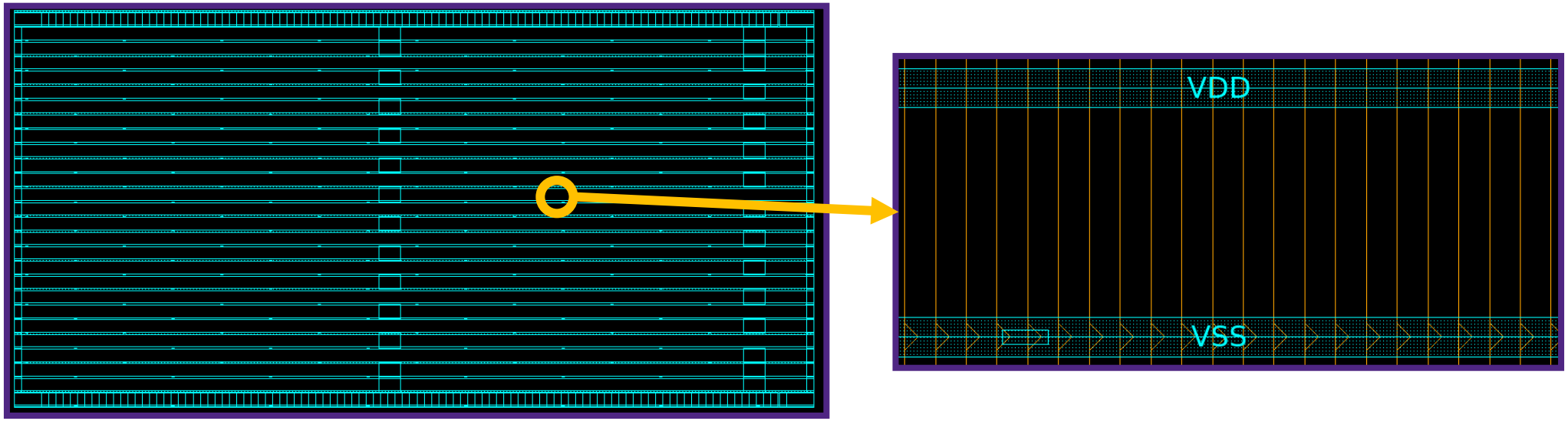
- Floorplan Overview
- Floorplan Creation
- ***Power/Ground Network Creation***
- IO cell placement

Power Network



Standard Cell's PG Rails

- Rails are needed to connect supply pins of standard cells to the Power/Ground Network of the block or chip.
- PG Rails are created with pitch equal to unit tile's height.
- The width of PG Rails are defined by library developer.

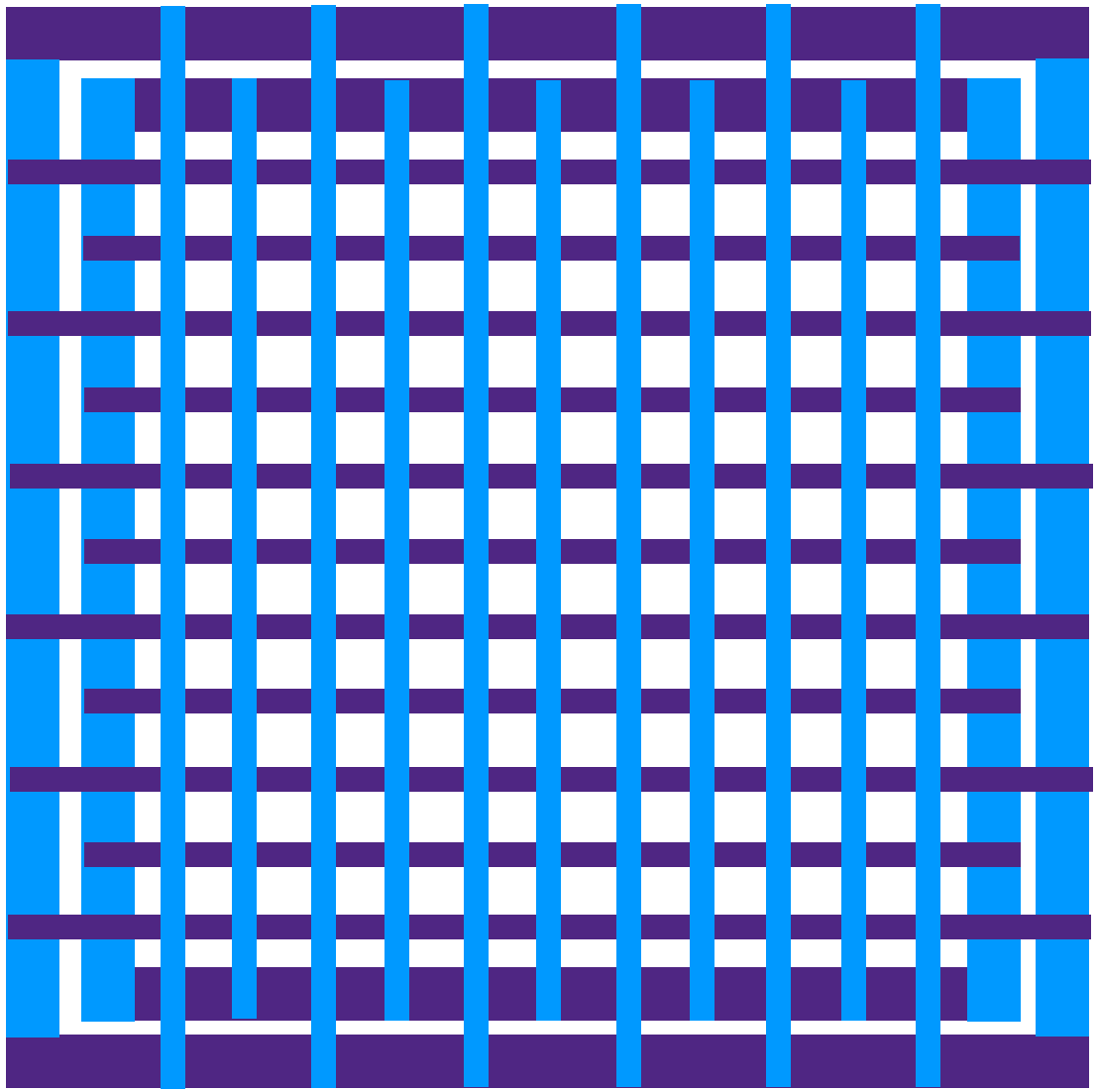


Power Grid Planning

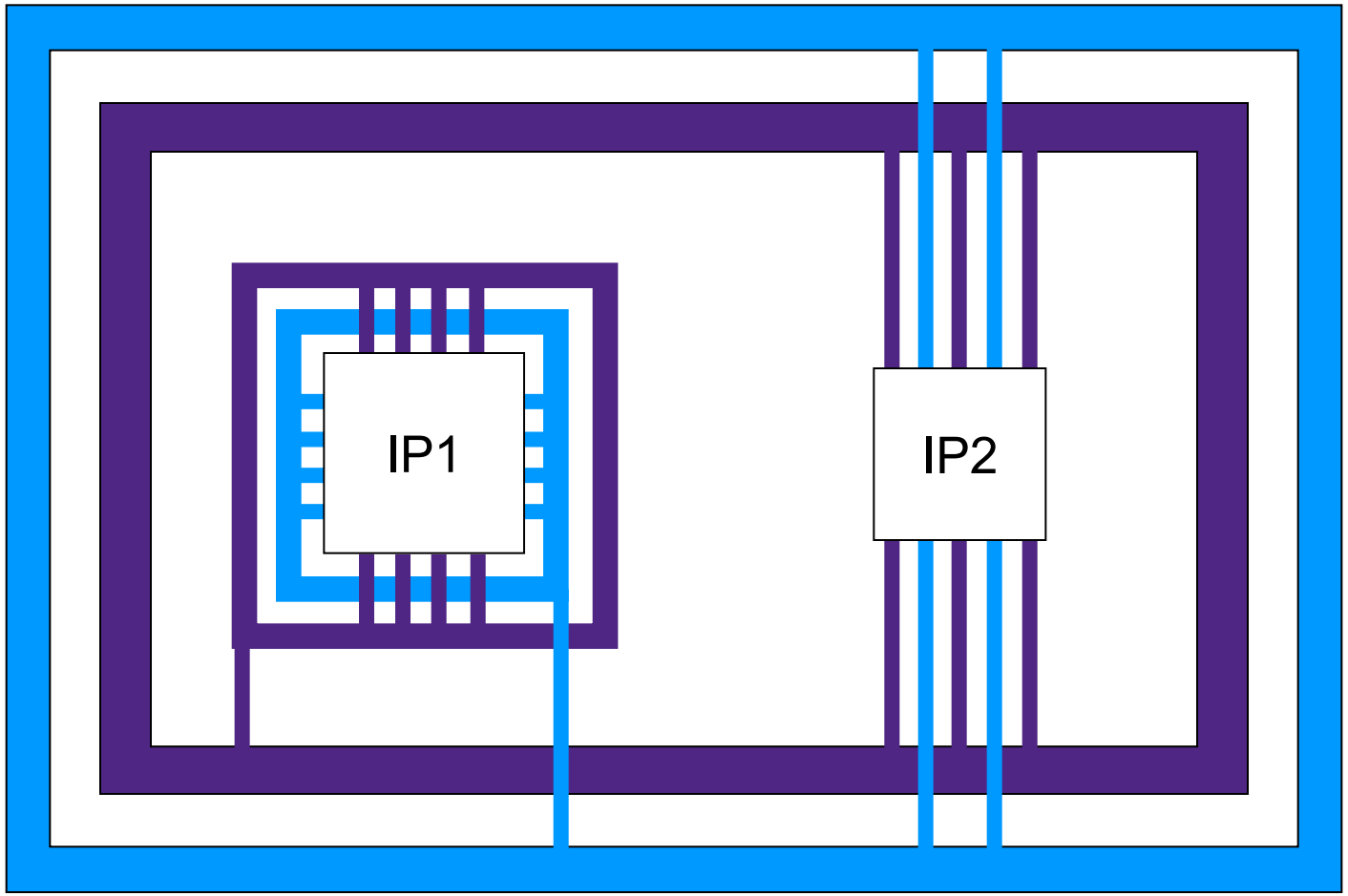
- Steps in Power Grid Planning
  - Determine the number of power pads
  - Determine which metal layers will be used for power/ground network creation
  - Define the width of the top-level power/ground buses
  - Determine the structure for block and macro-level power/ground network
  - Power Network Analysis (PNA)

Top-level Power Network

- User can specify
  - Number of straps
  - Width of straps
  - Width of ring
  - Layers



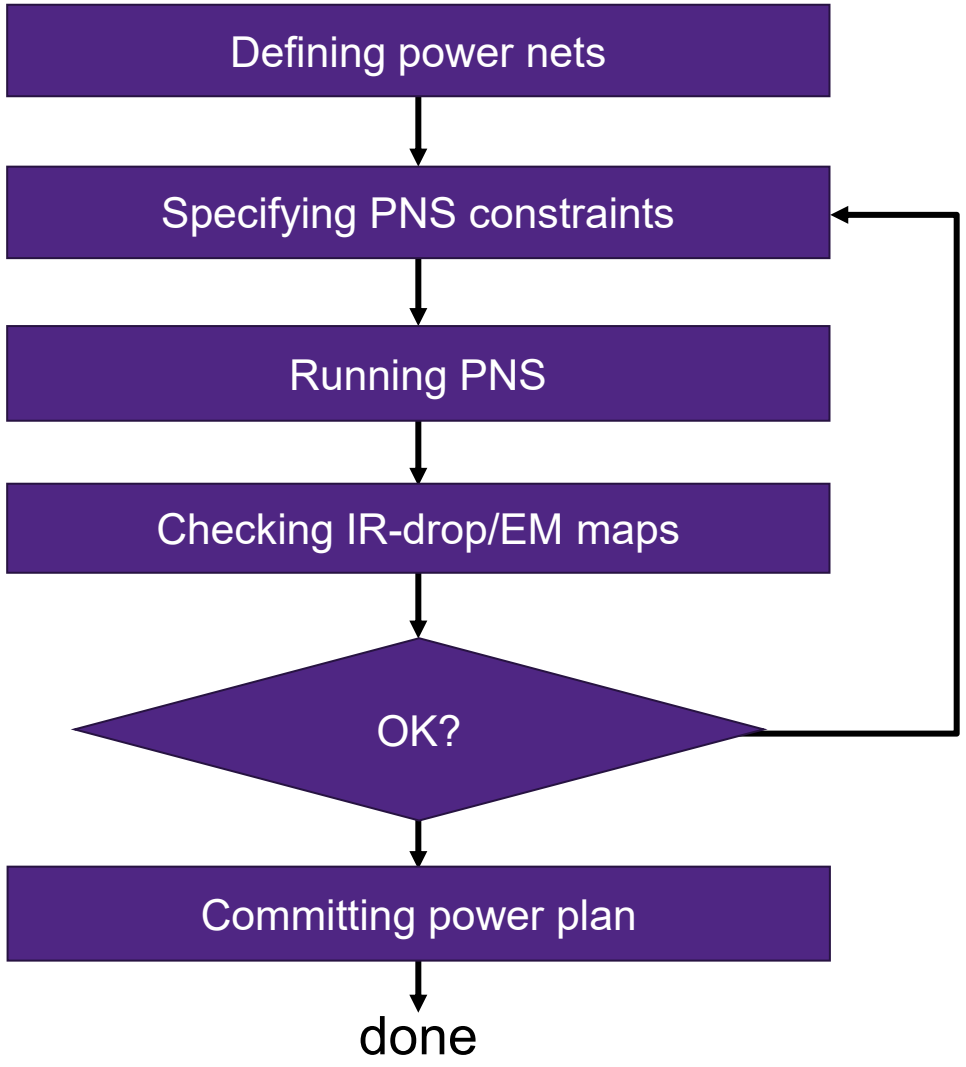
Block Level Power Network



Power Network Synthesis

- PNS
  - Power network creation
    - Definition of the topology
    - Calculation of the width and the number of power straps to meet IR constraints
    - Performance of detail connections and placement of vias
    - Validation with PNA
  - PNS automates these tasks

PNS Flow



Constraint-based PNS speeds up power network creation

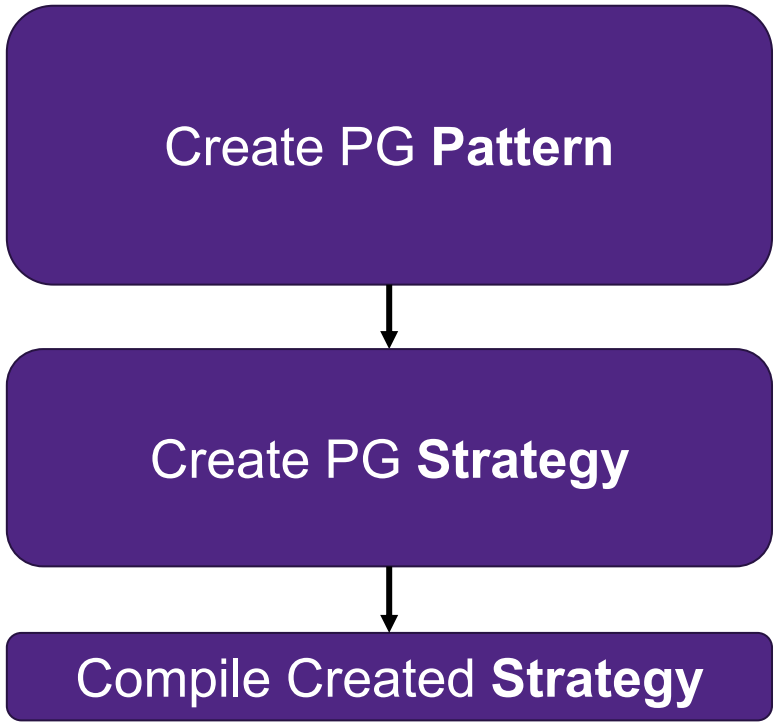
Power Network Analysis (PNA) can be done both before and after the power plan is committed

PNS Requirements

- Requirements for PNS
  - Power nets must be connected logically to all corresponding standard cell and macro power ports
  - Power pads need to be defined
    - Pad master or instances
      May be supplemented with virtual pads

PG Network's Objects Creation Flow

Script example for **PG Rail** creation.



