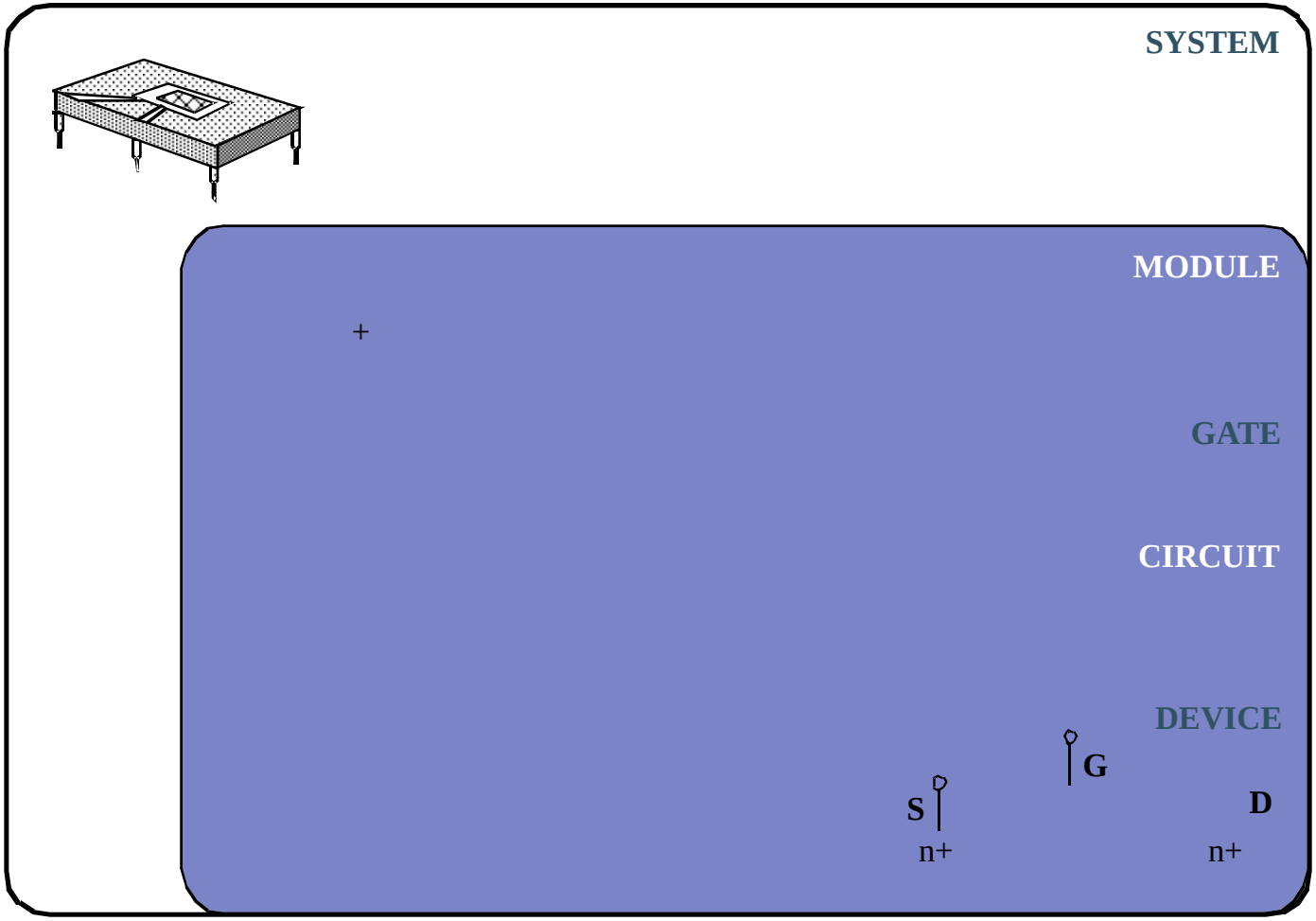
NARLabs
National Applied Research Laboratories

Brief Introduction of Cell-based Design

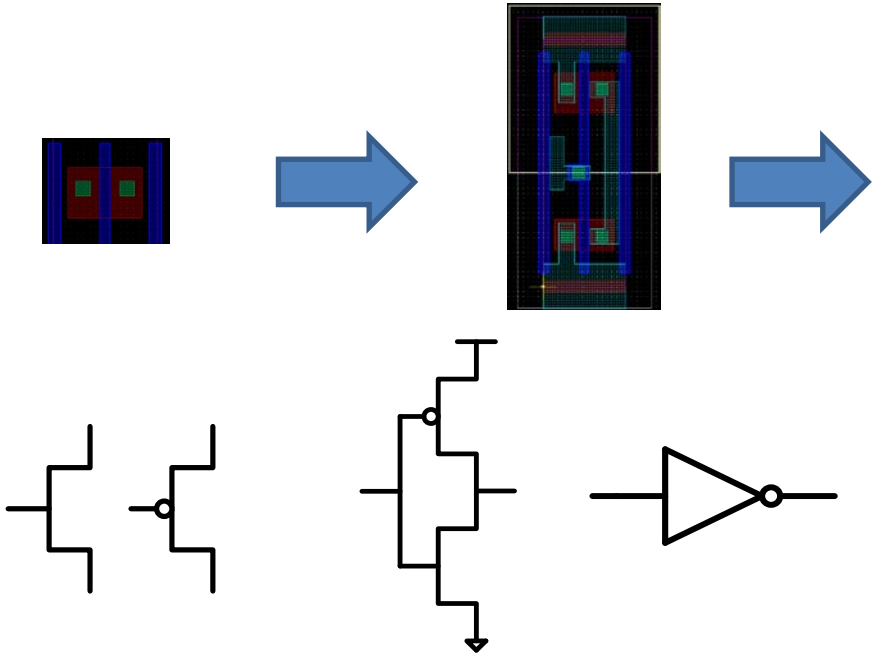
Ching-Da Chan CIC/DSD

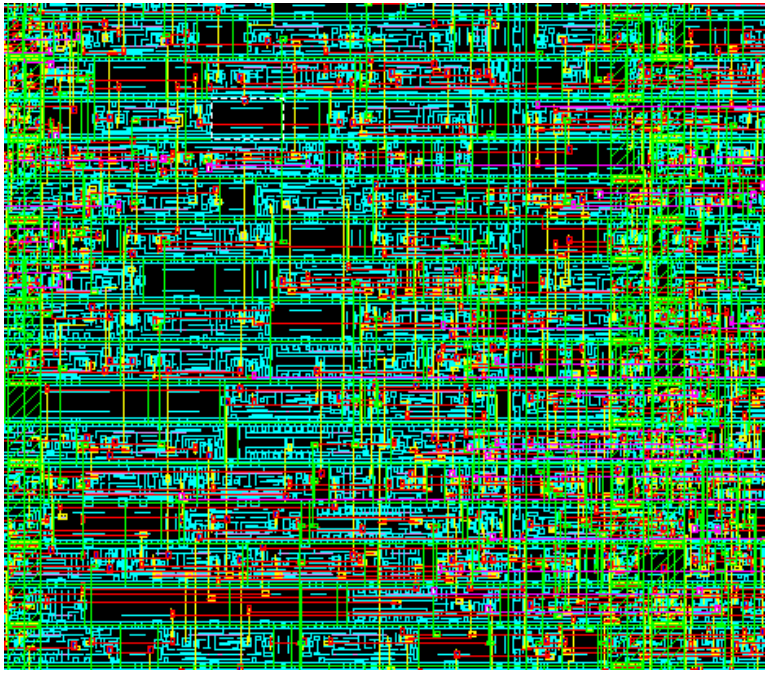
Design Abstraction Levels



Full Custom V.S Cell based Design

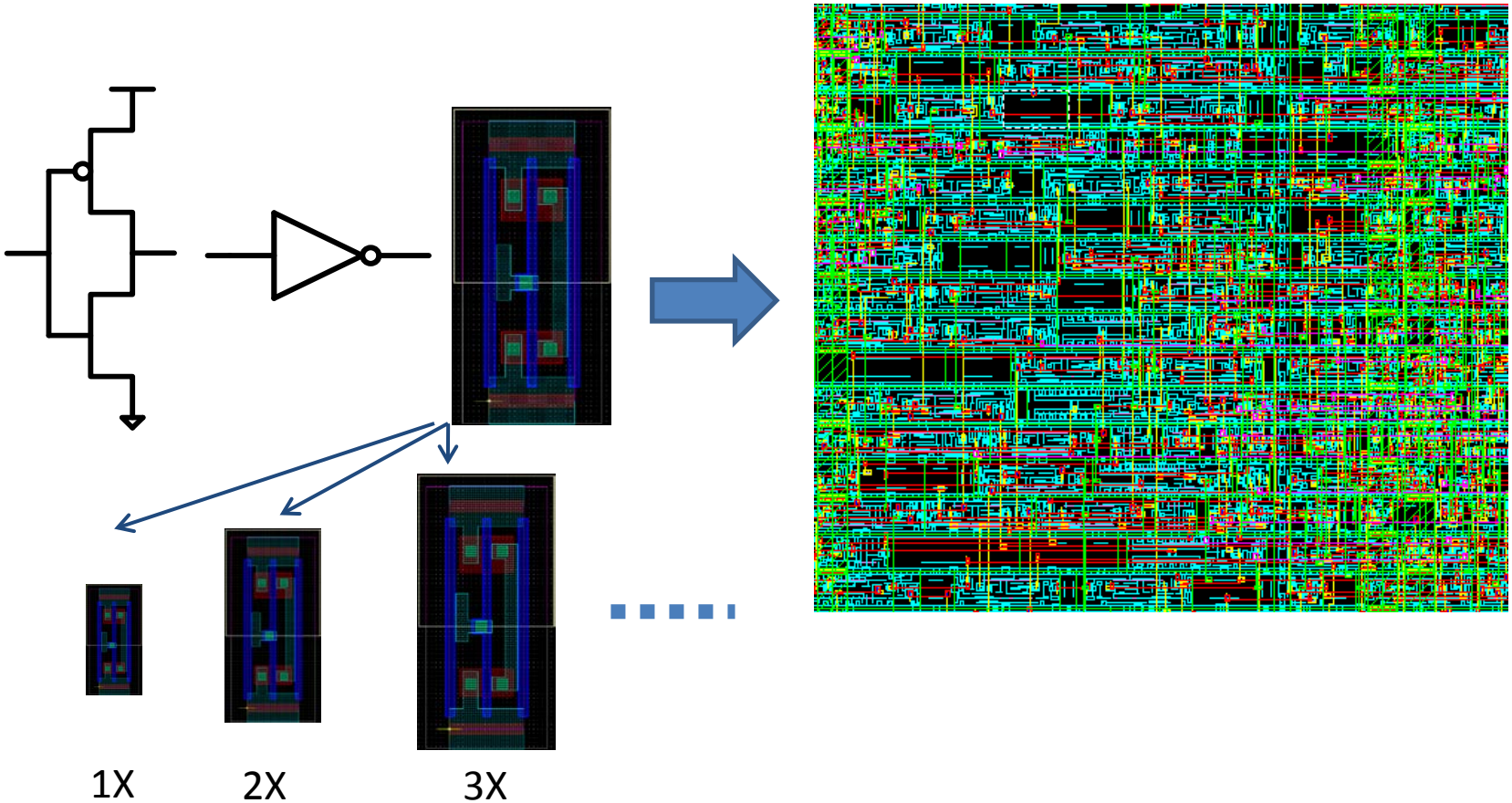
- Full custom design
  - Better patent protection
  - Lower power consumption
  - Smaller area
  - More flexibility





Full Custom V.S Cell based Design (cont.)