- **create_pg_std_cell_conn_pattern** M1_rail \
  -layers {M1} \
  -rail_width {@wtop @wbottom} \
  -parameters {wtop wbottom}

- **set_pg_strategy** M1_rail_strategy **-core**
  -pattern {{name: M1_rail} {nets: VDD VSS}
  {parameters: {0.094 0.094}}}

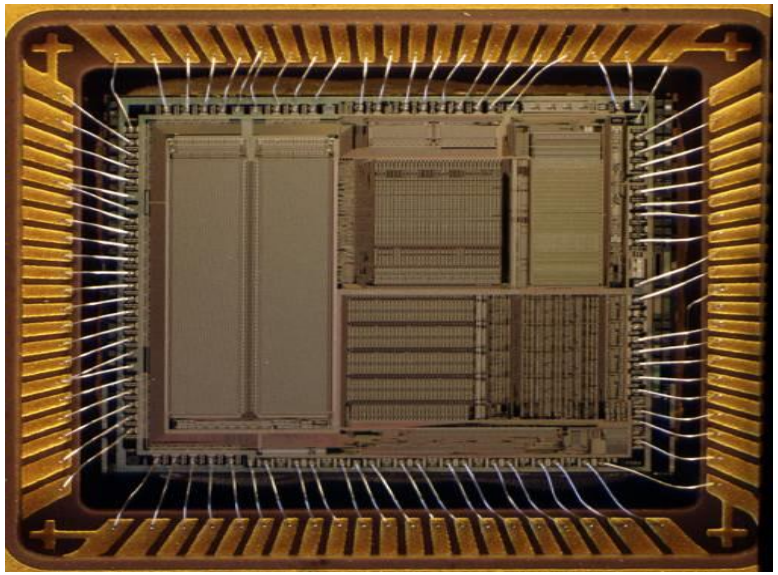
- **compile_pg -strategies** M1_rail_strategy

Agenda

- Floorplan Overview
- Floorplan Creation
- Power/Ground Network Creation
- ***IO cell placement***

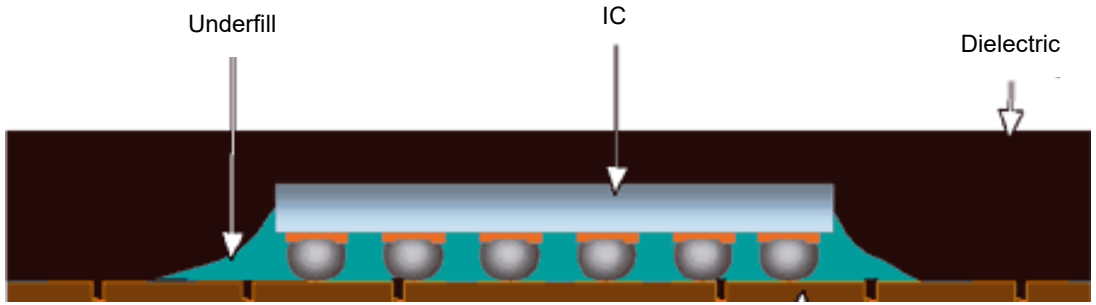
Packages Types

Wire Bond



http://ffden-2.phys.uaf.edu/212_spring2005.web.dir/george_walker/uses.htm

Flip-Chip

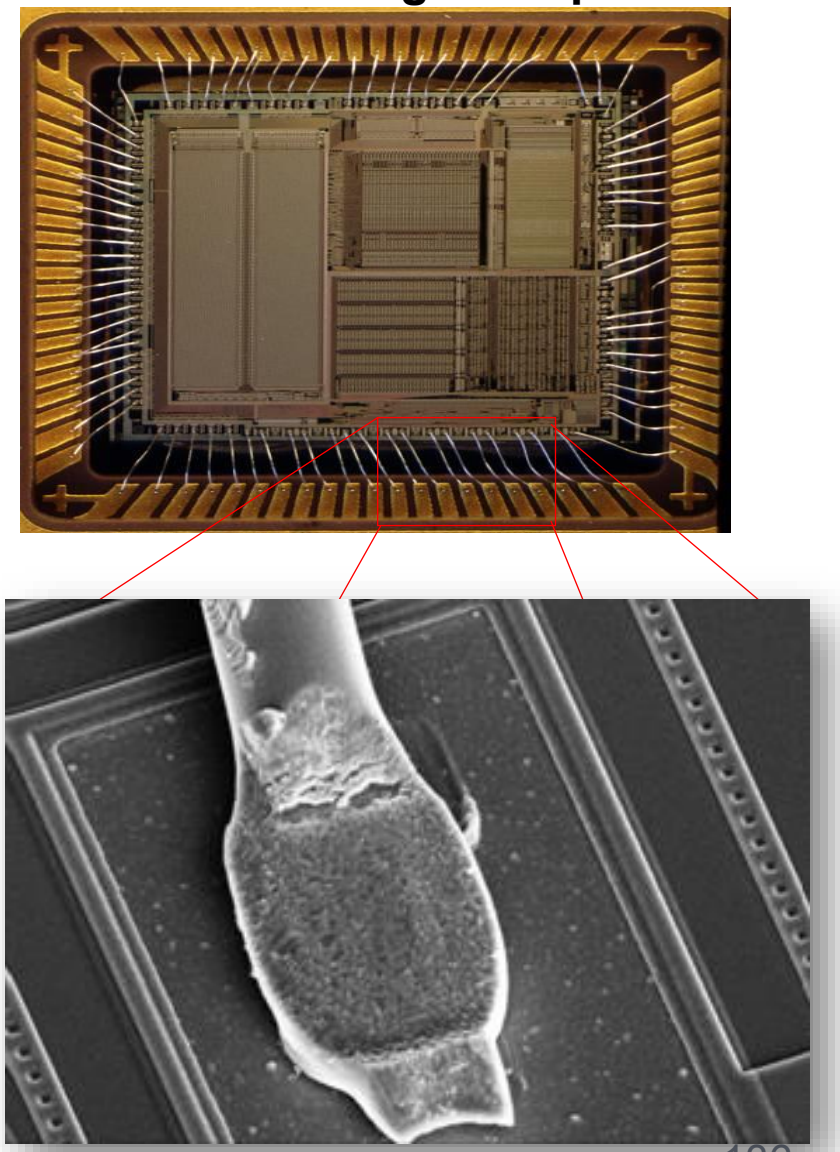


http://blog.daum.net/dourira/6824688

Wire Bond

- Most widespread method
- IC is connected with package pins with thin wires

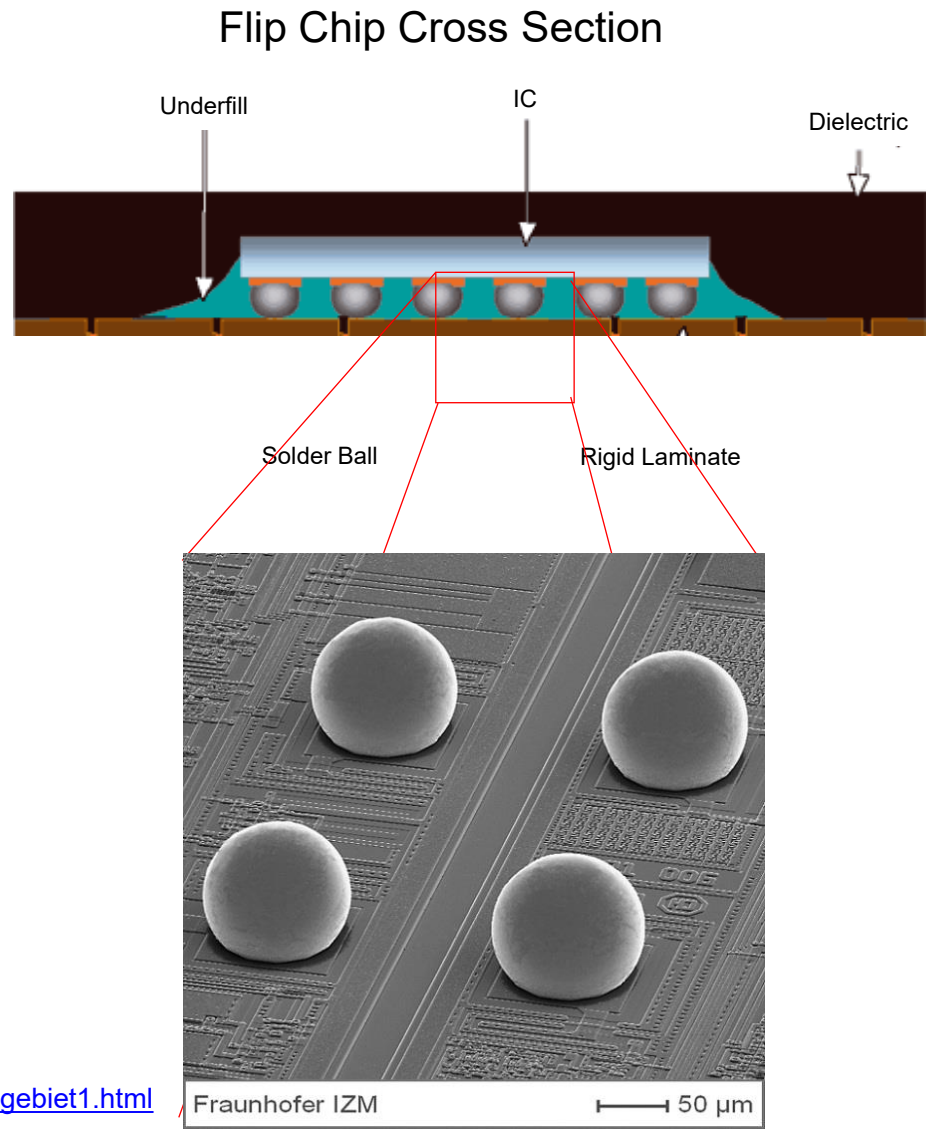
Wire Bonding Example



http://www.era.co.uk/case-studies/improving-the-reliability-of-chip-on-board-assembly/

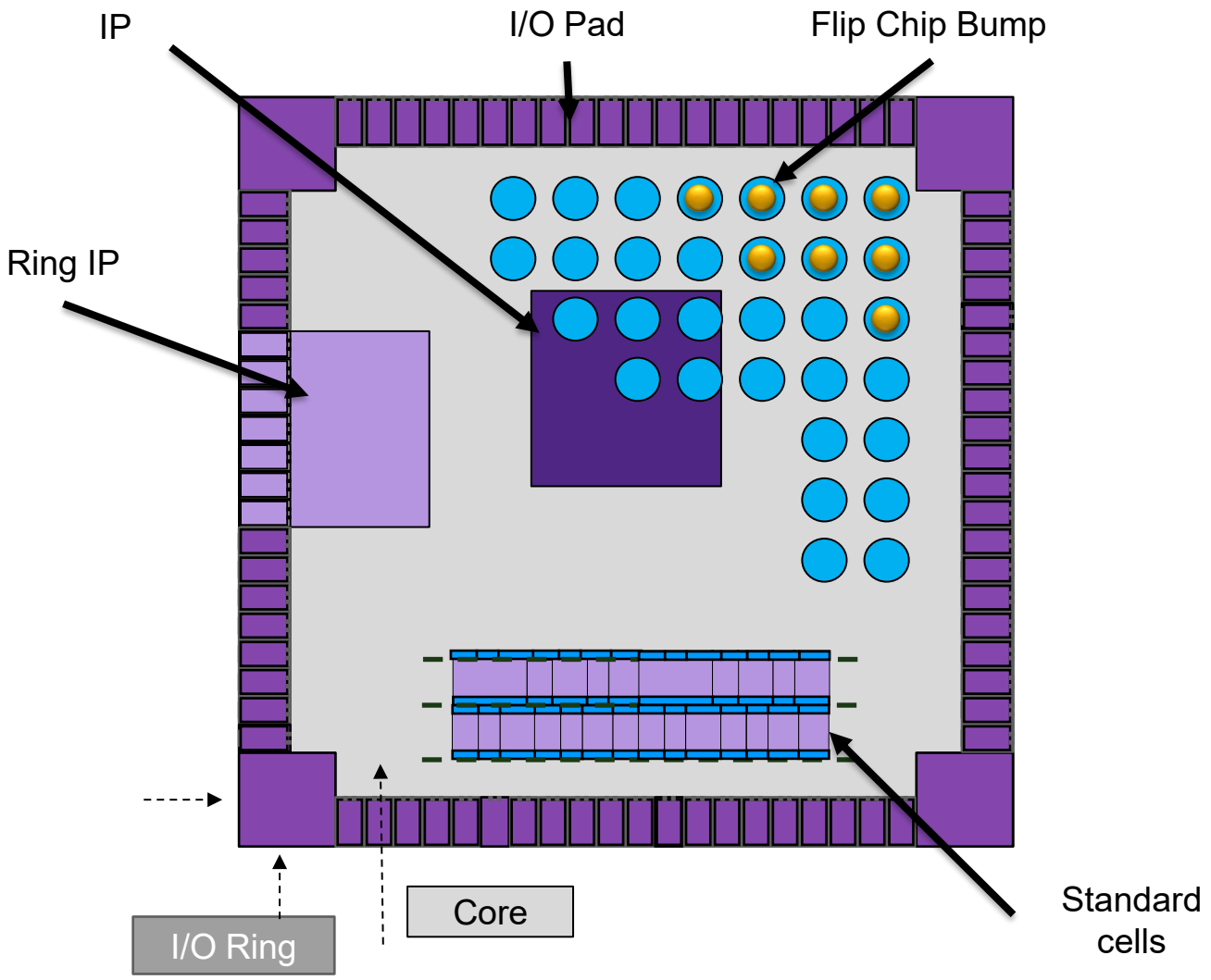
Flip-Chip

- Modern method of I/O connection
- IC is connected with package using solder balls

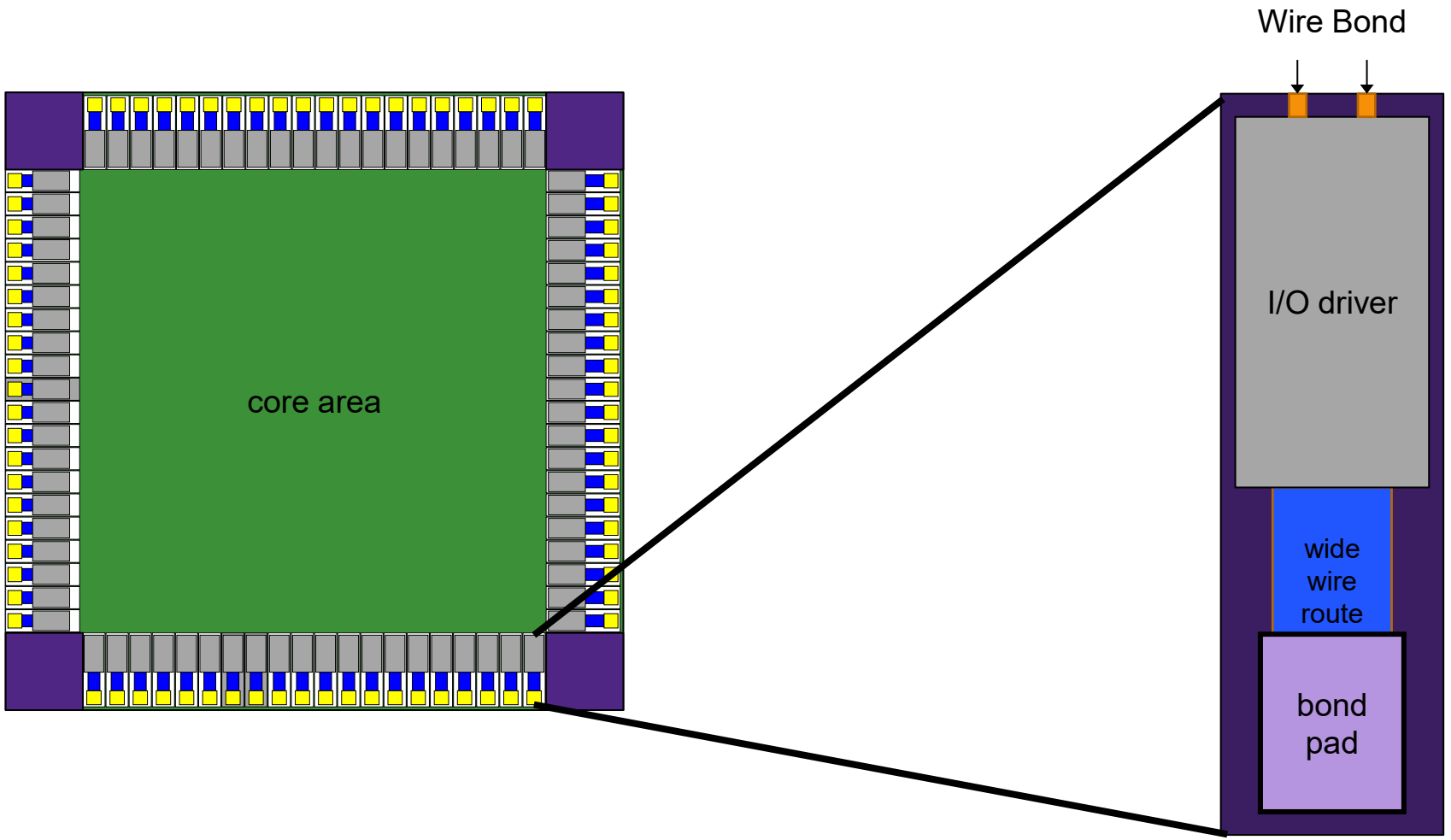


http://www.izm.fraunhofer.de/en/abteilungen/high_density_interconnectwaferlevelpackaging/arbeitsgebiete/arbeitsgebiet1.html