- Cell-based design
  - Less design time
  - Easer to implement large system



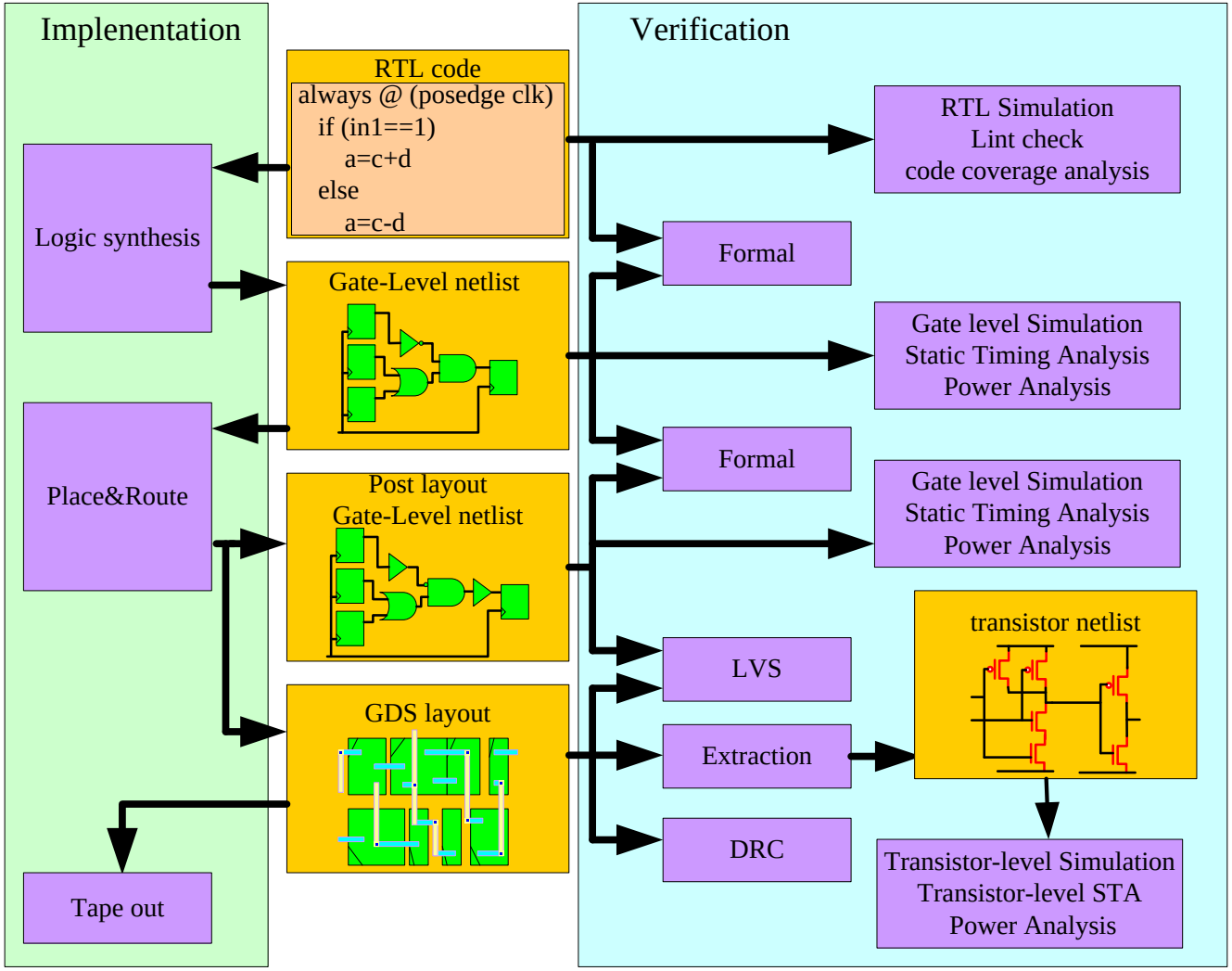
NARLabs

Prepare Standard Cell Library

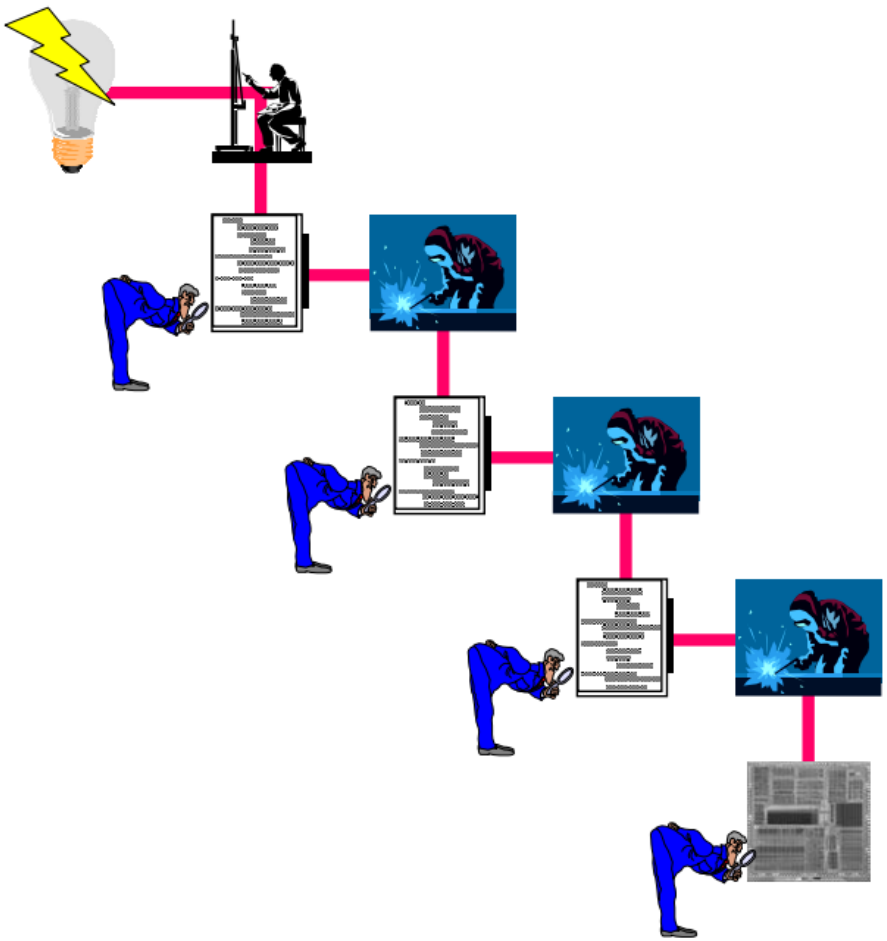
1. Circuit design
   - Equal height but different width.
   - Cell unit is unit tile(ex. 2X, 3X, 4X…).

2. Cell Characterization
   - Characterize all condition of input signal.
   - Extraction the cell view(abstract)

Cell-Based IC Design Flow Overview

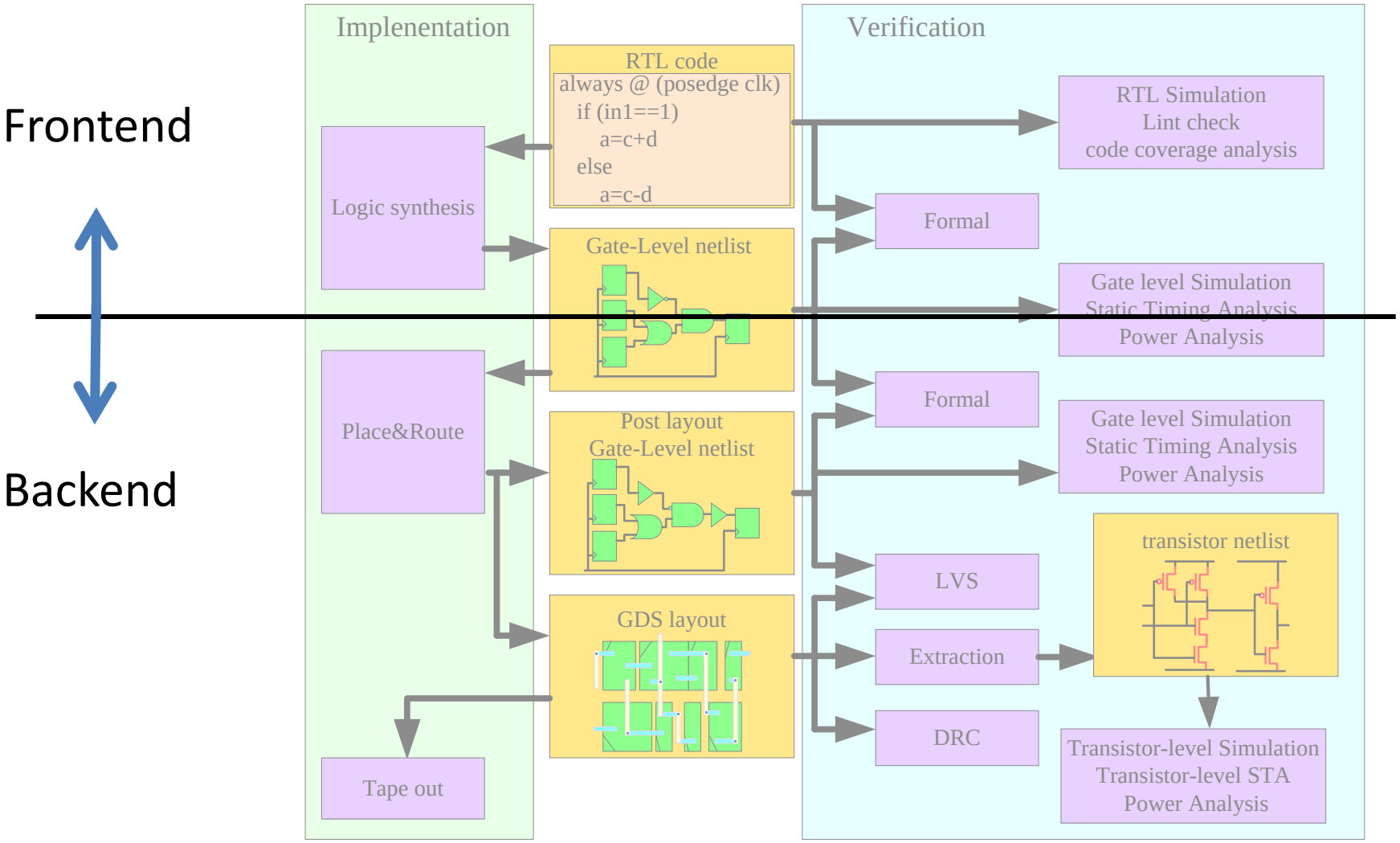


General Design Process



Idea
Design
Verification
Implementation
Verification
Implementation
Verification
Implementation
…
Verified Chip Layout