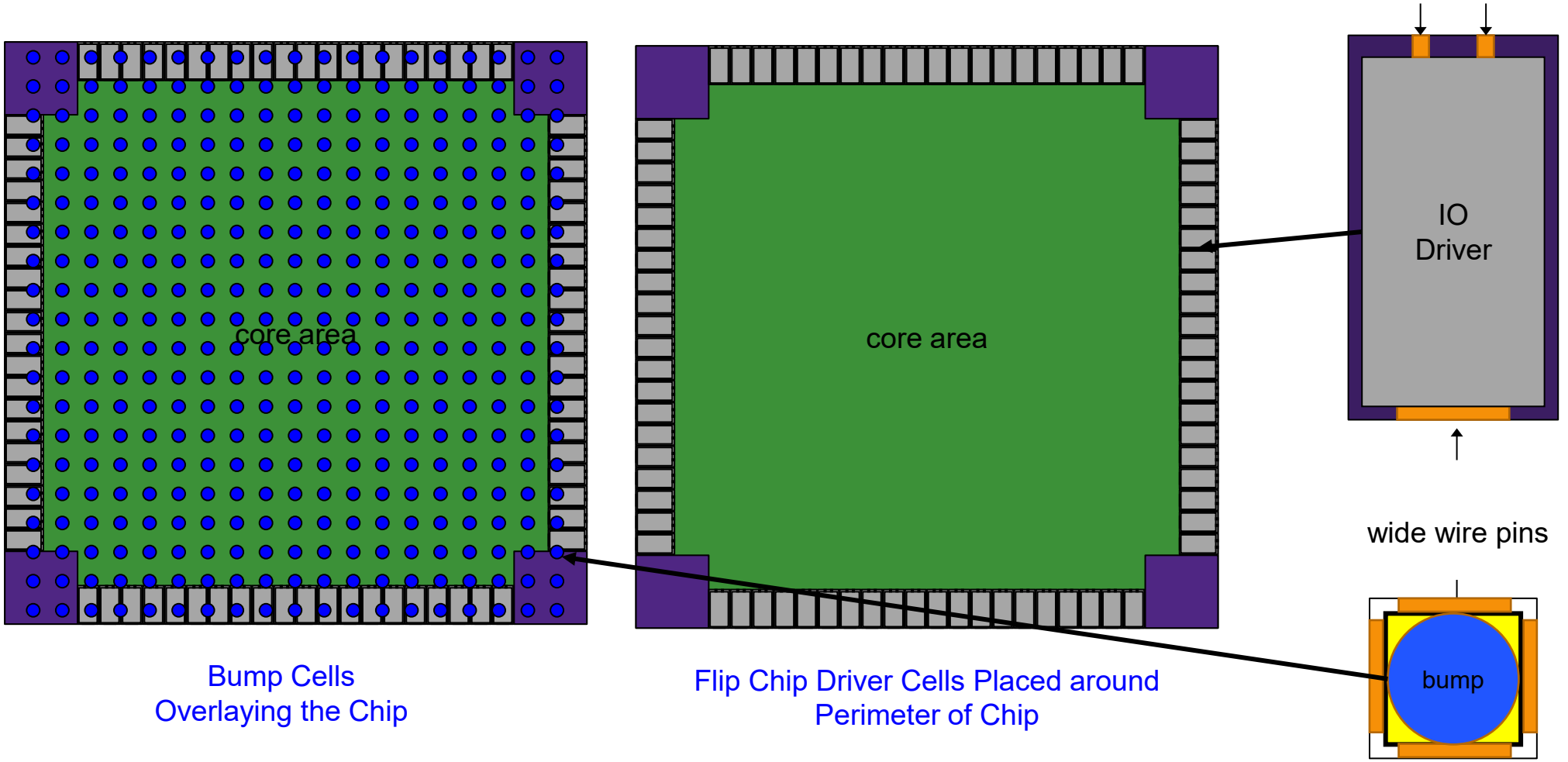
IO Example



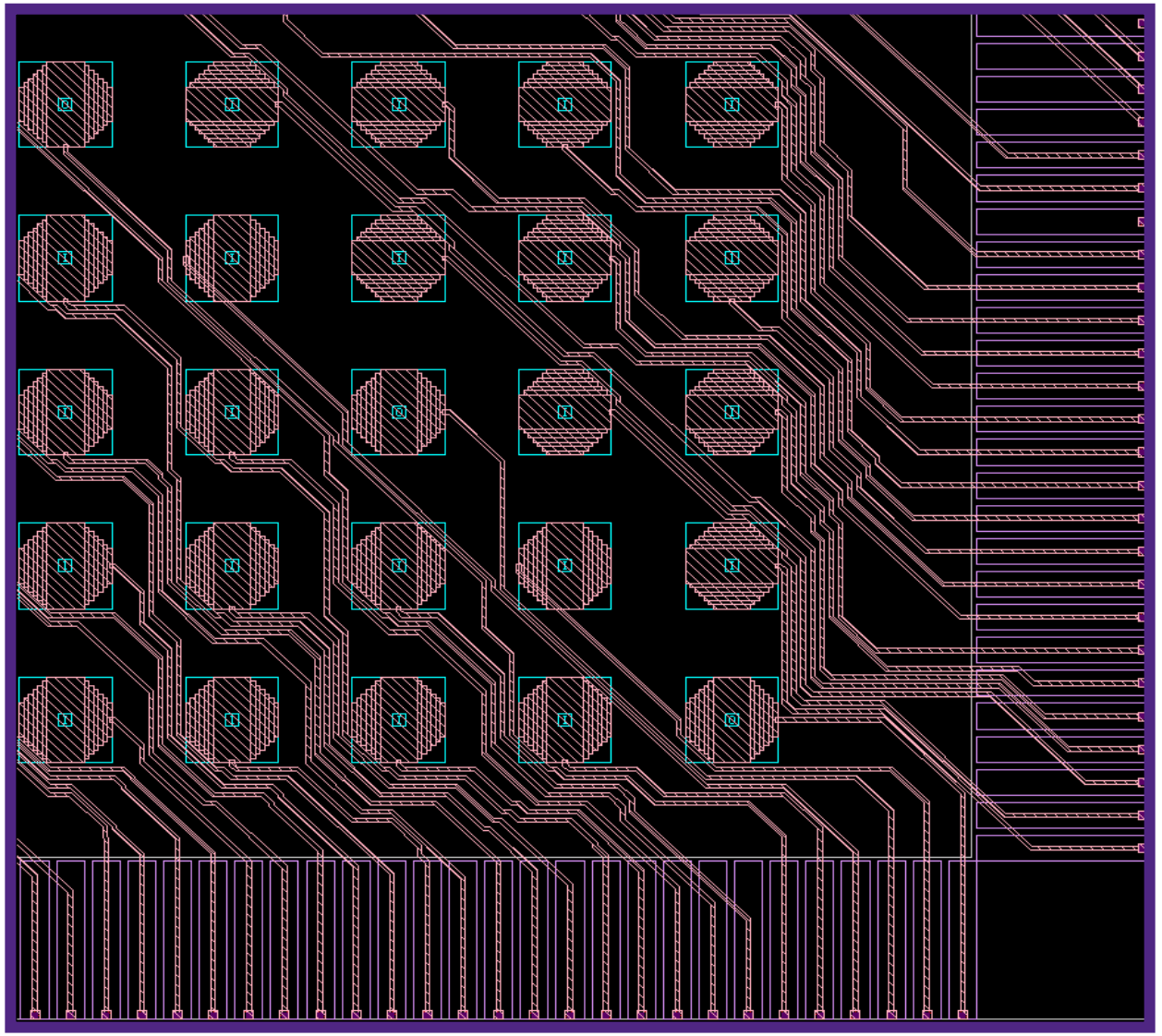
Wire Bond Placement



Flip Chip Placement



Routed Flip-Chip Example



Course Overview

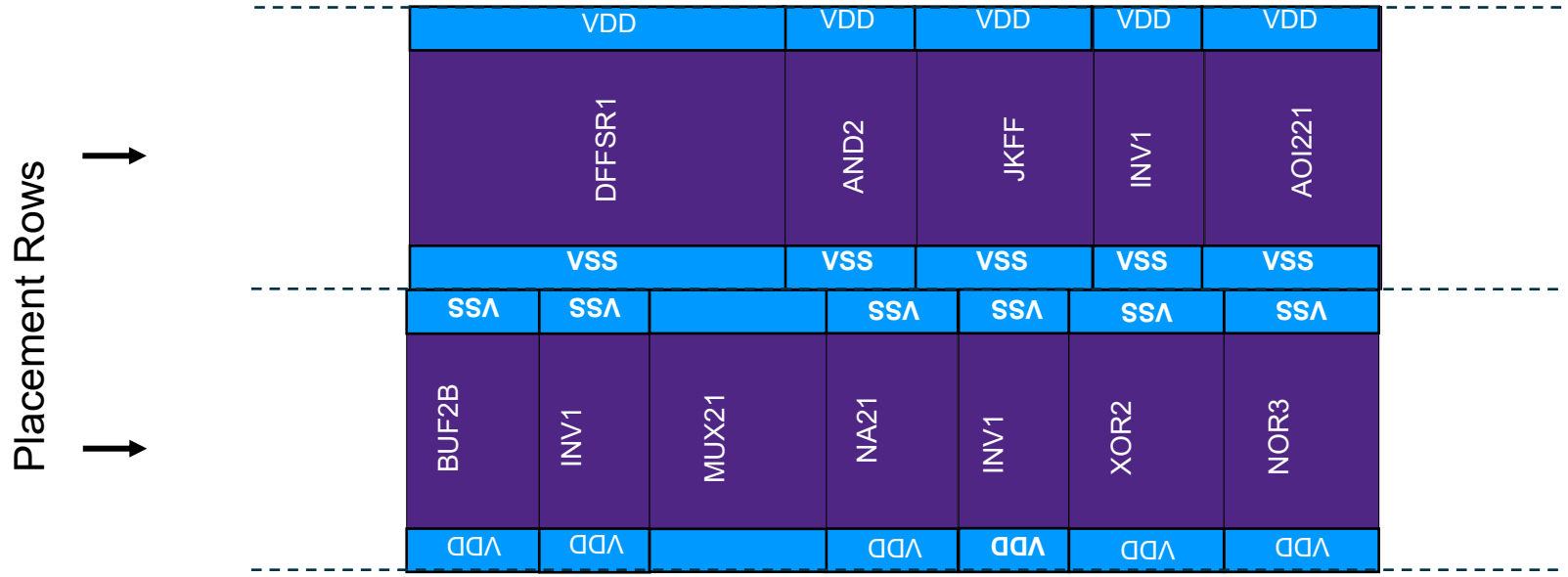
- Introduction
- Timing and Area Constraints
- Multi-Corner Multi-Mode and Mapping
- Application Options and Compile Flow
- Floorplan
- Placement
- Clock Tree Synthesis (not covered)
- Routing (not covered)
- Signoff (not covered)

Agenda

- ***Standard cell placement***
- Congestion-driven placement
- Timing-driven placement
- Macro block placement and constraints

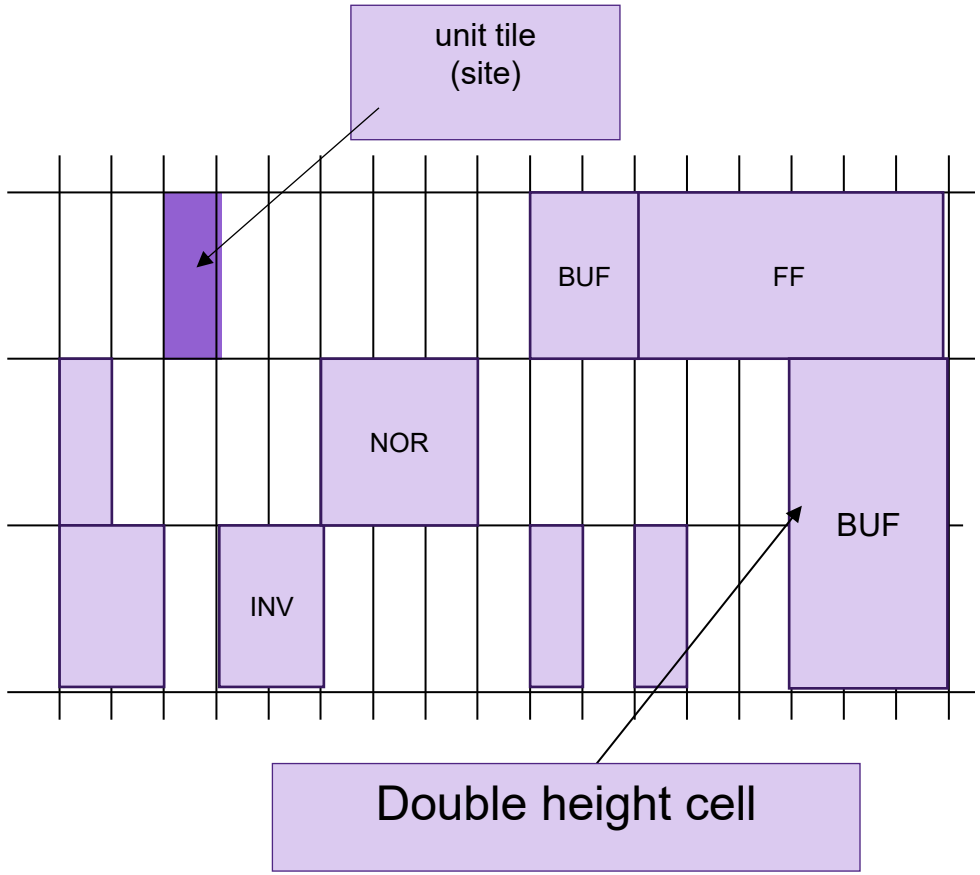
Standard Cell Placement

- Cells are placed in rows, next to each other.
- One cells structure continue previous one.
- Cells on neighbor rows are flipped so that they can share same supply.

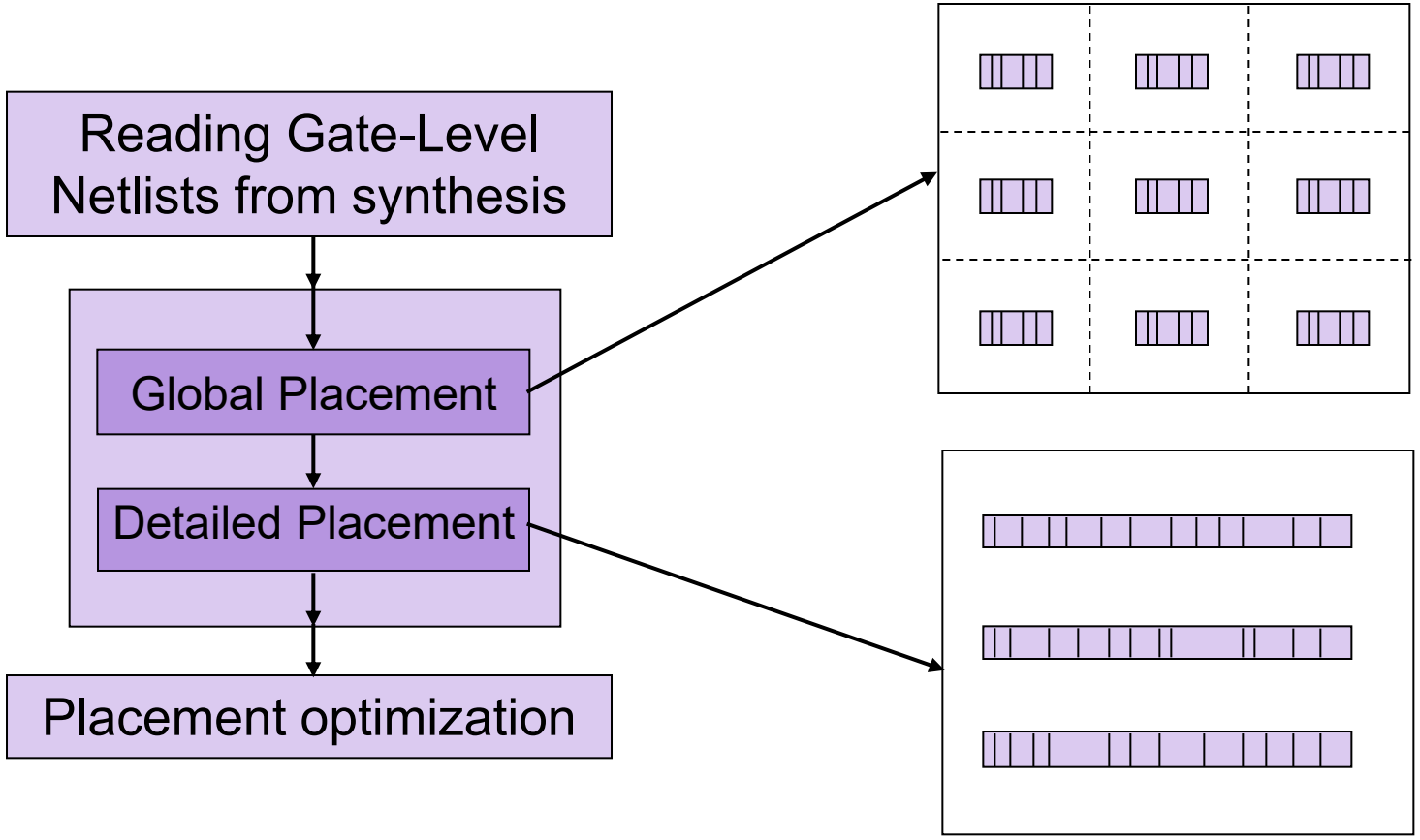


Site Rows

- Placement uses grid in which cells are placed.
- Unit tile is used to build this grid.
  - Unit tile is defined by a library developer.
  - All the cells in the library are designed to be multiple of unit tile.



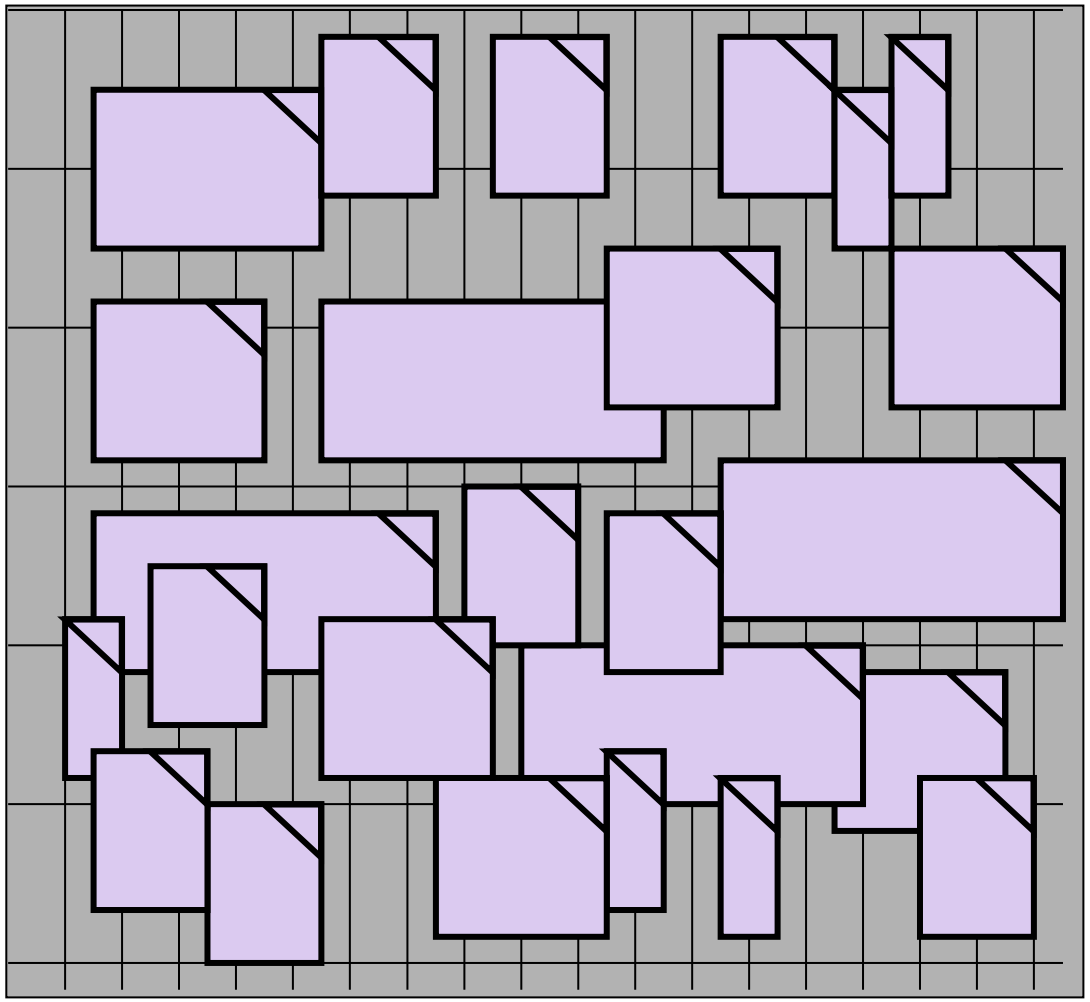
Global and Detailed Placement



Coarse Placement

- Coarse Placement
  - All the cells are placed in the approximate locations but **they are not** legally placed.
  - No logic optimization is done.
  - To create a coarse placement use:

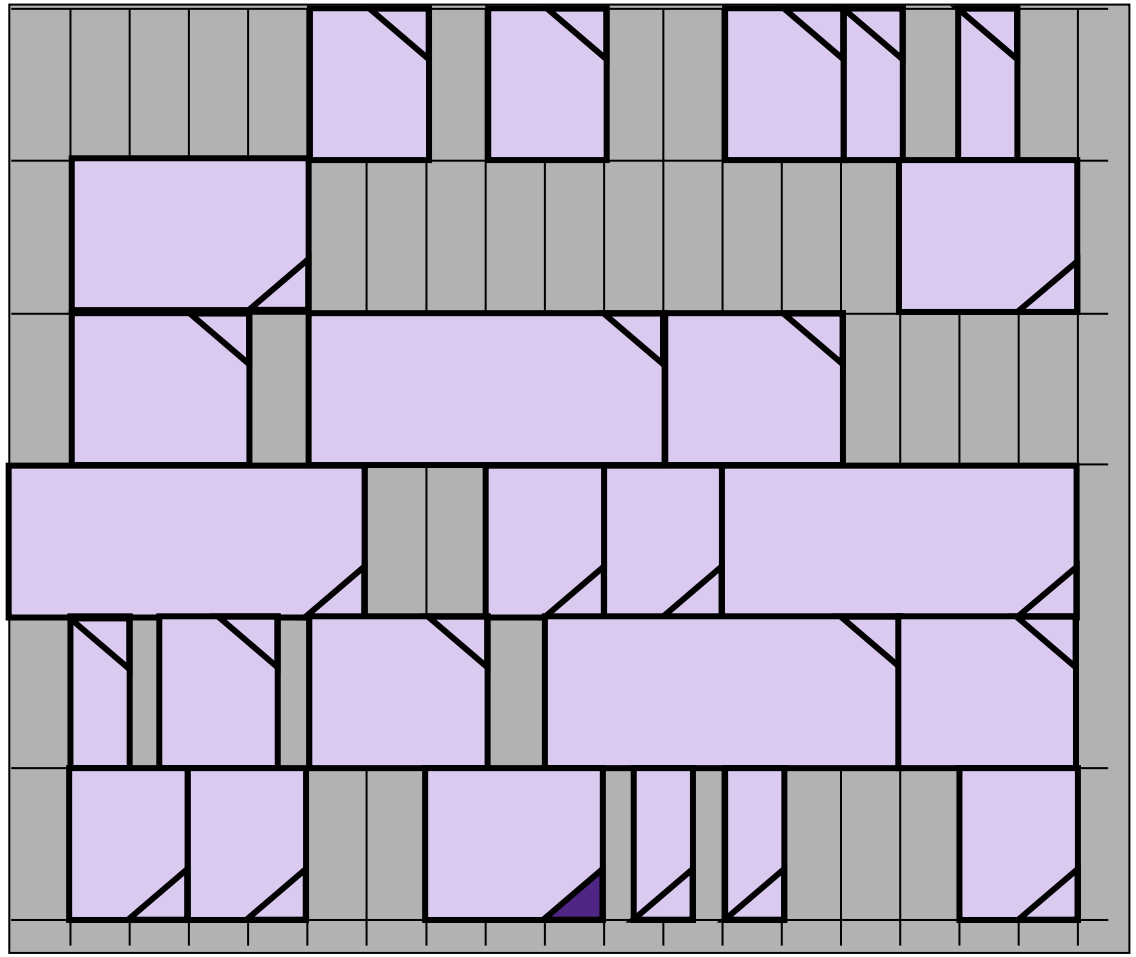
create_placement



Legalize Cell Placement

- During legalization process the cells are moved to their legal locations.
- Ensuring that the final placement is legal before starting next stages of the design process.

- **legalize_placement**



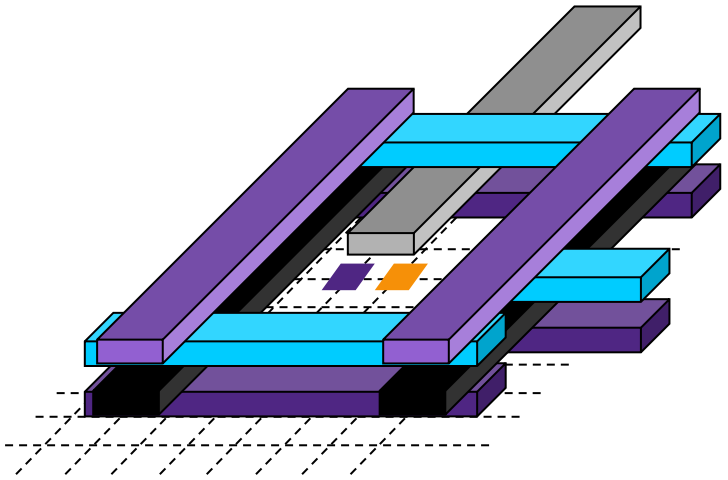
Legal placement of cells is not required for analyzing routing congestion at an early stage.

Agenda

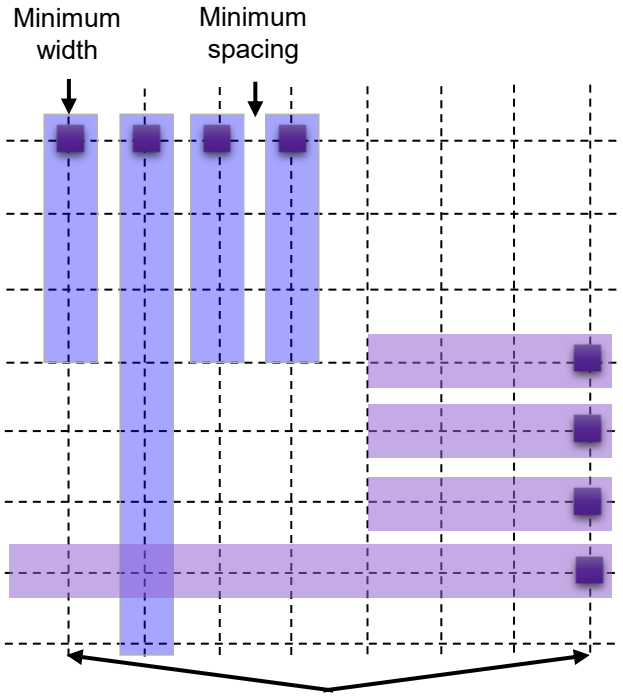
- Standard cell placement
- ***Congestion-driven placement***
- Timing-driven placement
- Macro block placement and constraints

Routing Resources

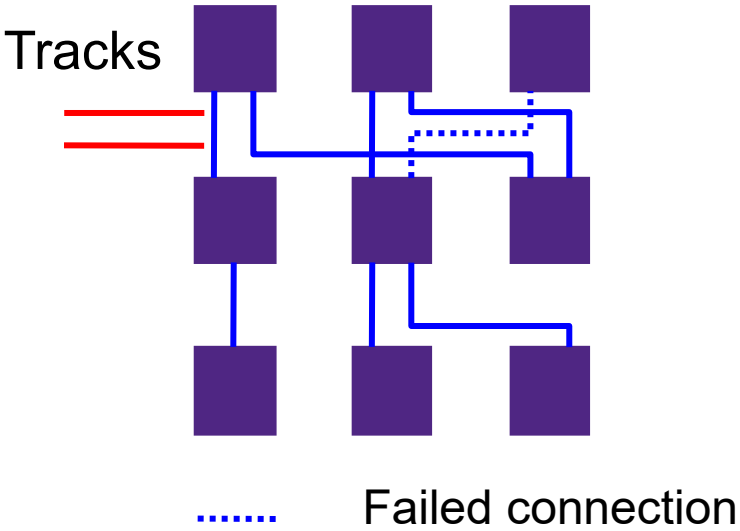
Layers have perpendicular directions



Routing is done on tracks

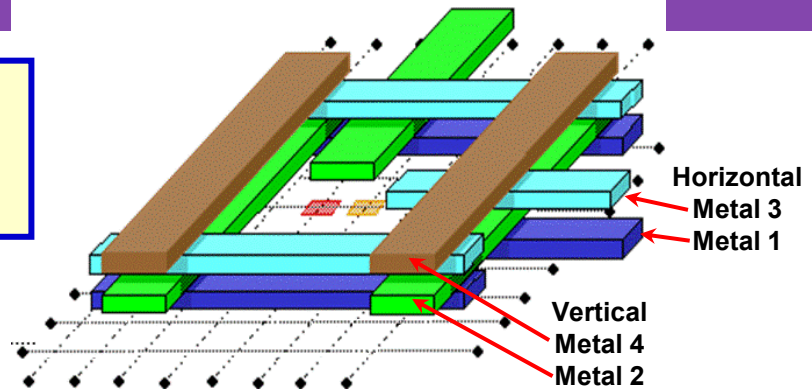


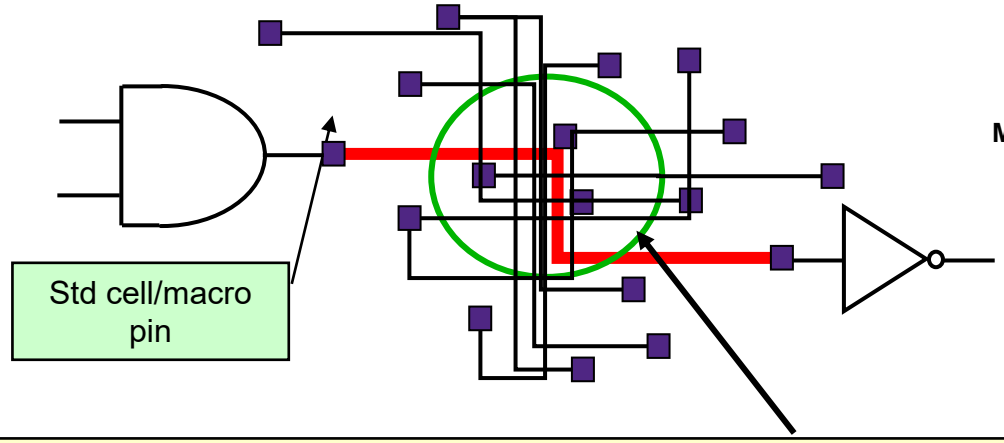
Insufficient number of tracks bring congestion

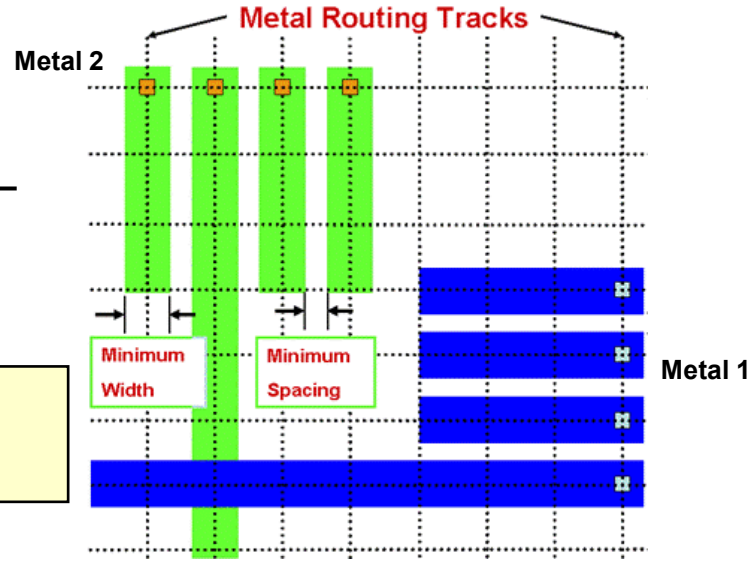


What is Congestion?