Cell-Based IC Design Flow Overview

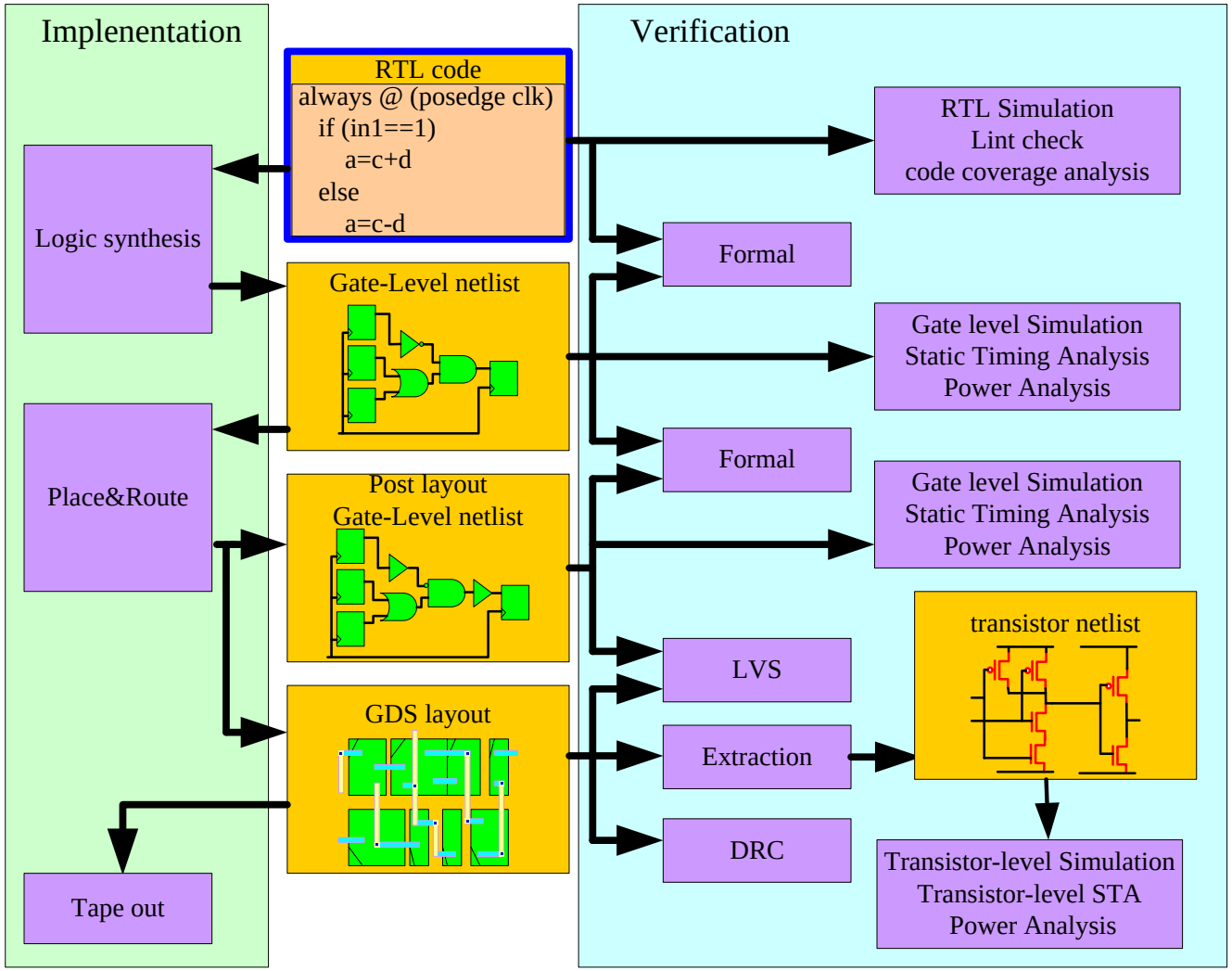


Partition of Design Flow

- Front end
  - System specification and architecture
  - HDL coding & behavioral simulation
  - Synthesis & gate level simulation
  - dynamic simulation and static analysis

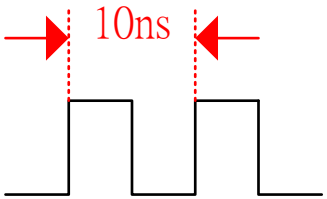
- Back end
  - Placement and routing
  - DRC (Design Rule Check), LVS (Layout vs Schematic)
  - dynamic simulation and static analysis

Cell-Based IC Design Flow - RTL Design



Specification of Design Example

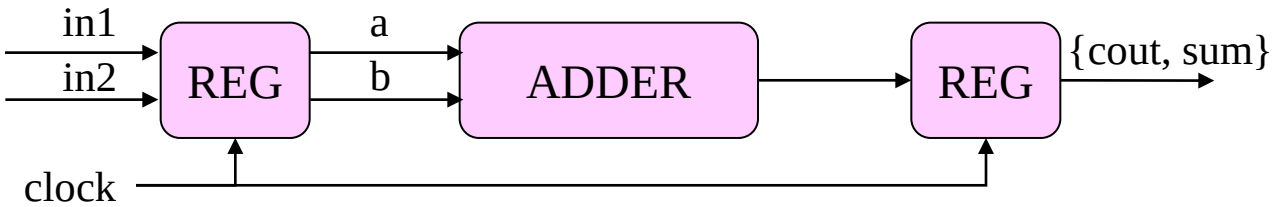
- Design Example Specification
  - Two 32-bit unsigned integer inputs are captured at the positive clock edge and the sum outputs at the following positive edge clock edge
    - 32-bit unsigned integer inputs means input data Types
    - Positive clock edge means input or output spec.
  - The clock period is 10ns
    - Clock period is the distance of two clock edge.



    - 10ns means the data must calculate finished at 10ns
  - The power consumption should be less than 1mw

RTL HDL Coding

- RTL = Register Transfer Level



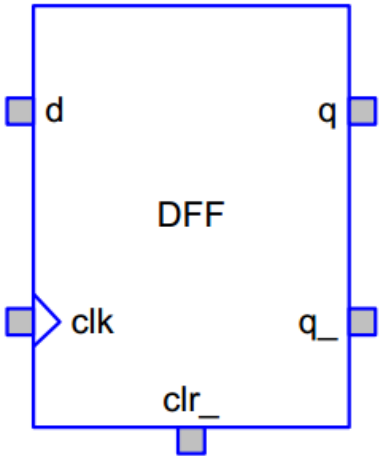

```
module MYADDER (clock, in1, in2, sum, cout);

input clock;   input [31:0] in1, in2;
output [31:0] sum;   output cout;

reg [31:0] a, b;   reg [31:0] sum;

always @(posedge clock)
        begin
          a = in1; b = in2; {cout, sum} = a + b;
        end

endmodule
```

Basic Building Block: Modules

- Modules are the basic building blocks.
- Descriptions of circuit are placed inside modules.
- Modules can represent:
  - A physical block (ex. standard cell)
  - A logic block (ex. ALU portion of a CPU design)
  - The complete system
- Every module description starts with the keyword module, followed by a name, and ends with the keyword endmodule.

Communication Interface: Module Ports

- Modules communicate with the outside world through ports.
- Module port types can be: **input, output or inout(bidirectional)**




```
module DFF (d, clk, clr_, q, q_);
  output q, q_;
  input  d, clk, clr_;

endmodule
```

What's Inside the Module?

Structural Descriptions

```
module ...(...);
  . . .
  not (a, b);
  and (c, d, e);
  xor (x, y, z);
endmodule
```

Behavioral Descriptions

```
module ...(...);
  . . .
  assign out = a & b;
  assign q = ~p
endmodule
```

Behavioral Descriptions

```
module ...(...);
  . . .
  always @(. . .)
    begin
      . . .
    end
  always @(. . .)
    begin
      . . .
    end
  initial @(. . .)
    begin
      . . .
    end
endmodule
```

Mixed Descriptions

```
module ...(...);
  . . .
  and (a, b, c);
  assign out = a & b;
  always @(. . .)
    begin
      . . .
    end
  initial @(. . .)
    begin
      . . .
    end
endmodule
```

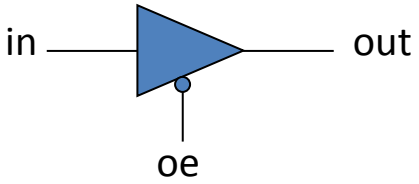
Verilog Basis & Primitive Cell Supported

- Verilog Basis
  - parameter declarations
  - wire, wand, wor declarations
  - reg declarations
  - input, ouput, inout declarations
  - continuous assignments
  - module instantiations
  - gate instantiations
  - always blocks
  - task statements
  - function definitions
  - for, while loop
- Synthesizable Verilog primitives cells
  - and, or, not, nand, nor, xor, xnor
  - bufif0, bufif1, notif0, notif1

HDL Compiler Unsupported

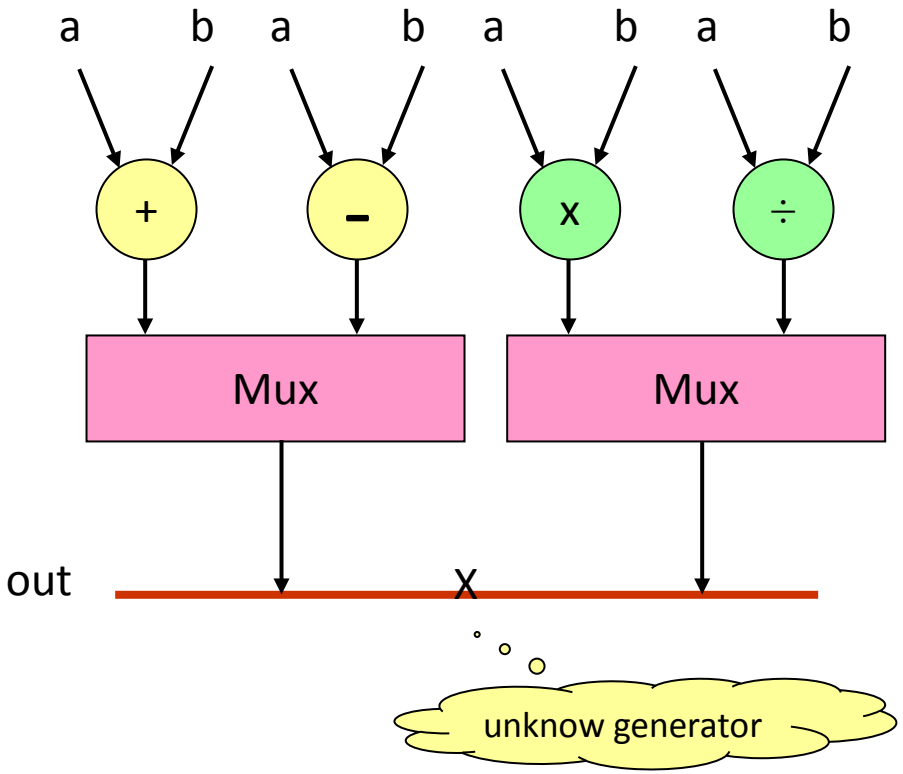
- delay
- initial
- repeat
- wait
- fork … join
- event
- deassign
- force
- release
- primitive -- User defined primitive
- time
- triand, trior, tri1, tri0, trireg
- nmos, pmos, cmos, rnmos, rpmos, rcmos
- pullup, pulldown
- rtran, tranif0, tranif1, rtranif0, rtranif1

example: wire out=(!oe)? in : ’hz;



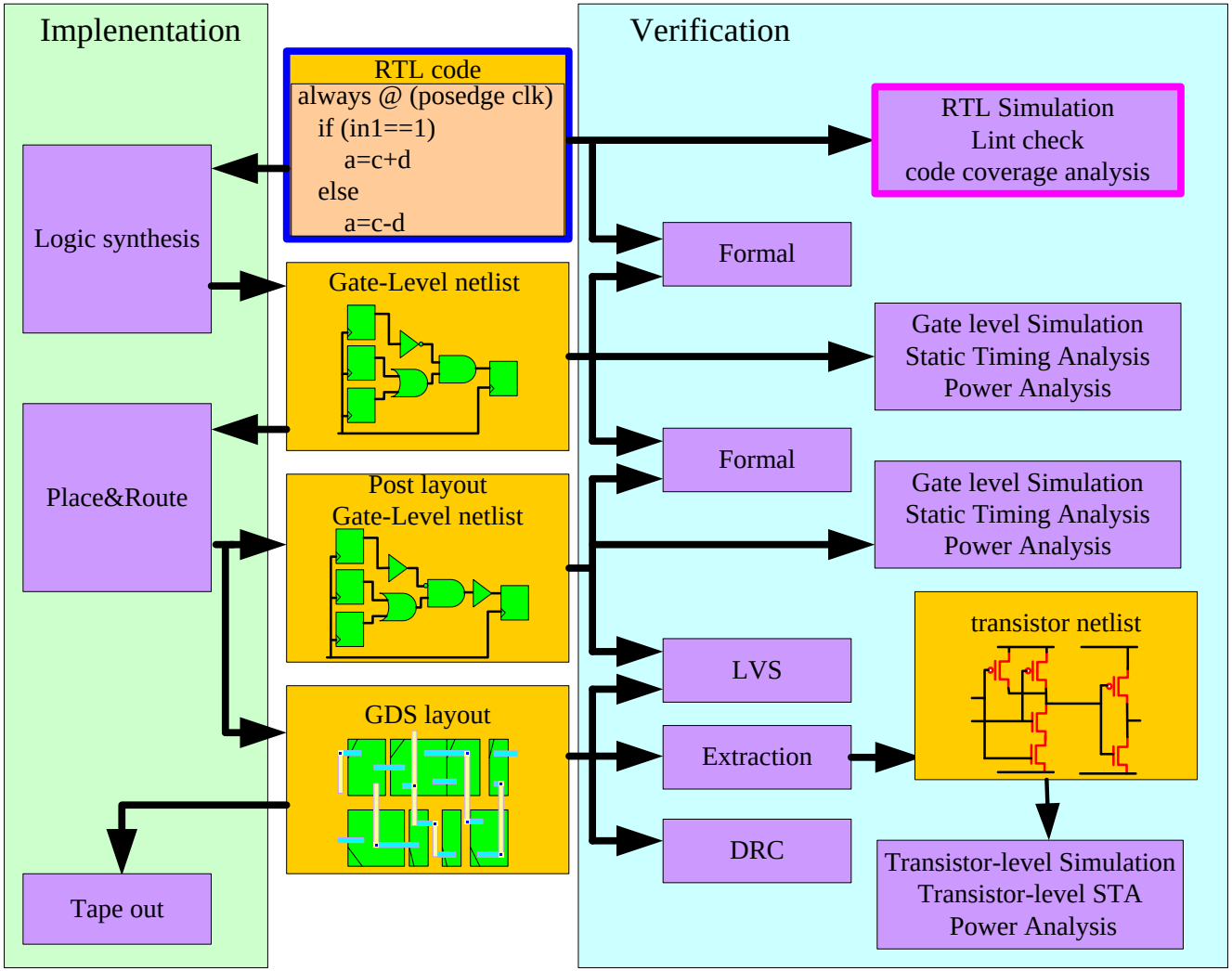
Example for Error always syntax

```
module top(clk, a, b, out, …);
output  out;
input   a, b, clk;
reg     out;
          :
          :
always  @(posedge clk)begin
   if(en1)    out=a+b;
   else       out=a-b;
end
          :
          :
always  @(posedge clk)begin
   if(en2)    out=a*b;
   else       out=a/b;
end
          :
endmodule
```



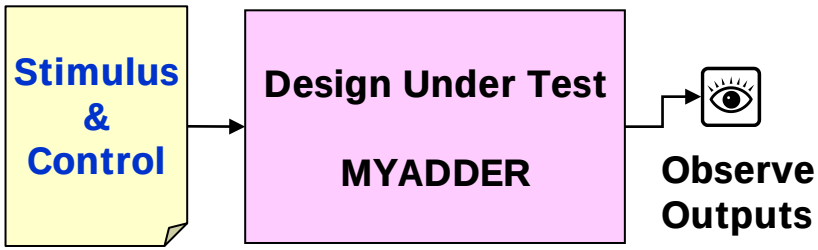
Synthesis Unsupported

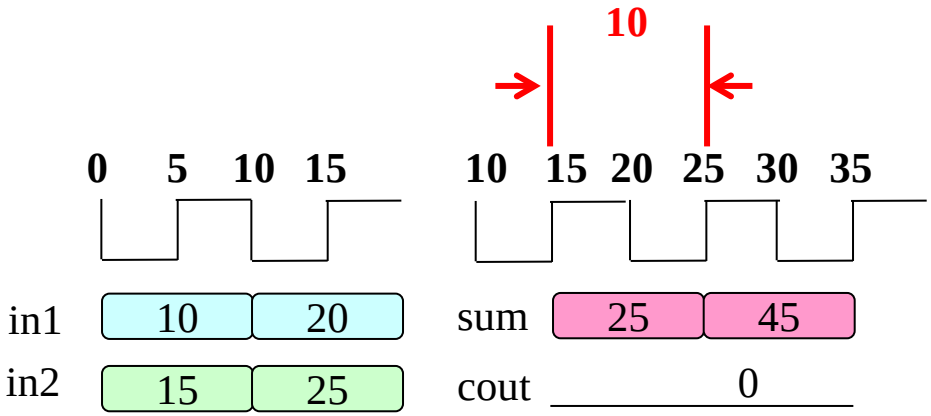
RTL Verification



RTL Verification

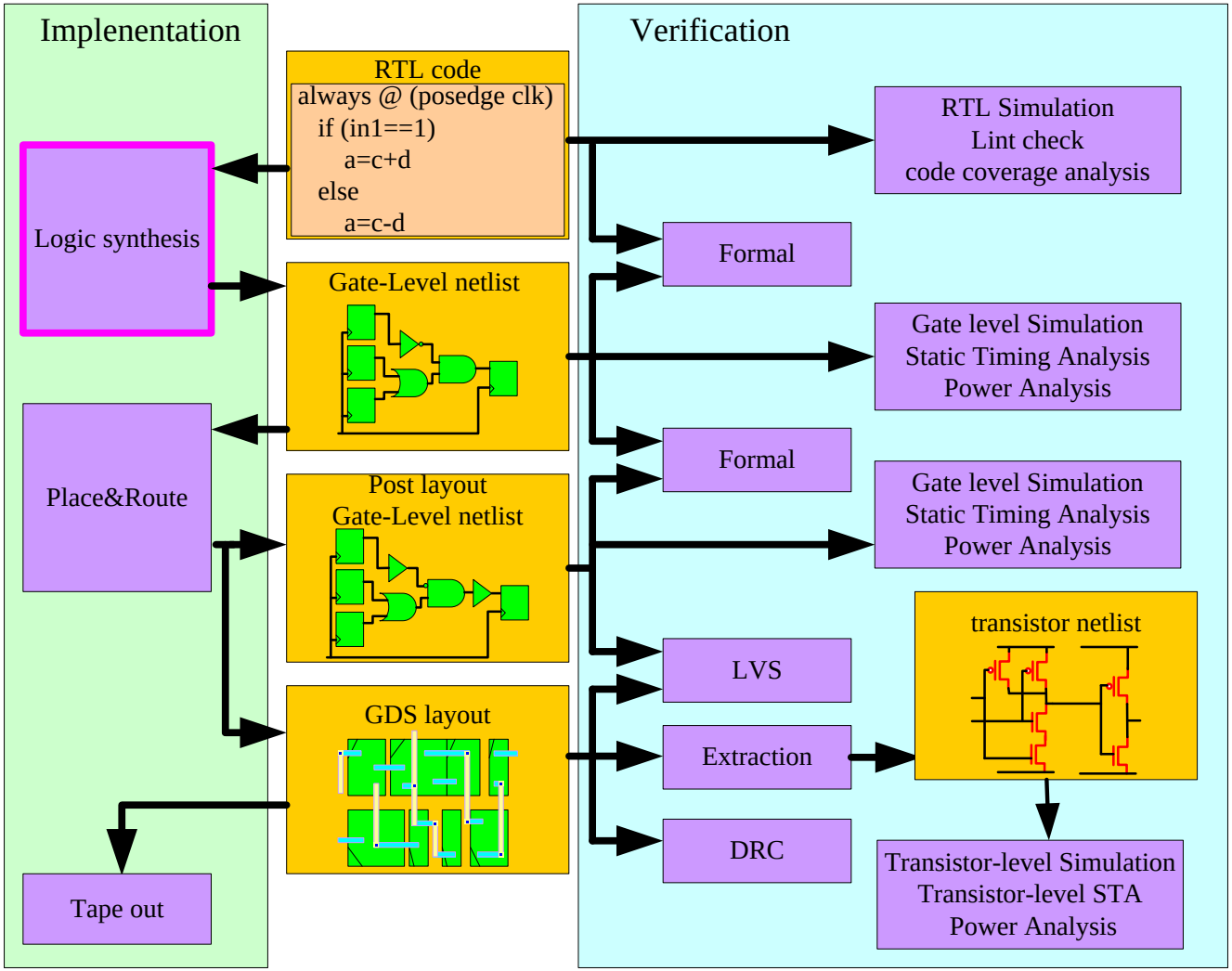
- RTL Simulation






```
module test;

reg clock;   reg [31:0] in1, in2;
wire [31:0] sum;   wire cout;

MYADDER U1 (clock, in1, in2, sum, cout);

always #5 clock = ~clock;

initial begin
        clock = 0;
        in1 = 10; in2 = 15;
        @(negedge clock)
        in1 = 20; in2 = 25;
        @(negedge clock)
         $finish;
      end

endmodule
```