It is a measure of a design's "routability", and is analyzed prior to actual routing





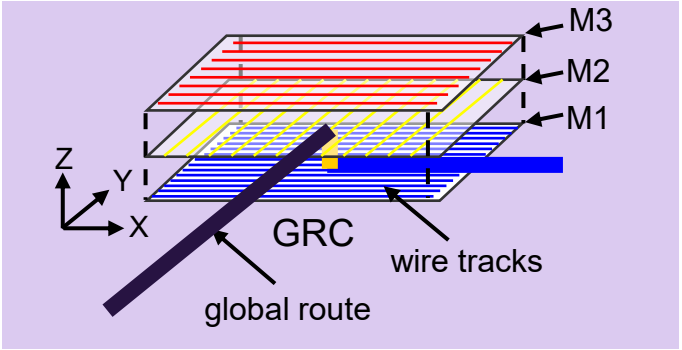


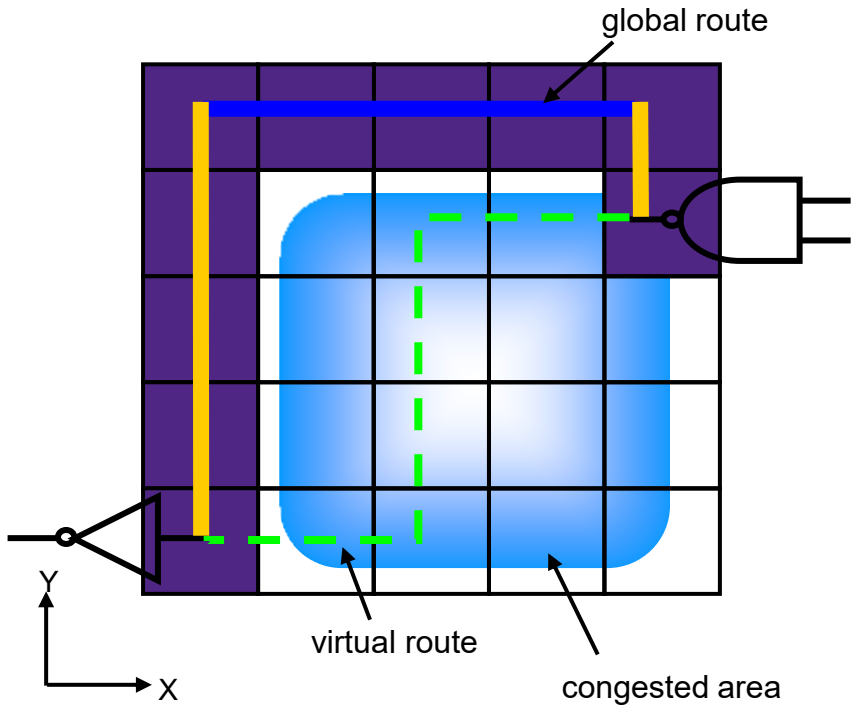
There is a limit to the number of nets that can be routed through a given small area

When you approach or exceed this limit, this area is said to be congested

Global Route for Congestion Map

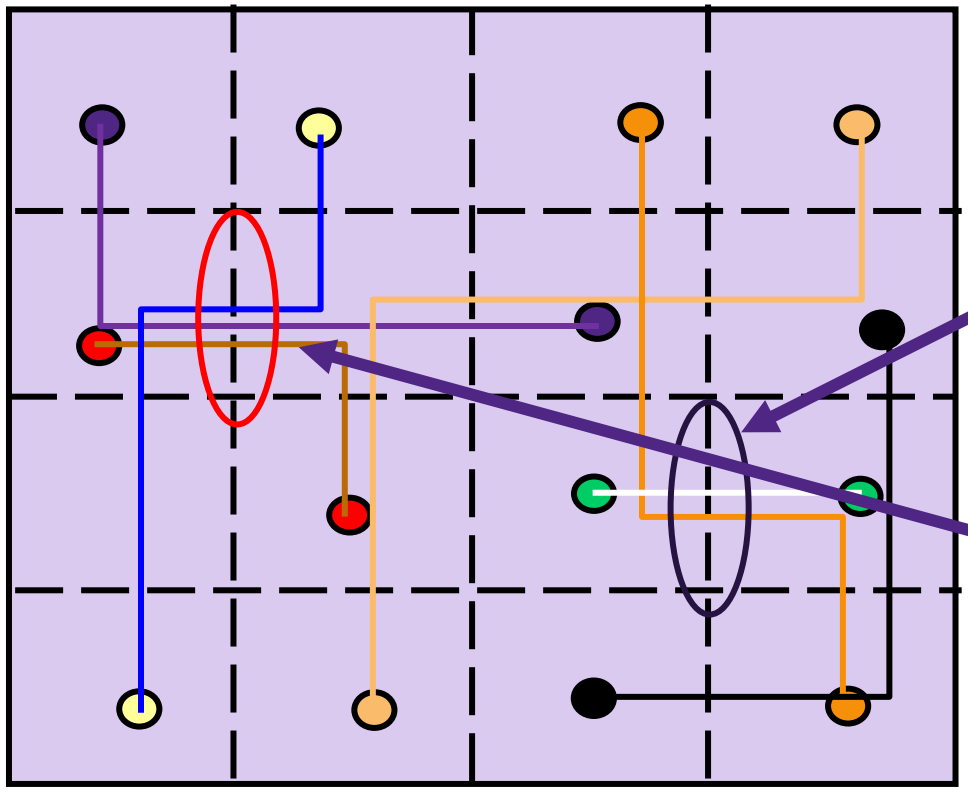
- GR for Congestion Map
  - GR assigns nets to specific metal layers and global routing cells (GRCs)
    - The detailed router follow GRCs path
  - GR determines if each GRC along a path has enough wire tracks for assigned nets
  - If not enough wire tracks, GR reassigns metal layers or GRC accordingly





Routing Congestion

- Routing resources < Routing Requirements → Congestion



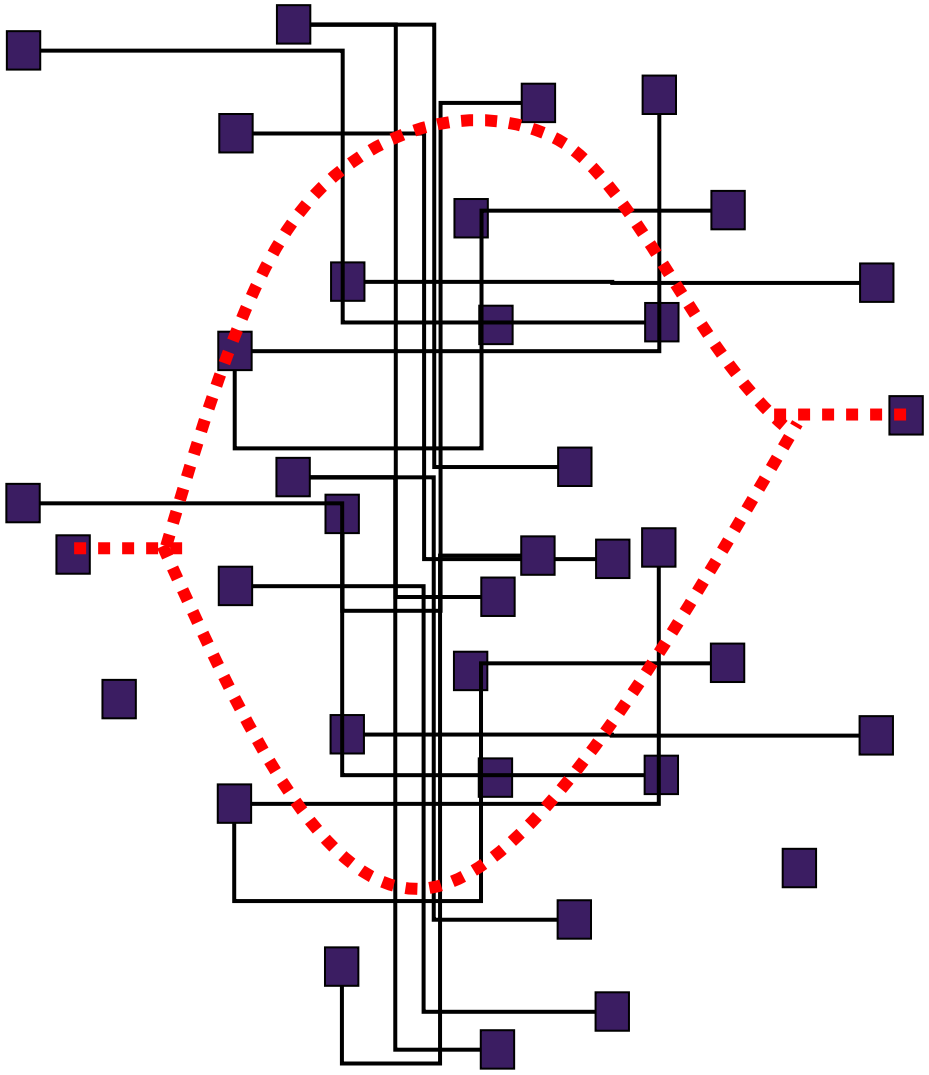
Routing requirement = 2
Routing resource = 2
overflow = 2 - 2 =0→ No Congestion

Routing requirement = 3
Routing resource = 2
overflow = 2 - 3 = -1 → Congestion

Requires Congestion-Aware Placement

Congested Design

- Not routable on severely congested design
  - It is important to minimize or eliminate congestion before continuing
  - Severe congestion can cause a design to be unroutable



Sources of Congestion and Solutions

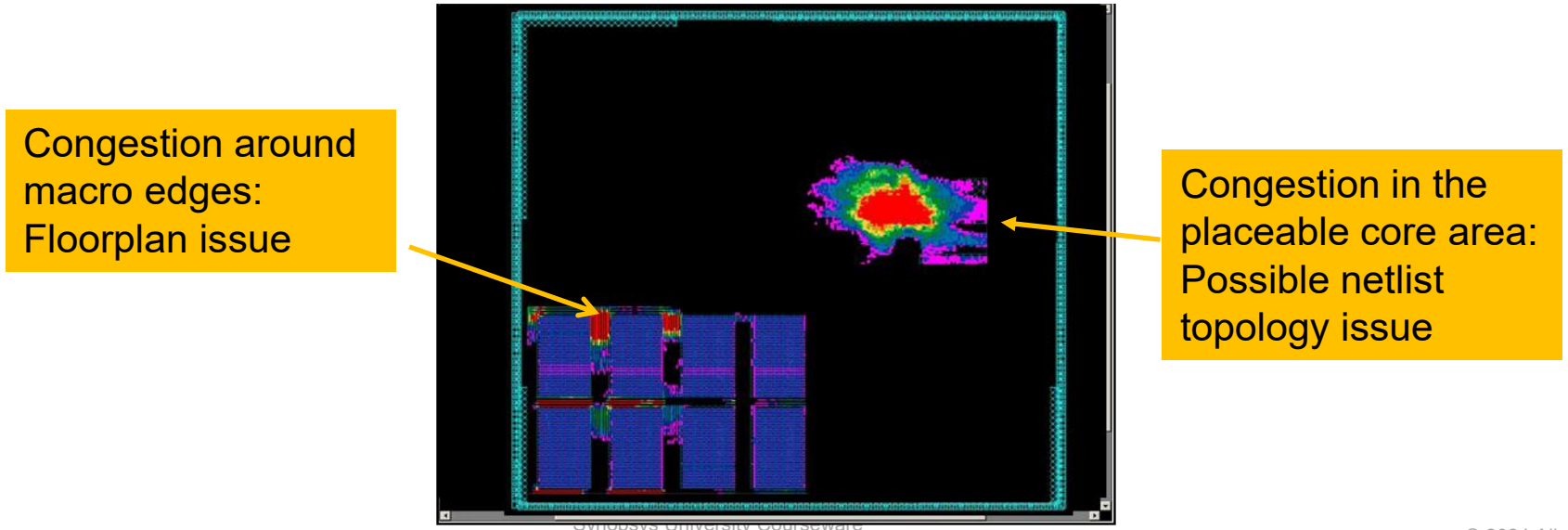
- High placement density
  - Too much routing required on small area
  - Solution: less dense placement
- Bad standard cell and IP pin access
  - Bad pin placement causes congestion
  - Solution:
    - Increase IP pin access
    - Increase standard cell pin access

Analyzing the Congestion Map

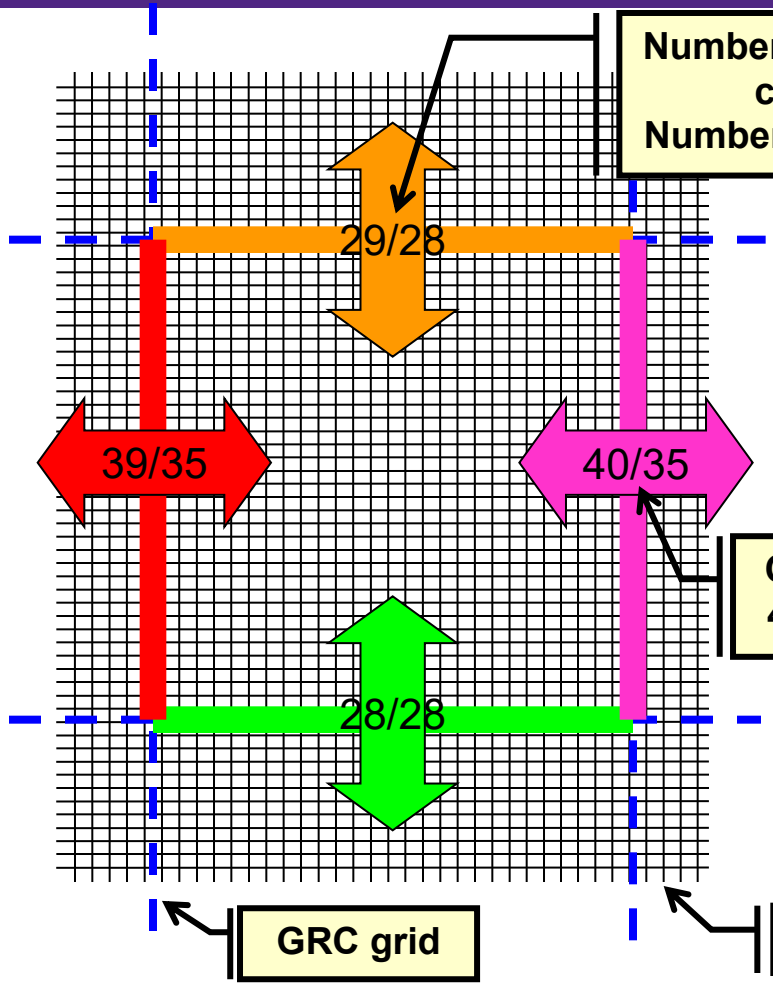
- Analyzing the Congestion Map
  - Reviewing the congestion map
    - Placing the potential hot spots
    - Their presence in expected areas
    - Presence or need of blockages
  - Reviewing cell placement in the GUI
    - Placing cells
    - Placing cell density in a region causing a congestion hot spot

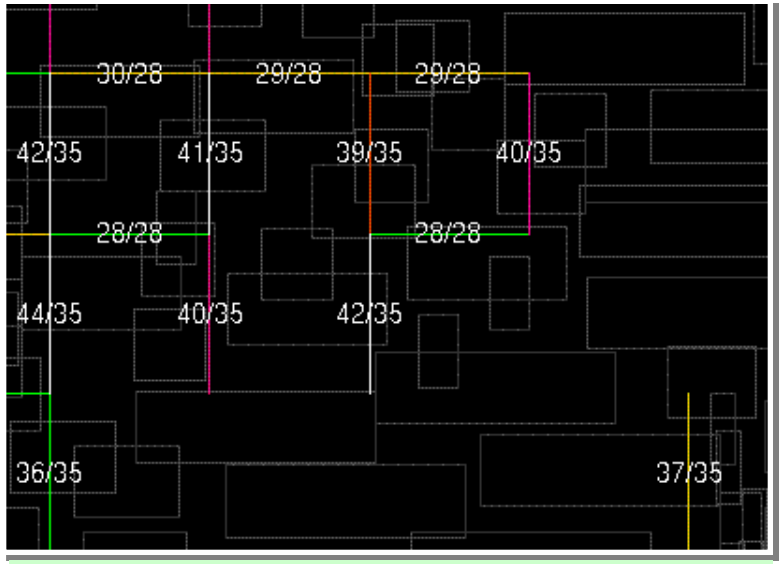
Analyzing Congestion in the GUI

- Use the congestion heat map to determine:
  - If your design has serious congestion that needs to be addressed
  - What is causing the various congestion areas and how best to address them
- Congestion can stem from two main sources:
  - The floorplan
  - The netlist



Understanding the Congestion “Heat Map”

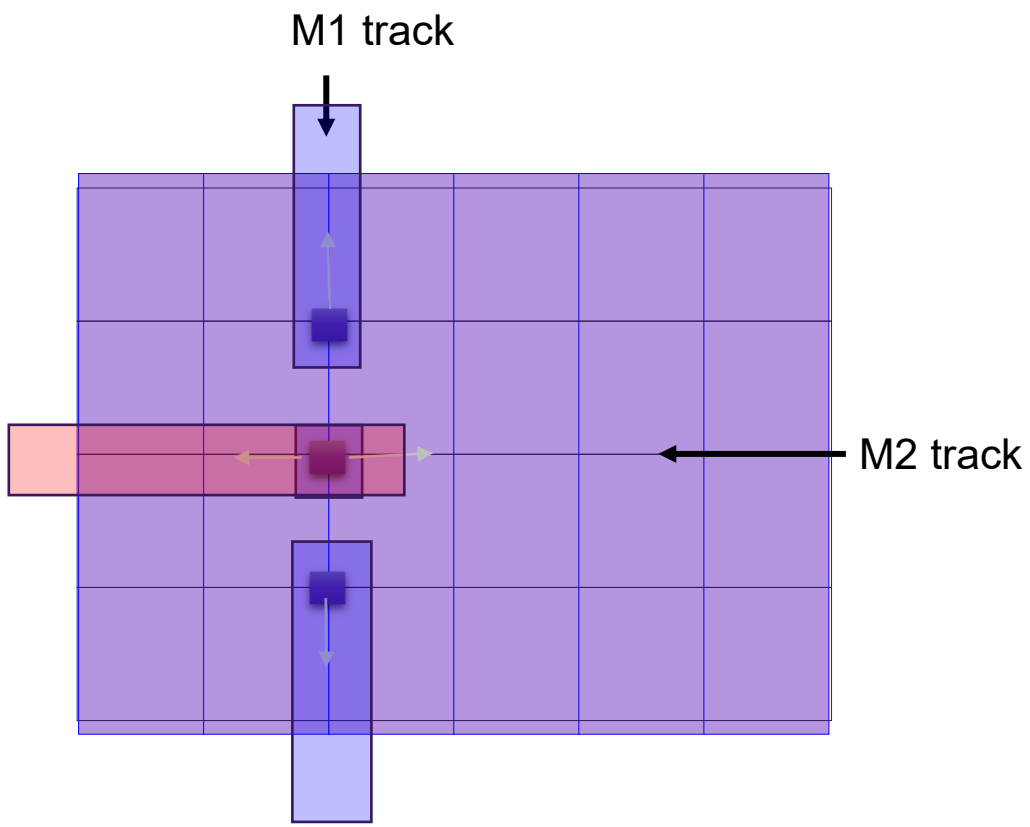




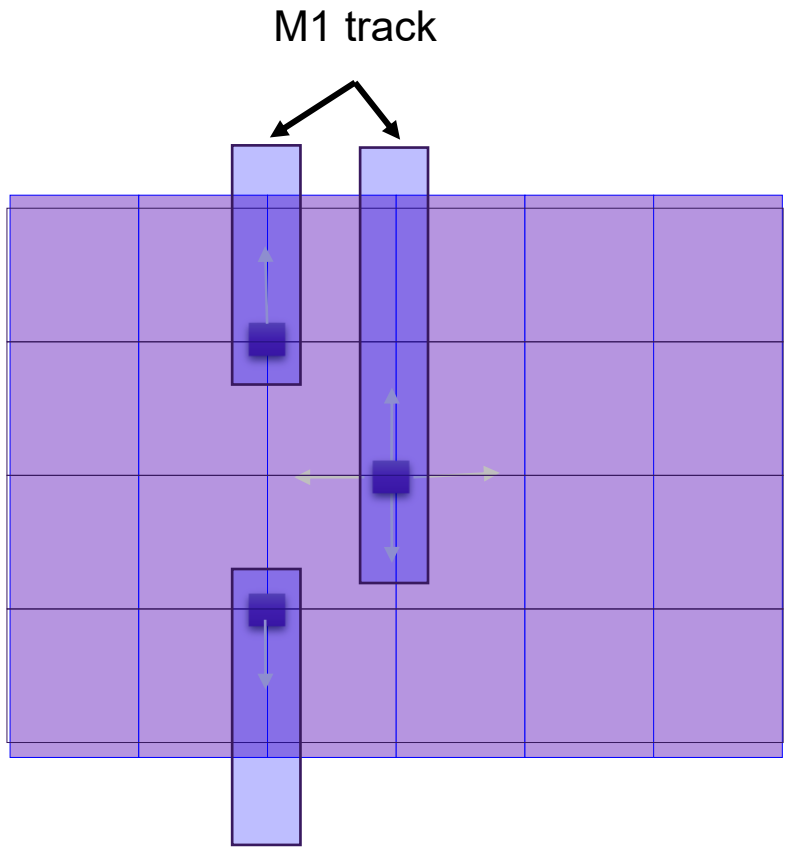
Congestion map (heat map)