Cell-Based IC Design Flow - Logic Synthesis



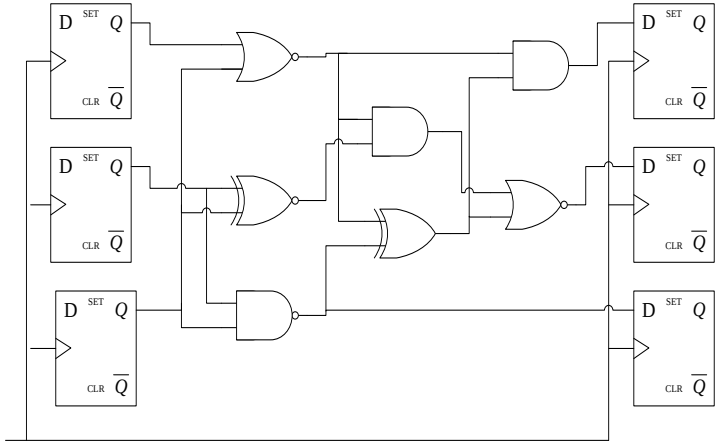
Logic Implementation

- Logic Synthesis
- Design for Testability (DFT)
  - Memory BIST
  - Scan Insertion

What is Synthesis? (1/2)

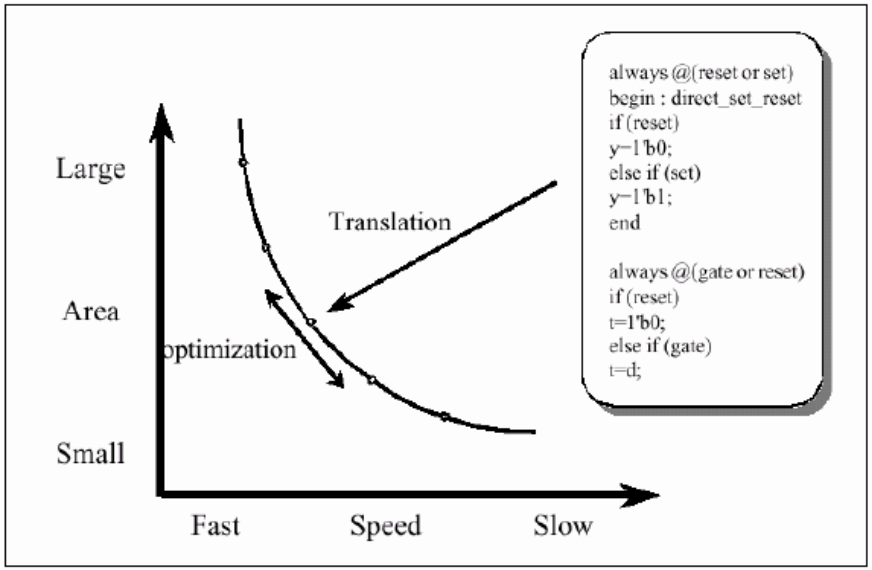
- Synthesis = translation + optimization

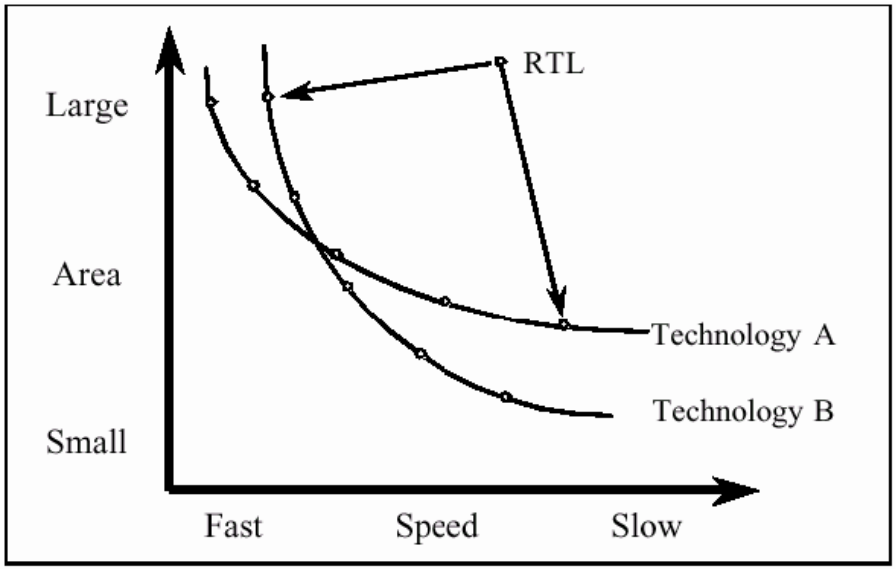
```
always @ (posedge clk)
begin
   if (reset) begin
       a=0;
       ……..
    end
    else  begin
       a=c+d*b;
        …….
    end
end
```



What is Synthesis? (2/2)

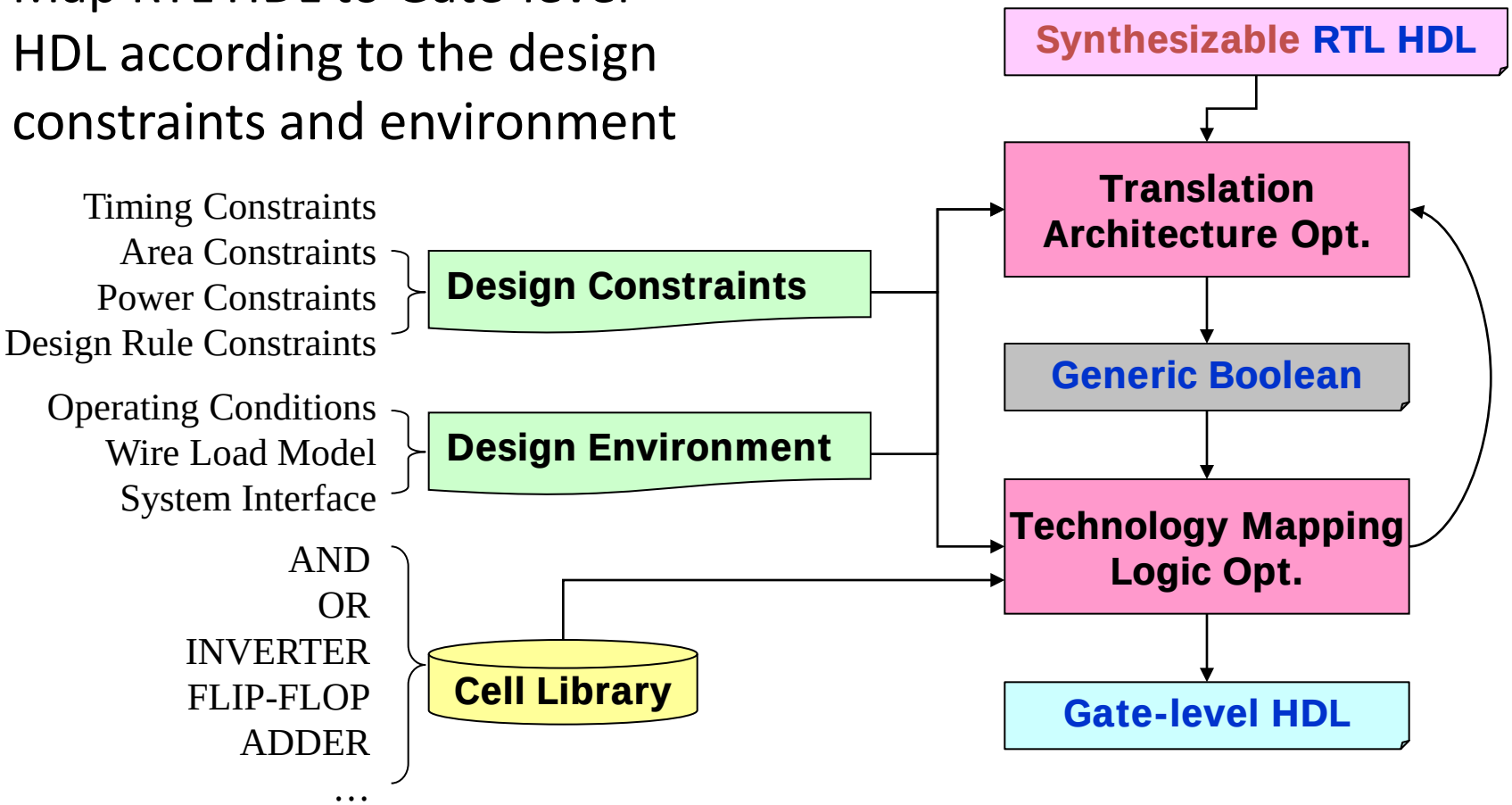
- Synthesis is constraint driven
- Technology independent



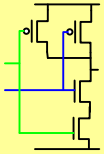
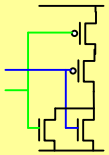
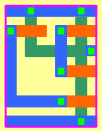
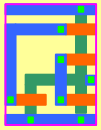
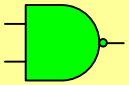


Logic Synthesis

- Map RTL HDL to Gate-level HDL according to the design constraints and environment

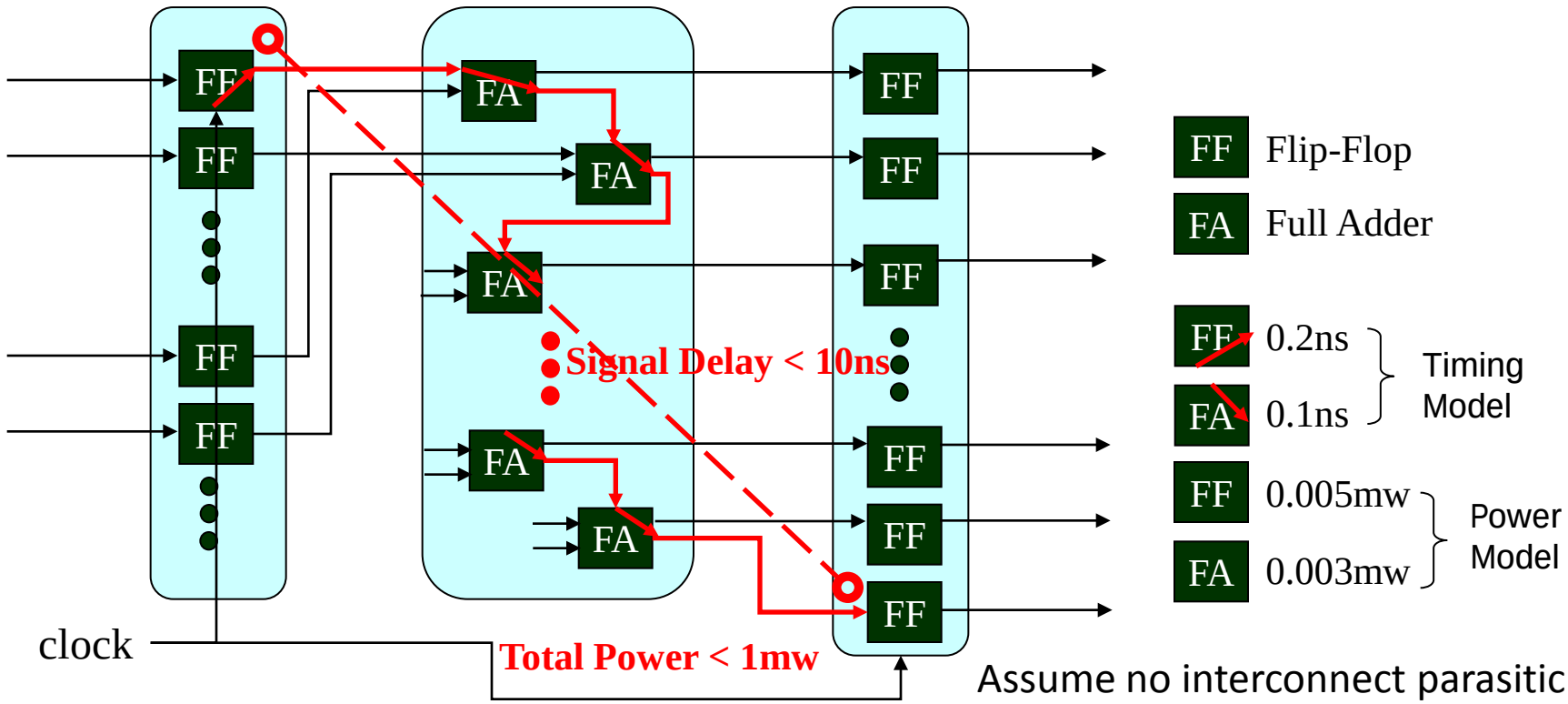


Cell Library

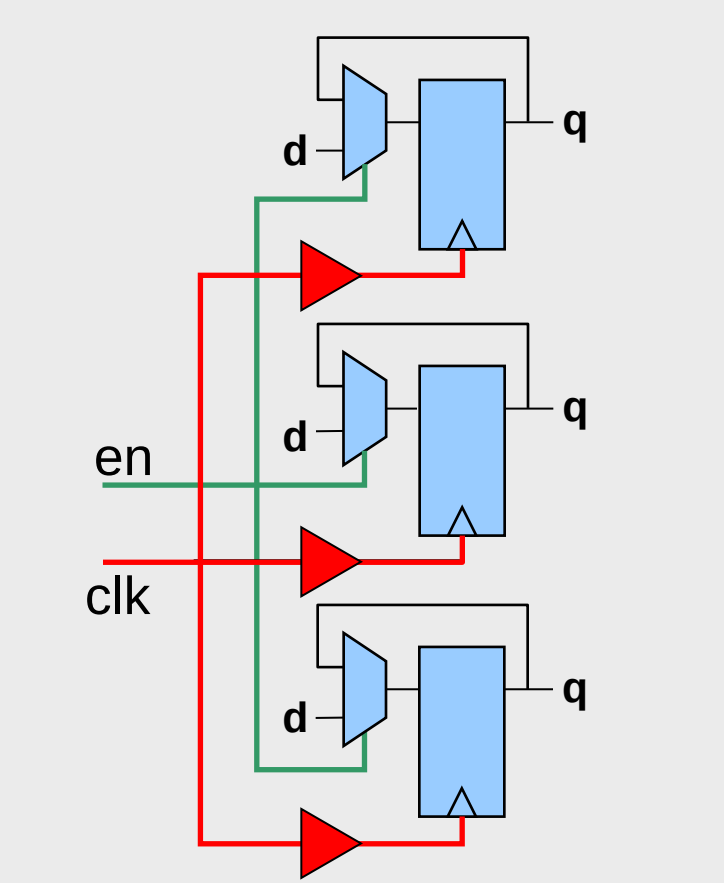
function	NAND	NOR	XOR	INV	ADD	FF	•••••••
schematic							
layout							
symble			○○○○○○	○○○○○○	○○○○○○	○○○○○○	••••••
timing	A1→O 0.1ns A2→O 0.2ns	A1→O 0.1ns A2→O 0.2ns					
power	A1→O 0.1pw A2→O 0.2pw	A1→O 0.1pw A2→O 0.2pw					
abstract							

Design Example

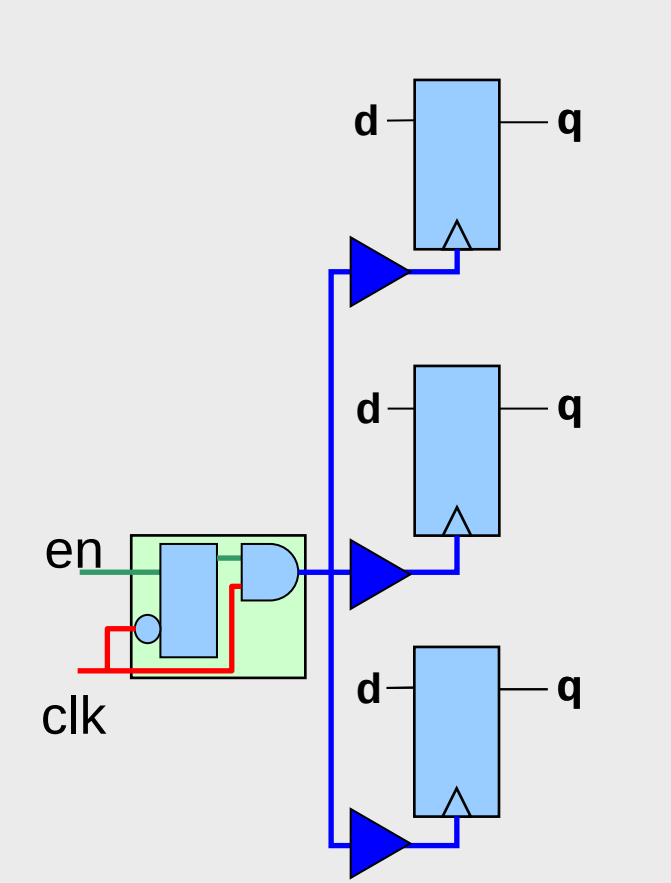
- Translation: Ripple Adder
- Technology Mapping: Meet timing & power constraints



Clock Gating Reduces Both Power and Area



No Clock Gating
en=0 consumes power



Clock Gating
reduces power when en=0

Clock gate reduces switching activity on the clock tree

Clock Gating Coding Style

If statement

```
always@ (posedge clk)
    if (enable) Q <= D_in;
```

If + Loop statement

```
always @ (posedge clk)
  if (enable)
    for (i=0; i<8; i=i+1)
      s[i] = a[i] ^ b[i];
```

Conditional Assignment

```
always@ (posedge clk)
    Q <=(enable)? D_in : Q;
```

Case statement

```
always@ (posedge clk)
    case (enable)
        1'b1: Q  <=  D_in;
        1'b0: Q  <=  Q;
    endcase
```

For Loop

- Provide a shorthand way of writing a series of statements
- In synthesis, for loops loops are "*unrolled*", and then synthesized
- Example

```
always @(a or b) begin
  for( k=0; k<=3; k=k+1 ) begin
            out[k]=a[k]^b[k];
            c=(a[k]|b[k])&c;
  end
end
```

→

```
out[0] = a[0]^b[0];
out[1] = a[1]^b[1];
out[2] = a[2]^b[2];
out[3] = a[3]^b[3];
c = (a[0] | b[0]) & (a[1] | b[1]) &
    (a[2] | b[2]) & (a[3] | b[3]) & c
```