The size of the ‘overflow’ determines the highlighting color – larger overflows are brighter or ‘hotter’

Standard Cell Routability Improvement

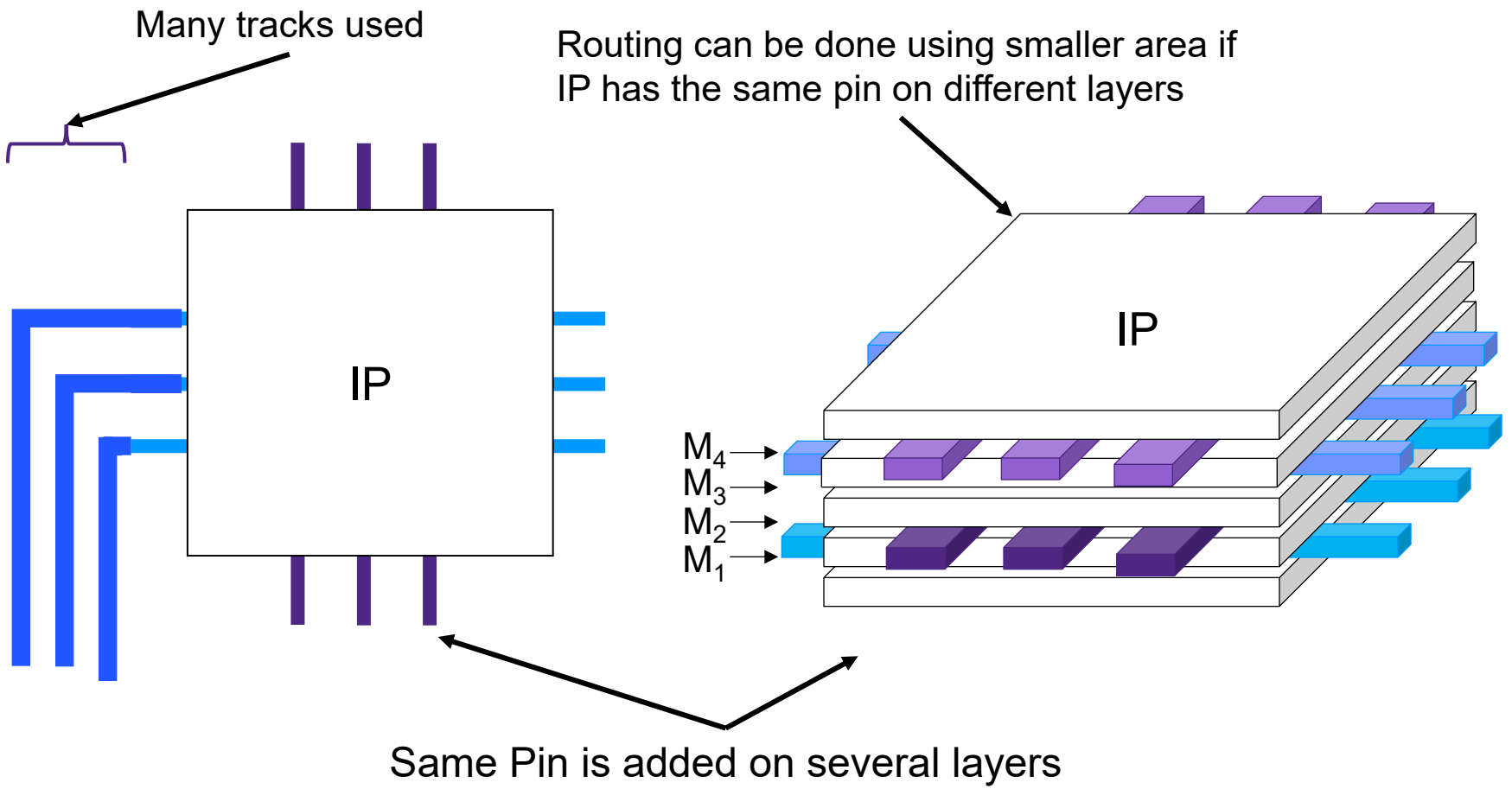


- If pins are placed on same track, additional M2 routing resources are spent



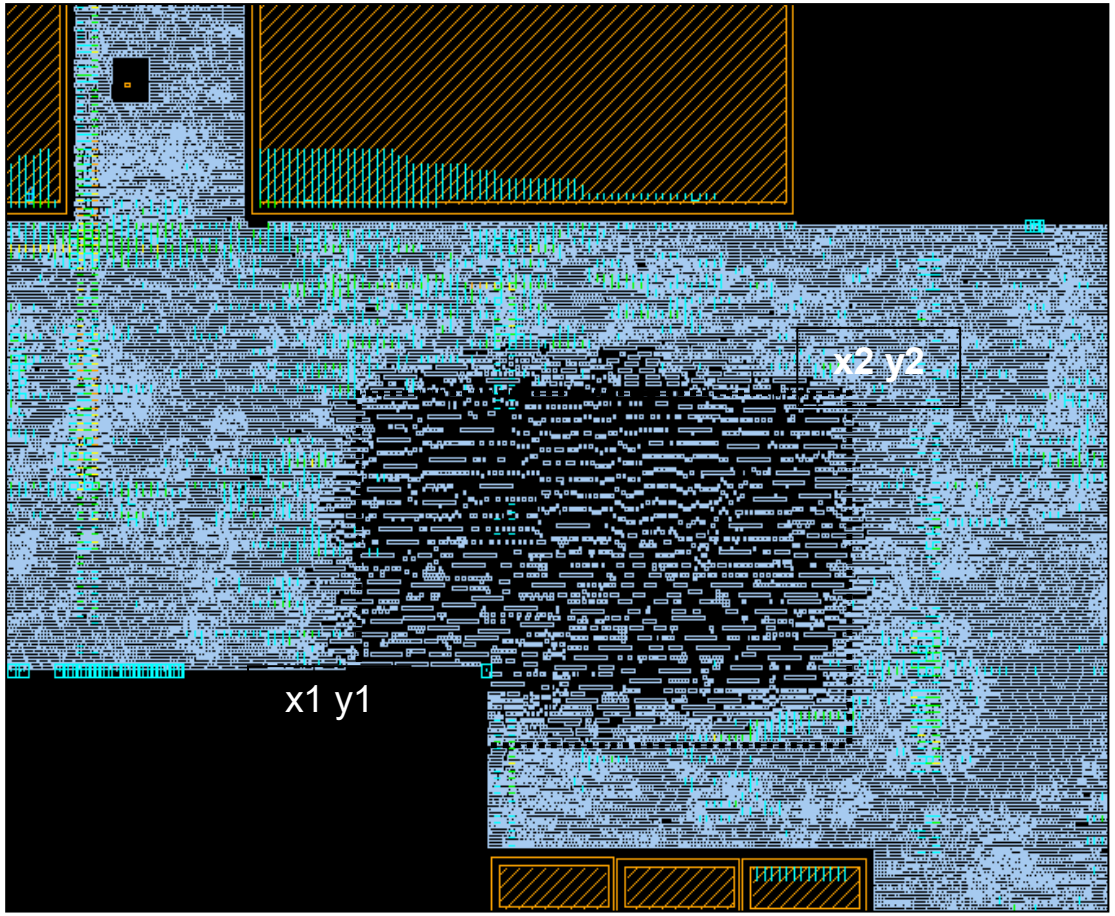
- Routing can be accomplished using only M1

IP Pin Access Improvement



Cell Density

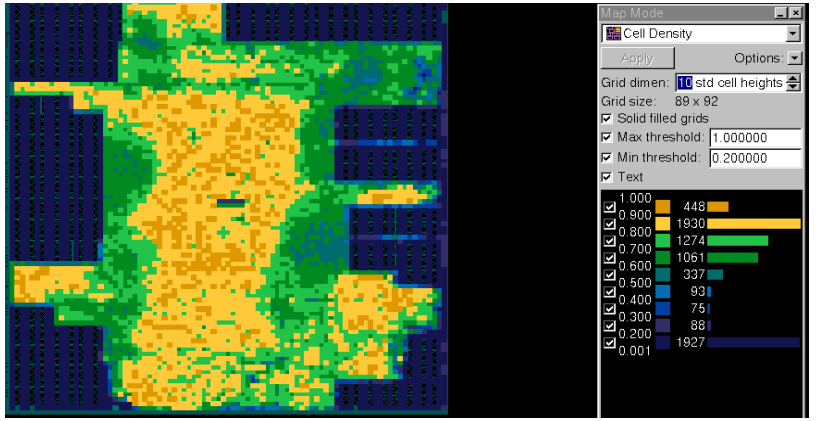
- Modifying Physical Constraints: Cell Density
  - Cell density can be up to 95% by default
    - Density level can also be applied to a specific region
  - Lower cell density in congested areas using –coordinate option



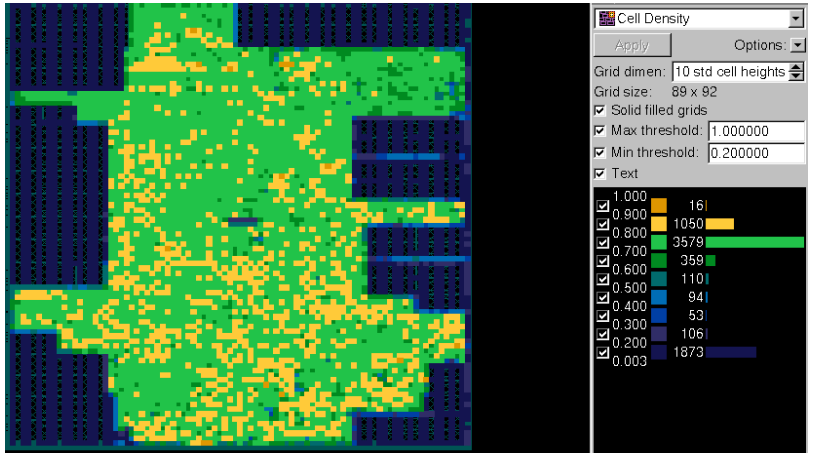
Congestion Map

- No need to use congestion unnecessarily
  - By default, physical synthesis tool performs some congestion optimization that has a reasonable chance of providing acceptable congestion
  - Congestion increases efforts of algorithm of a congestion
    - On average – congestion increases runtime by 20%
  - For better correlation to post-route, congestion-driven placement is enabled based on GR congestion map

Causes high local utilization

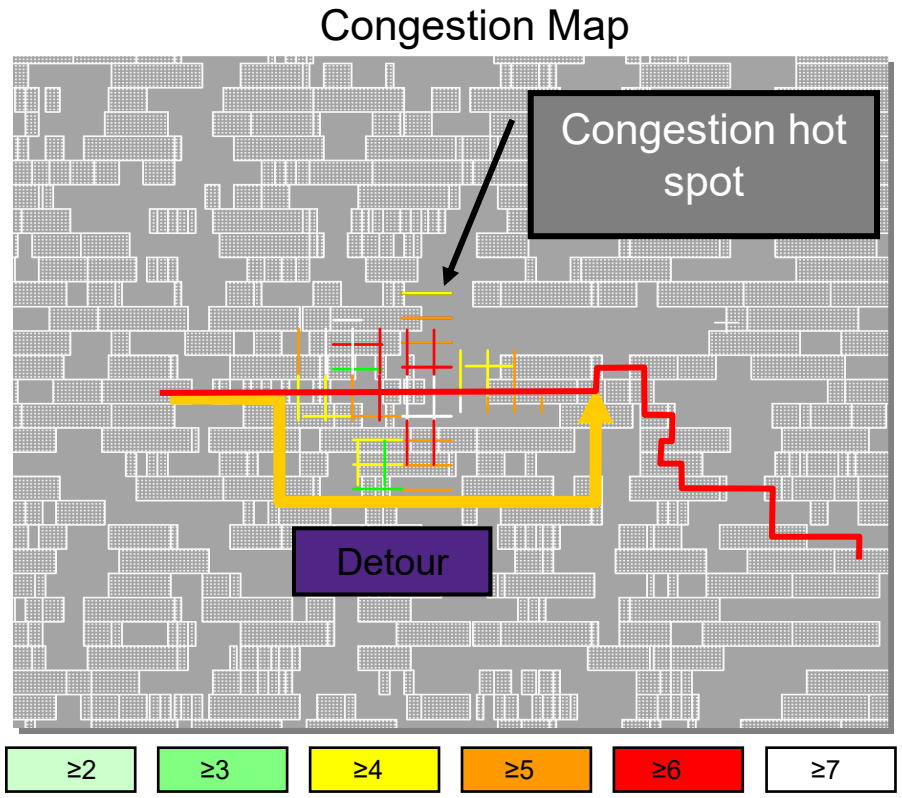


Gives uniform density



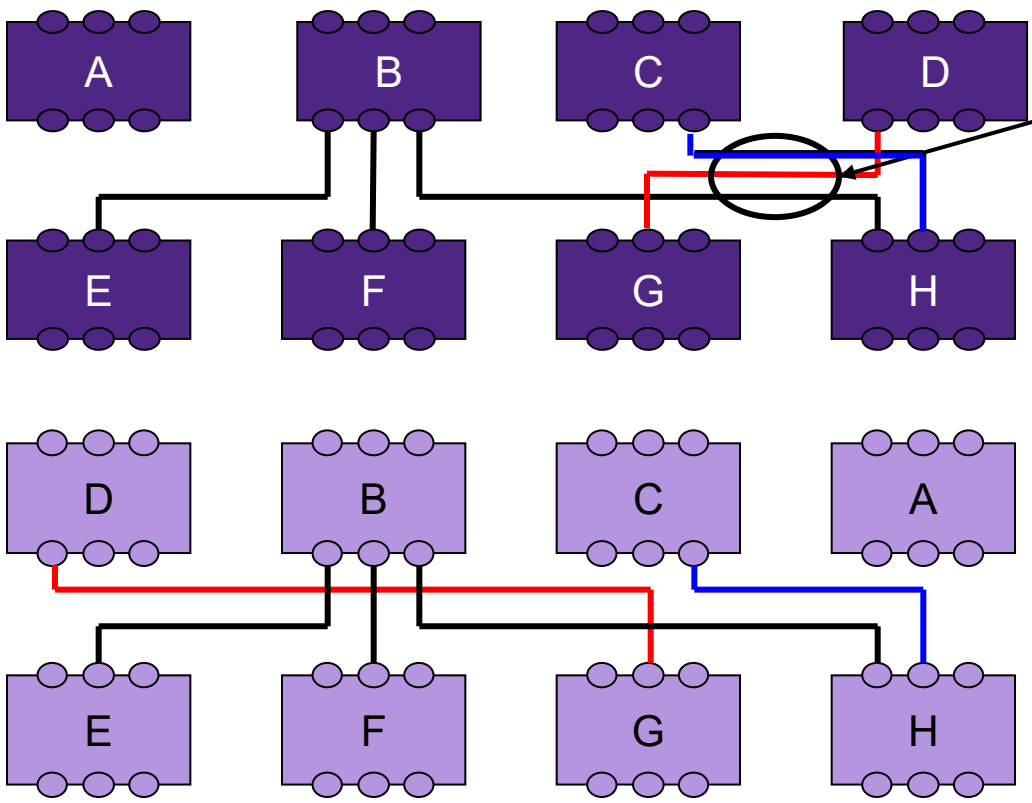
Placement Issues with Congestion

- Issues with congestion
  - If congestion is not too severe, the actual route can be detoured around the congested area
  - The detoured nets will have worse RC delay compared to the VR estimates



In highly congested areas, delay estimates during placement will be optimistic.