Netlist after Synthesis

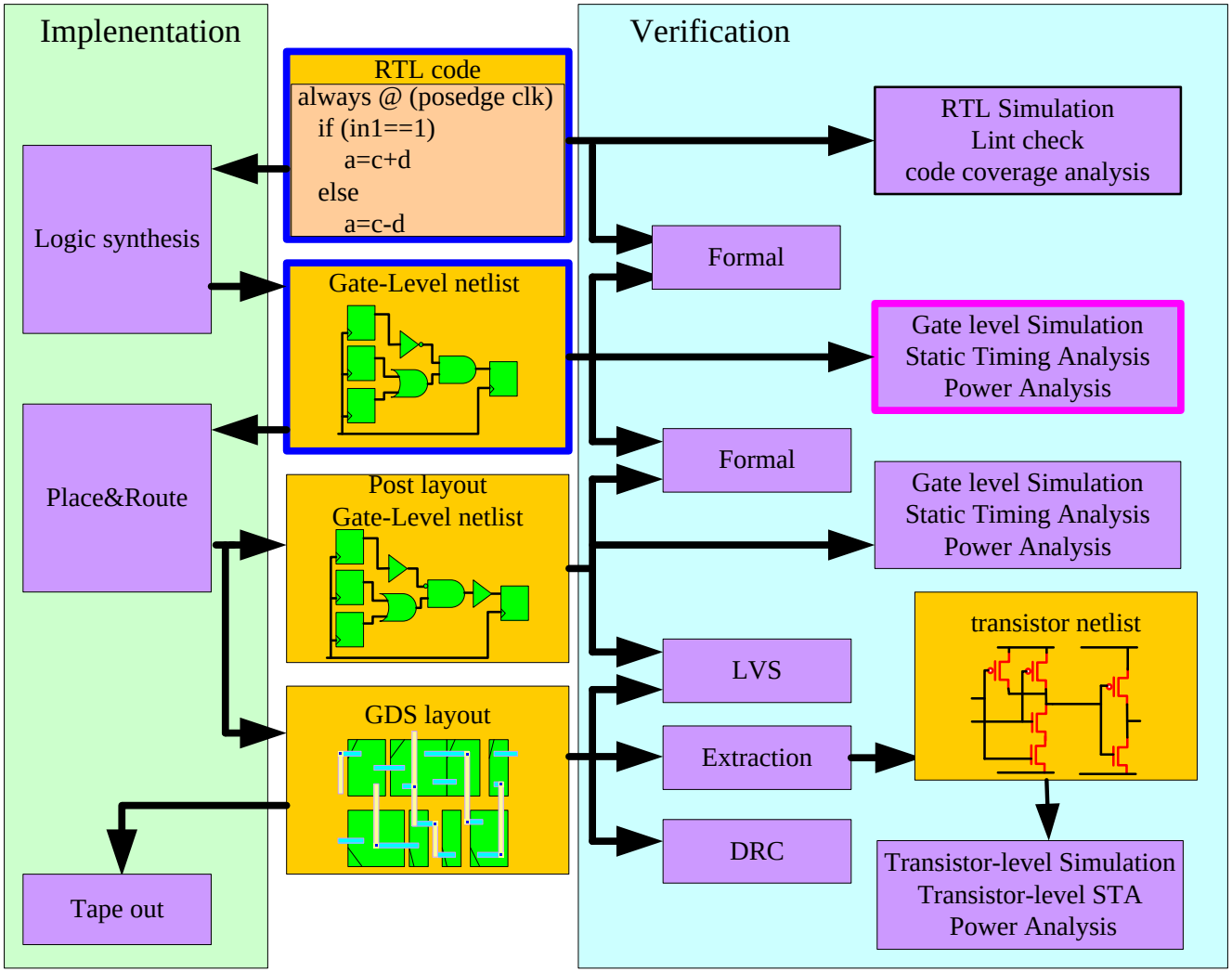
- RTL

```
module MYADDER (clock, in1, in2, sum, cout);

input clock;   input [31:0] in1, in2;
output [31:0] sum;   output cout;

reg [31:0] a, b;   reg [31:0] sum;

always @(posedge clock)
        begin
          a = in1; b = in2; {cout, sum} = a + b;
        end

endmodule
```

- Gate Level

```
module MYADDER (clock, in1, in2, sum, cout);

input clock;   input [31:0] in1, in2;
output [31:0] sum;   output cout;

DFCNQD1 11_ (.CP ( cki ) , .D ( n_22 )) ;
MOAI22D0 g301.A2 ( ck_down_12_ ) ) ;
IND2D1 g300 ( ck_down_13_ ), .ZN ( n_25 ) ) ;
DFCNQD1 ck_down_reg_12_(.D ( n_24 )) ;
MOAI22D0
...
```

Pre-layout Gate-level Verification



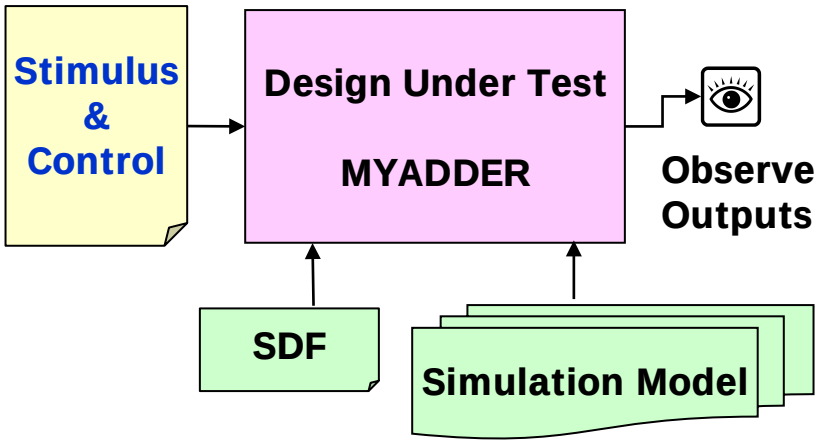
NARLabs

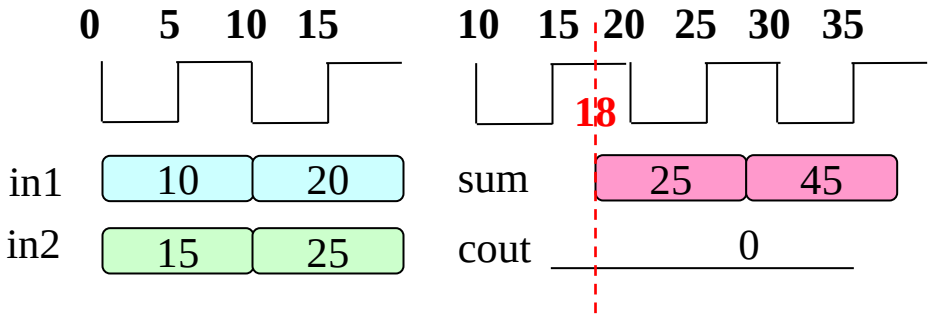
Pre-layout Gate-level Verification

- Function Verification
  - Pre-layout Gate-level Simulation
- Timing Verification
  - Pre-layout Gate-level Simulation
  - Pre-layout Static Timing Analysis (STA)
- Power Verification
  - Power Analysis