Standard Cells Placement Optimization

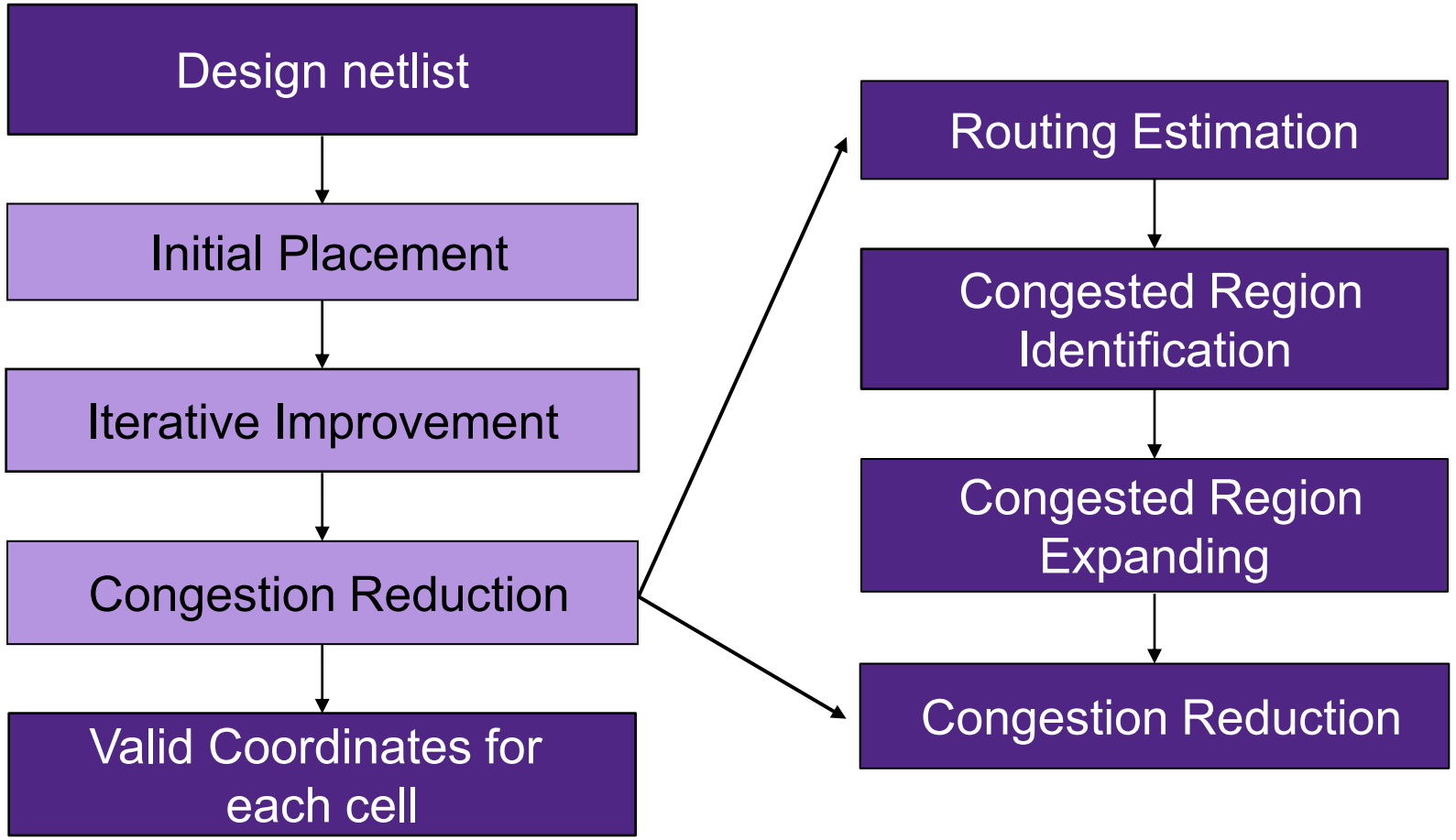


Shorter Wire length
Channel Density: 3
(track: 3)

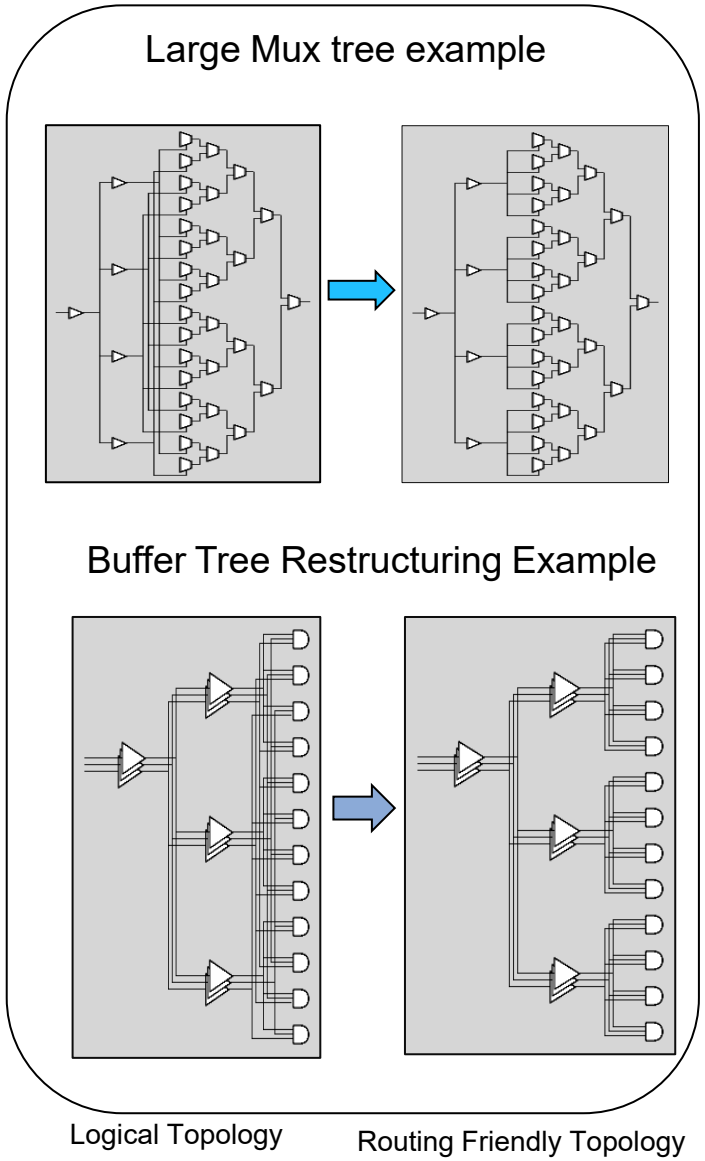
(channel capacities:2)
Unroutable Layout

Longer Wire length
Channel Density: 2
(track: 2)

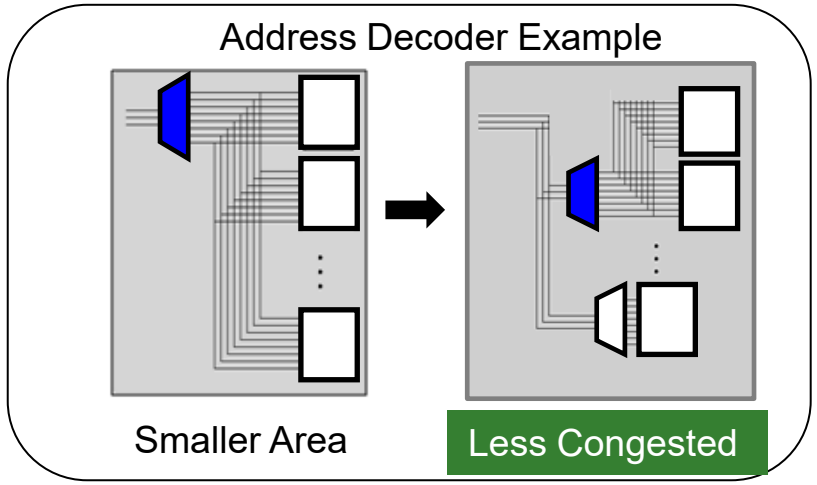
Congestion-driven Placement



Netlist Topology Congestion Optimization 1/3

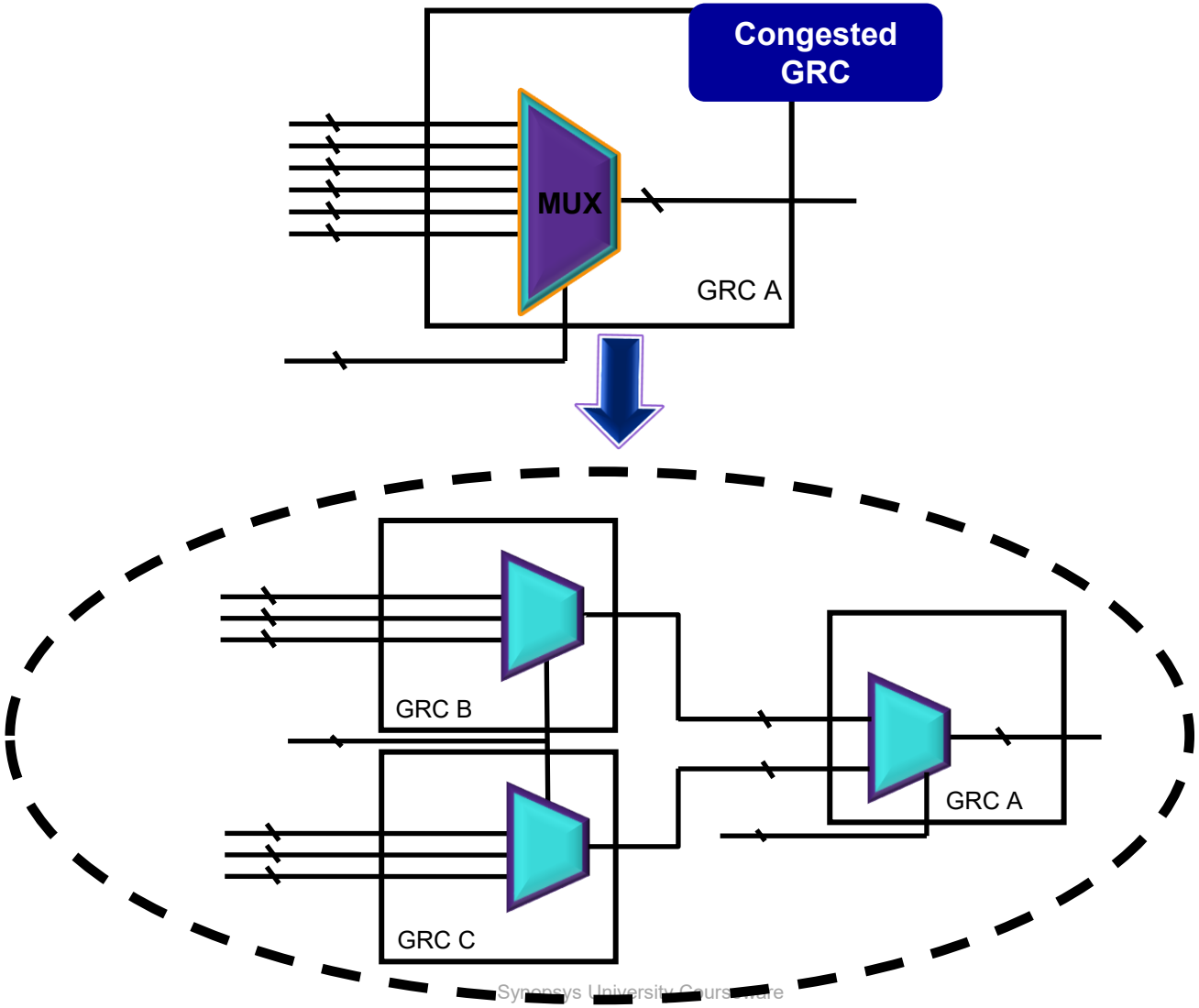


- Makes routing-friendly netlist “topology” or “structural” changes
  - Minimizes highly connected structures
  - Minimizes wires and wire crossing
  - Shares and un-shares logic as needed
- Concurrent optimization with timing, area, power and scan chains



Netlist Topology Congestion Optimization 2/3

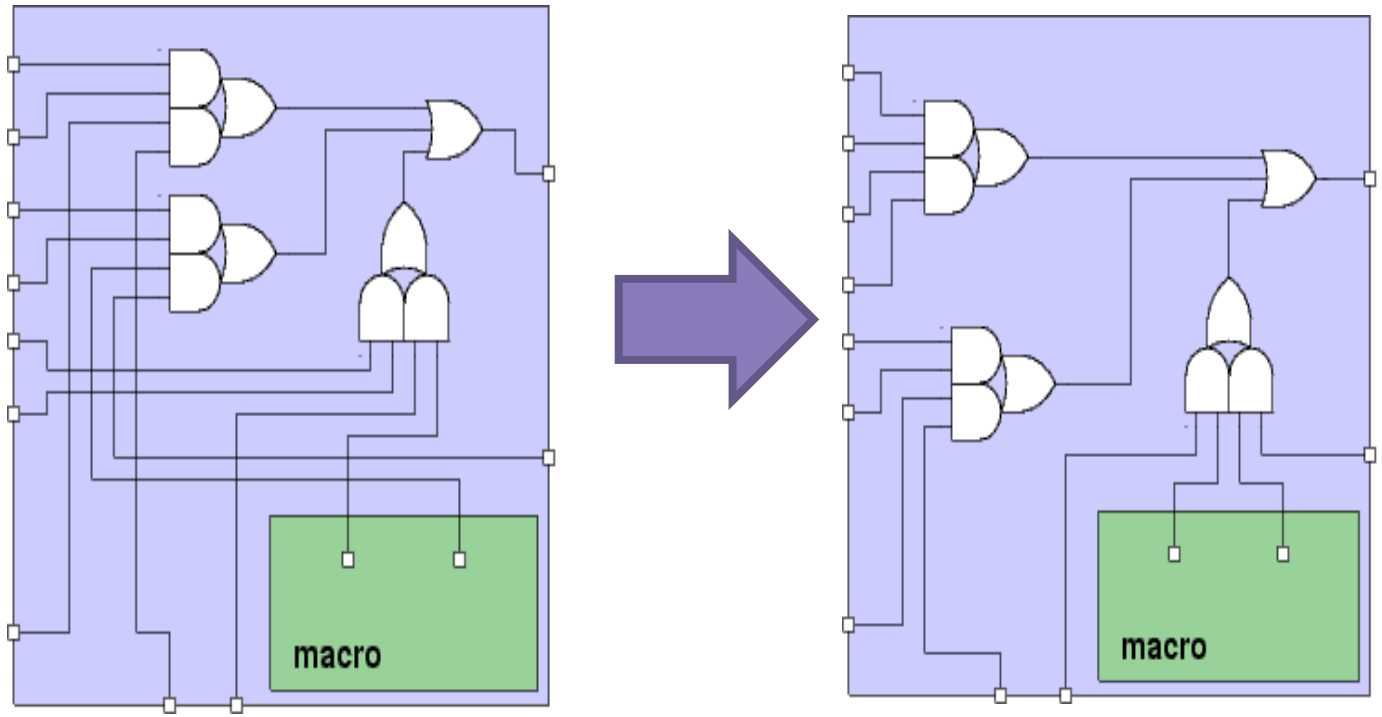
MUX restructuring



Netlist Topology Congestion Optimization 3/3

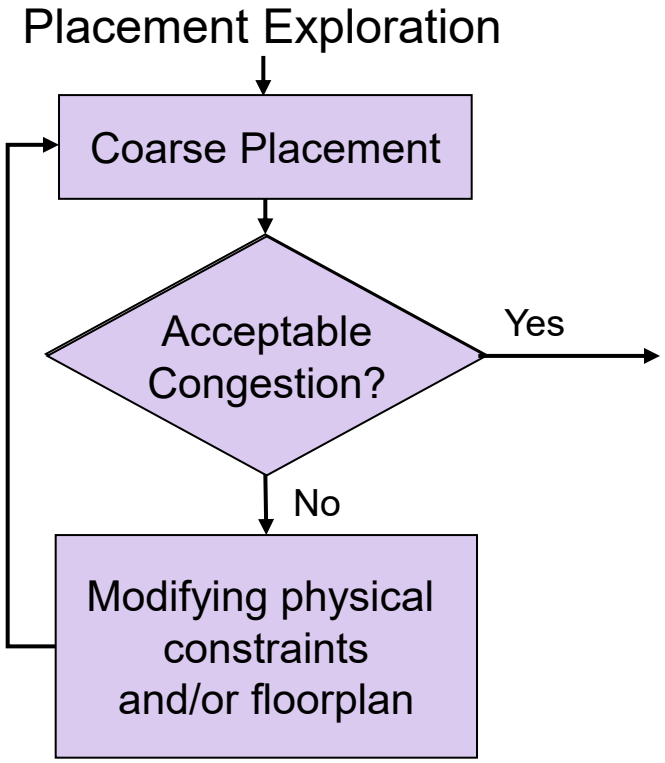
Associative-Commutative Tree Decomposition

Improved congestion by decomposing and restructuring AND/OR trees based on port or macro pin locations



Strategies to Fix Congestion

- Rerunning coarse placement with congestion-driven option
- Modifying physical constraints
  - Adjusting cell density in congested areas
  - Adding/modifying blockages
- Modifying the floorplan
  - Moving macros
  - Changing core shape/size
  - Moving pins/pads
  - …



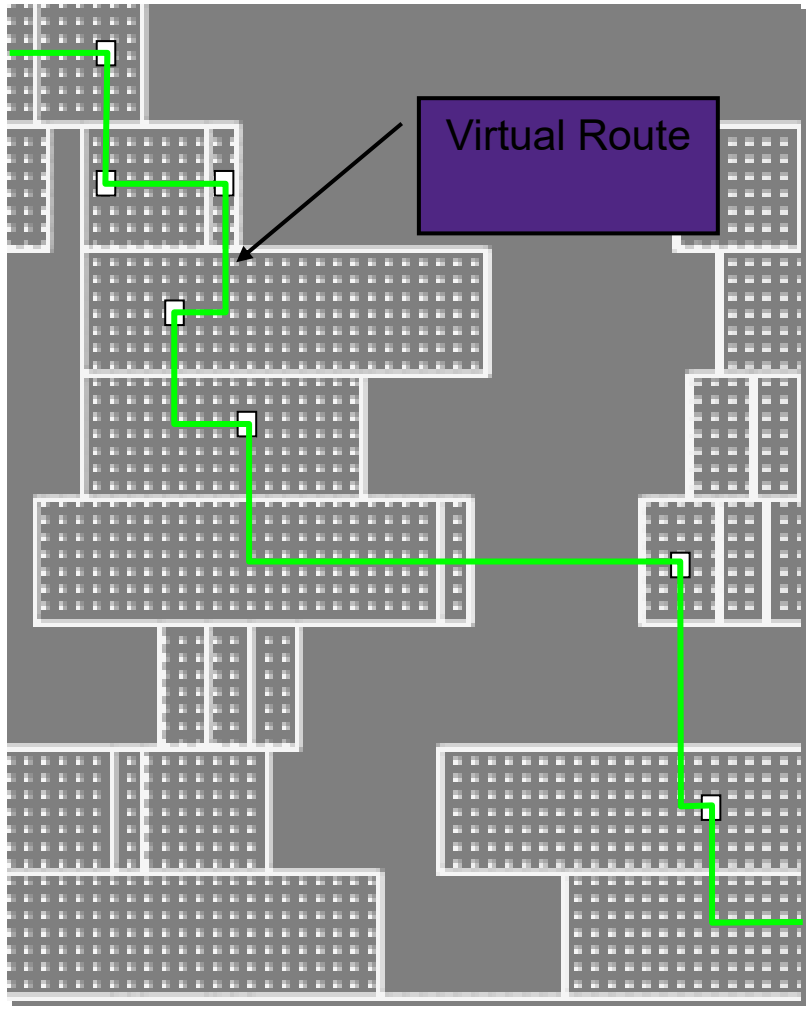
Trying each recommendation one-by-one, in sequence, then in combination if needed

Agenda

- Standard cell placement
- Congestion-driven placement
- ***Timing-driven placement***
- Macro block placement and constraints

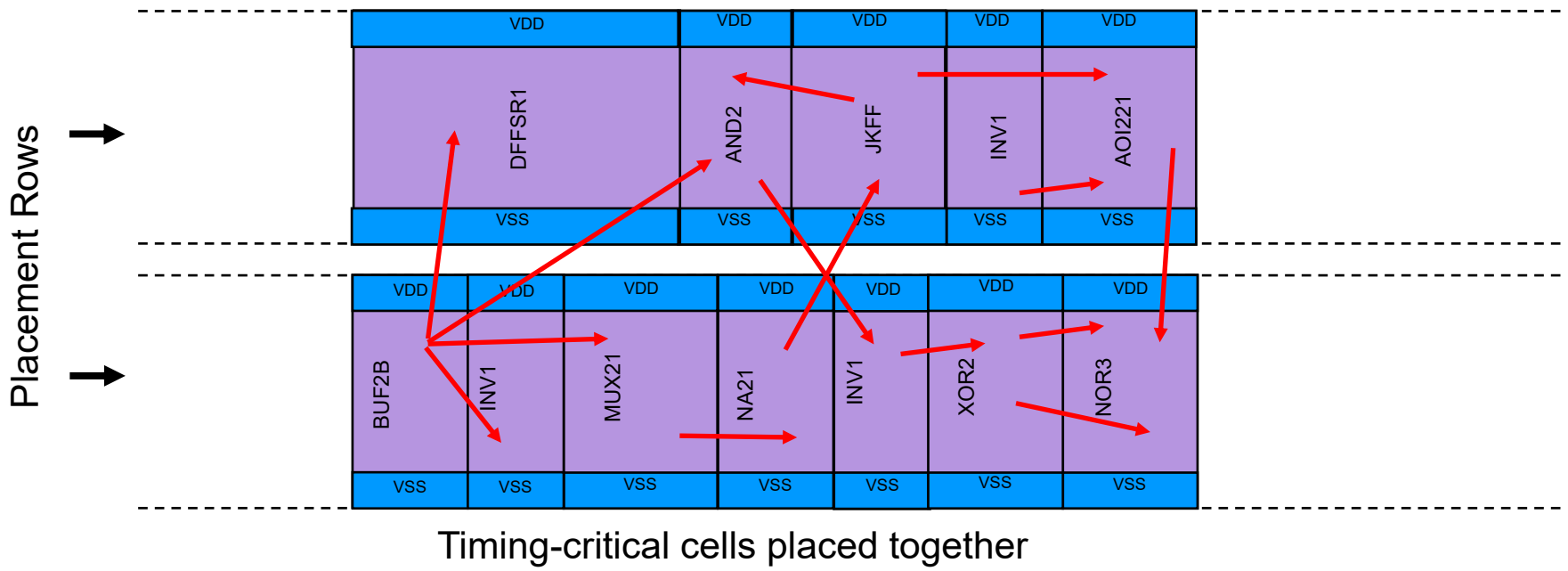
Timing-Driven Placement

- Timing-driven Placement
  - All steps including placement are timing-driven
  - Timing-driven placement tries to place critical path cells close together to reduce net RCs and meet setup timing
  - RCs are based on Virtual Route (VR)



Placement in Site Rows

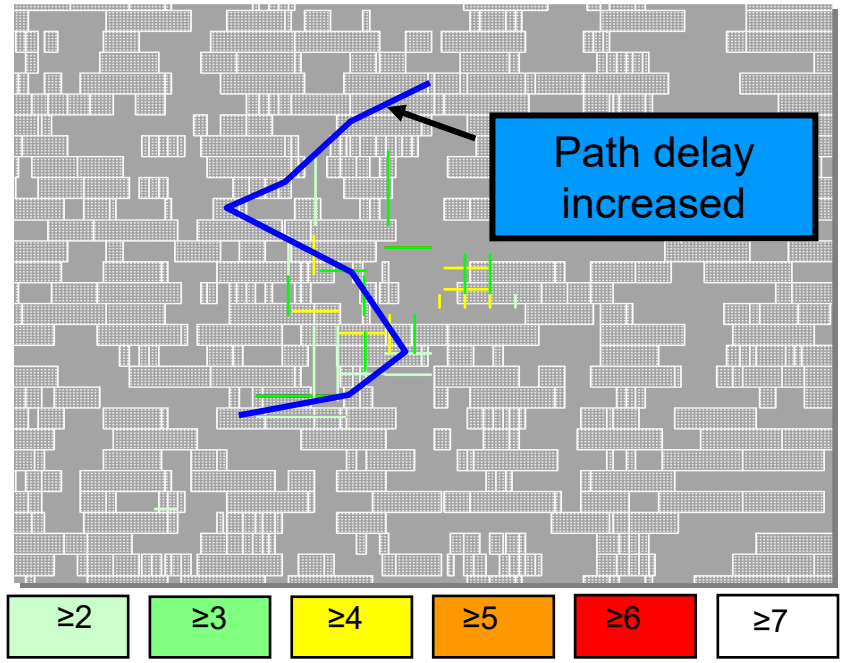
- Timing-Driven Placement
  - Standard cells are placed in “placement rows”
  - Cells in a timing-critical path are placed close together to reduce routing-related delays → Timing-Driven Placement



Timing-critical cells placed together

Congestion vs. Timing-Driven Placement

- Cells along timing critical paths can be spread apart to reduce congestion.
- These paths may now violate timing.



  - Small timing violations can be resolved later by incremental logic optimization.

Agenda

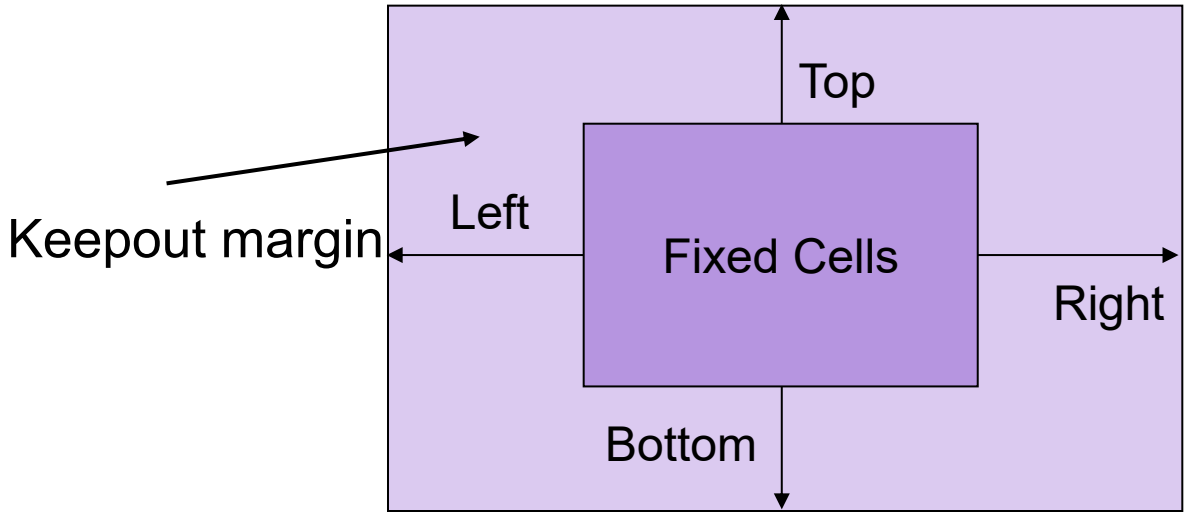
- Standard cell placement
- Congestion-driven placement
- Timing-driven placement
- ***Macro block placement and constraints***

Macro Placement Constraints

- Macro placement constraints has impact on the placement and further global routing of standard cells.
- Some basic constraints' examples of Macro placement:
  - Alignment of macros by edges
  - Grouping macros such that for standard cells rectangular area should be left as much as possible
  - Alignment of macros with core boundary
- Some basic parameters of Macro placement:
  - Routability
  - Timing
  - Wire length
  - Area for standard cells

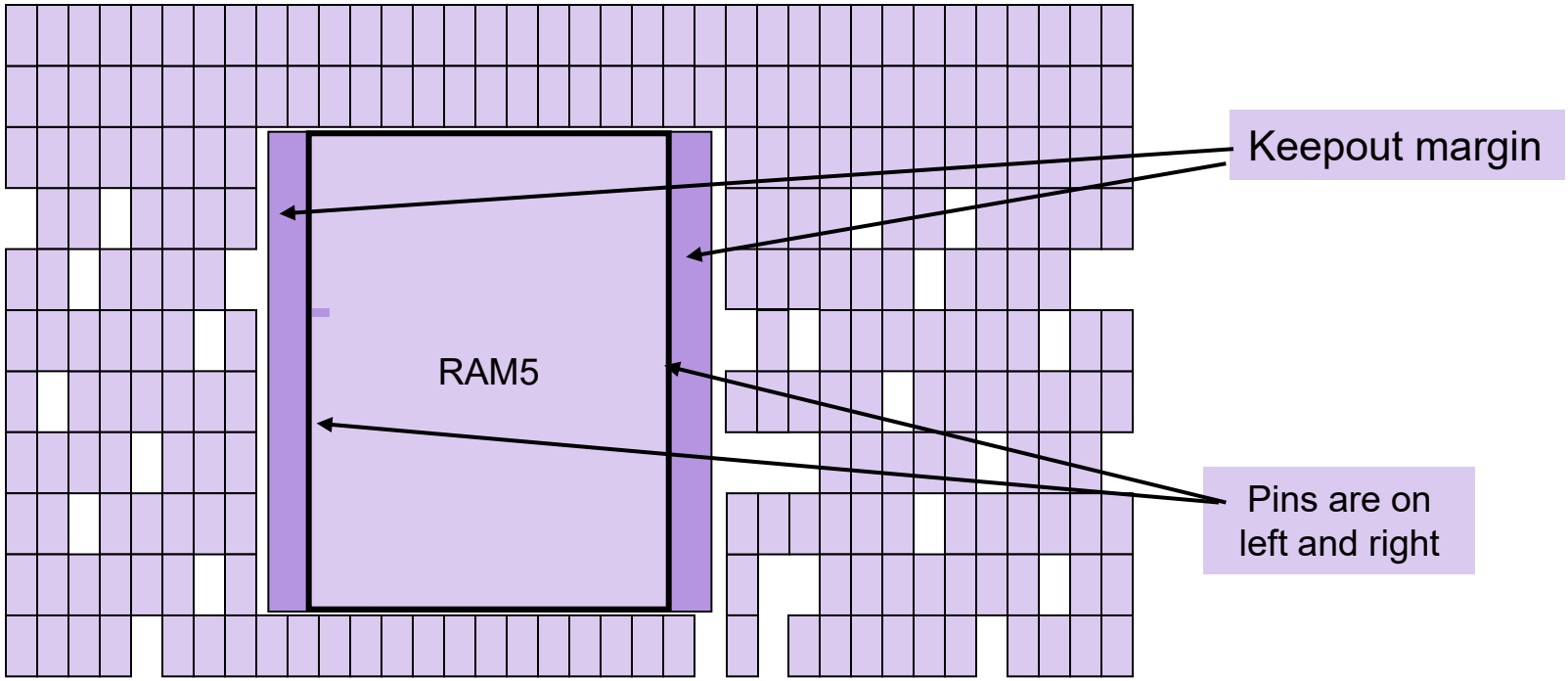
Defining Placement Blockages

- Placement blockages are areas where leaf cells must not be placed during placement and legalization, including overlapping any part of the placement blockage.



Macro Keepout Margin

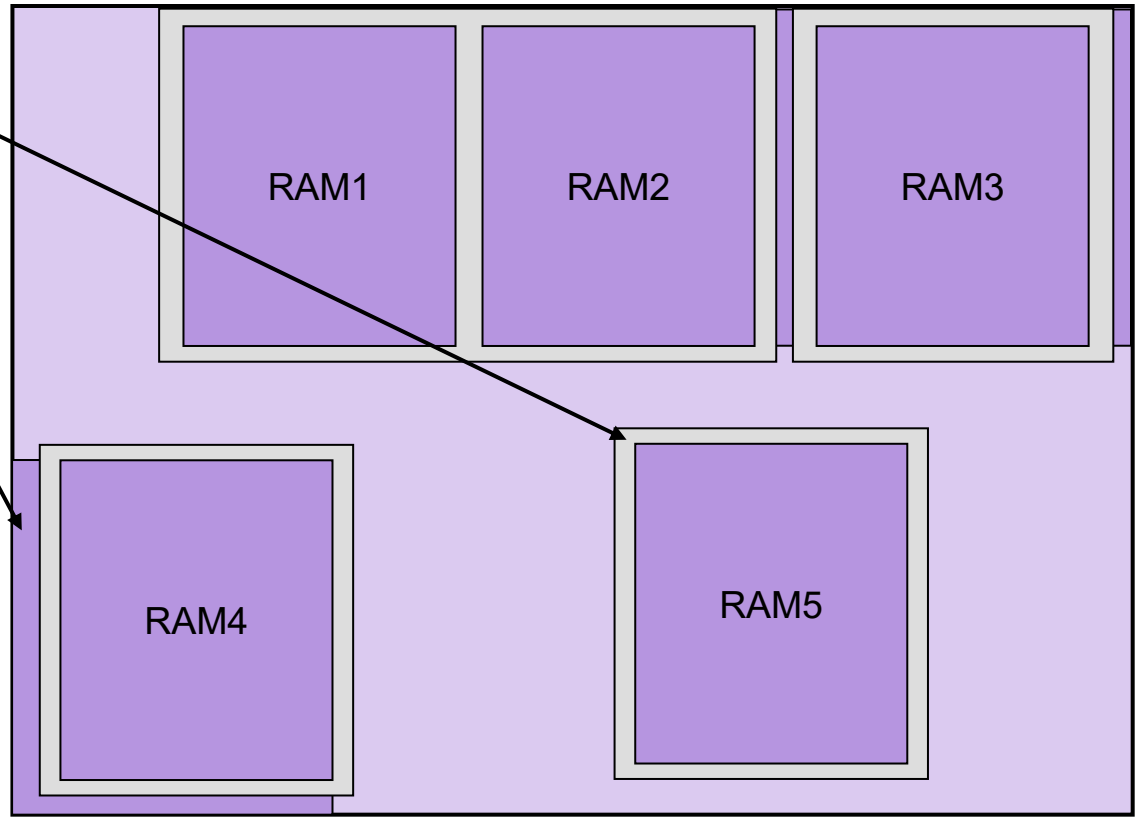
- A keepout margin is a region around the boundary of fixed macros in the design in which no other cells are placed.



Placement Blockages

Hard blockage always created on all four sides

Soft blockage created only for the channels between the macros or between the macro and the core boundary



- **create_placement_blockage -boundary** ${boundary_coordinates} **-type** hard
- **create_placement_blockage -boundary** ${boundary_coordinates} **-type** soft