Pre-layout Gate-level Simulation

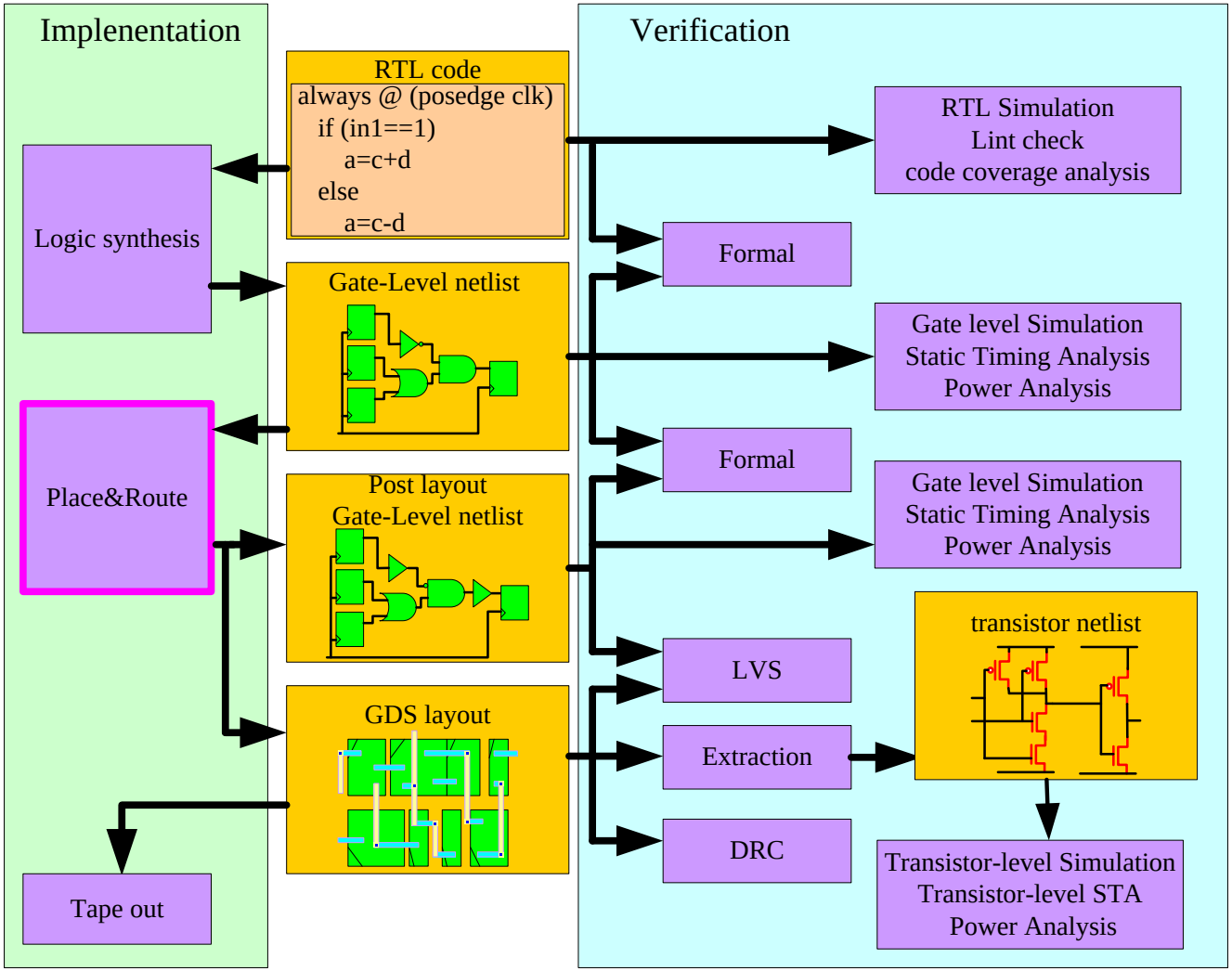
- SDF: Standard Delay Format



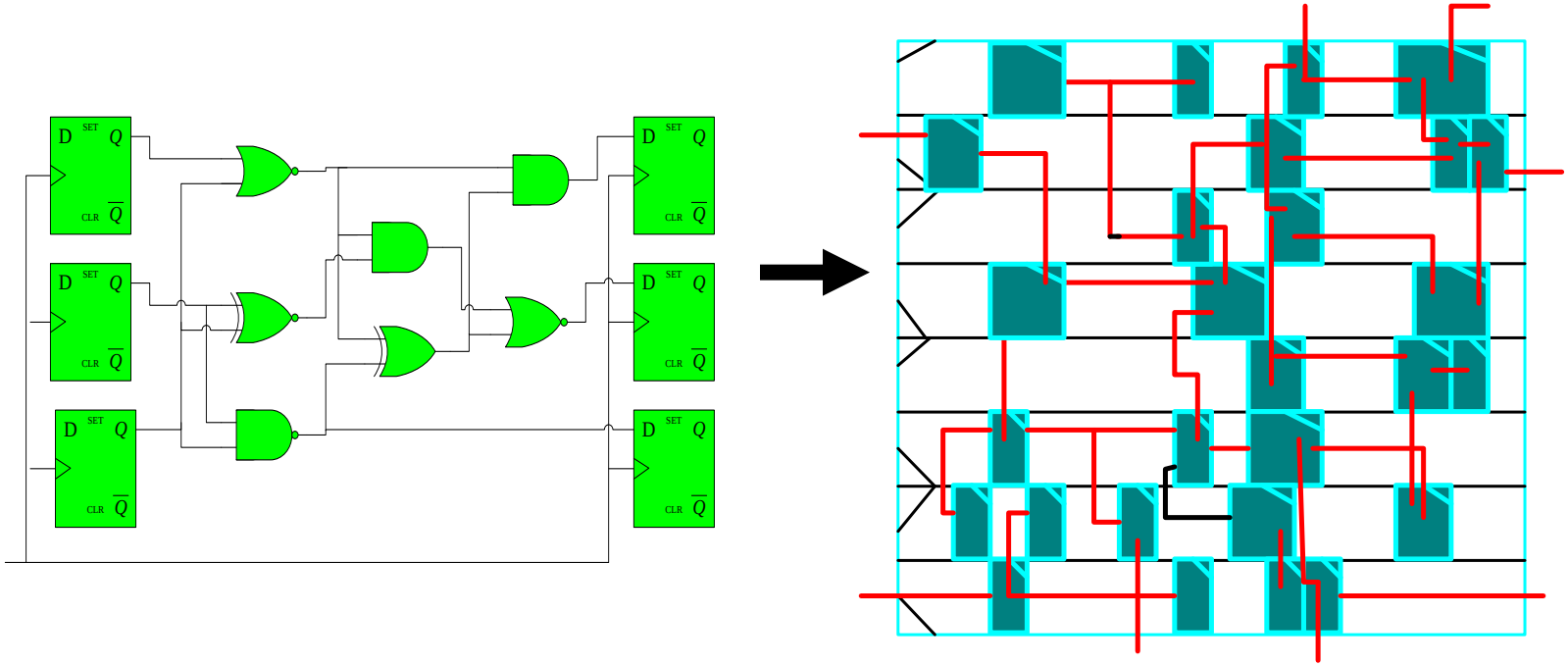



```
module test;

reg clock;   reg [31:0] in1, in2;
wire [31:0] sum;   wire cout;

MYADDER U1 (clock, in1, in2, sum, cout);

always #5 clock = ~clock;

initial $sdf_annotate("my.sdf",U1);
initial begin
        clock = 0;
        in1 = 10; in2 = 15;
        @(negedge clock)
        in1 = 20; in2 = 25;
        @(negedge clock)
         $finish;
      end

endmodule
```

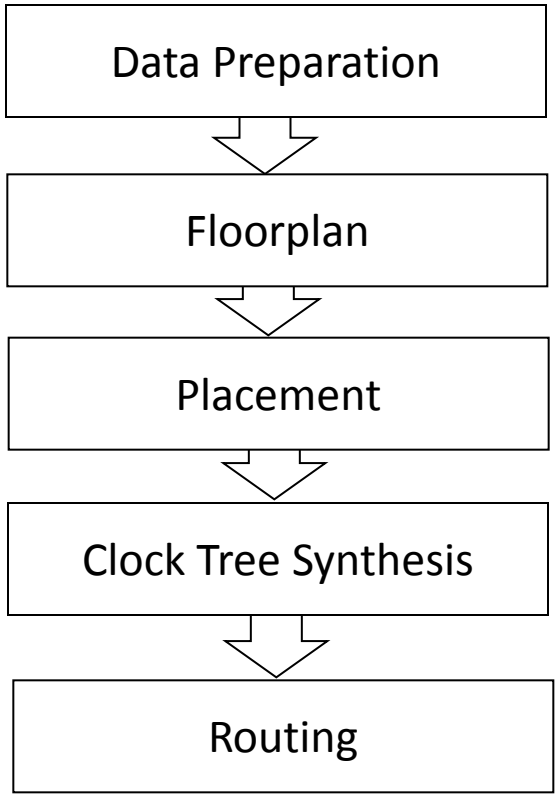
Cell-Based IC Design Flow - Place & Route



Physical Implementation



Physical Implementation

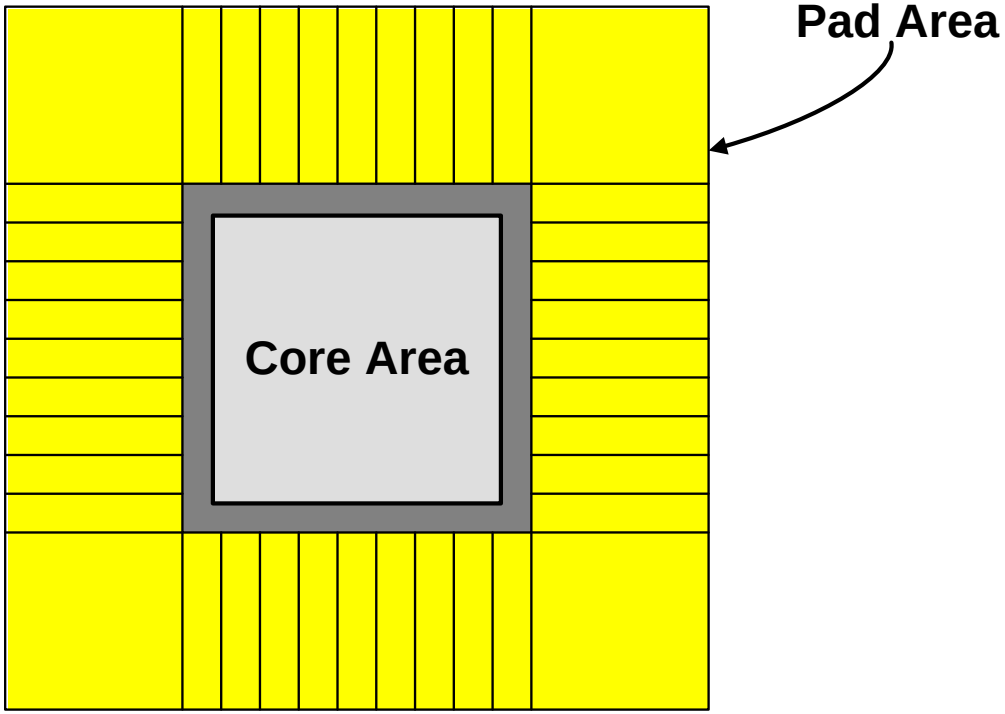


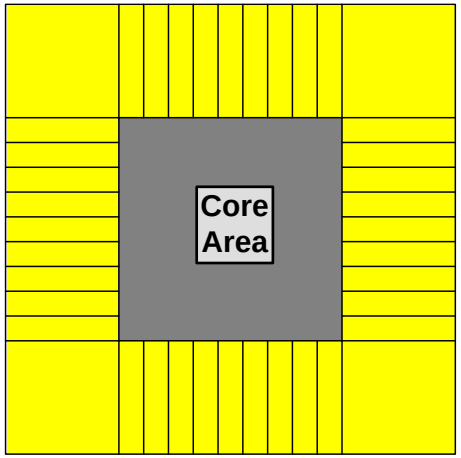
NARLabs

Data Preparation

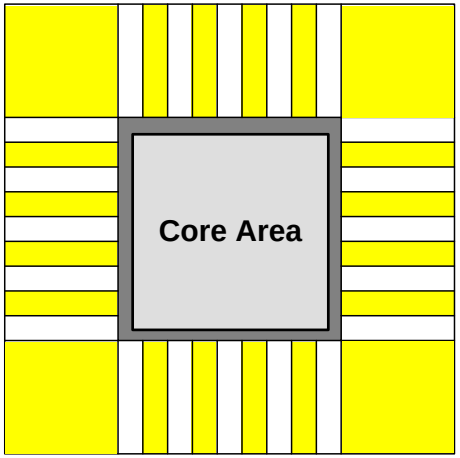
- Verified Pre-layout Gate-level HDL
- Timing/Power Constraints
  - Similar to those for logic synthesis
- Layout Constraints
  - Chip Size / Aspect Ratio / IO Constraints
  - Physical Partitions (Region / Group)
  - Macro Positions…
- Cell Libraries

Floorplan Area





Pad Limit



Core Limit

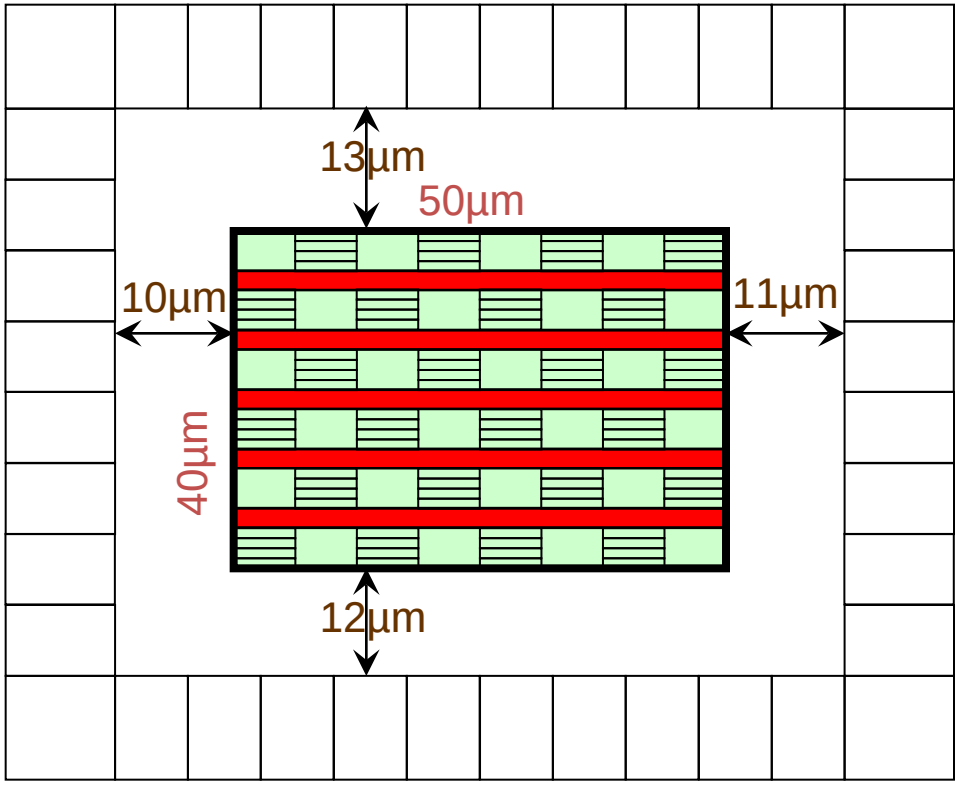
Floorplan - Parameters

Row
(Area：**1600** μm^2)

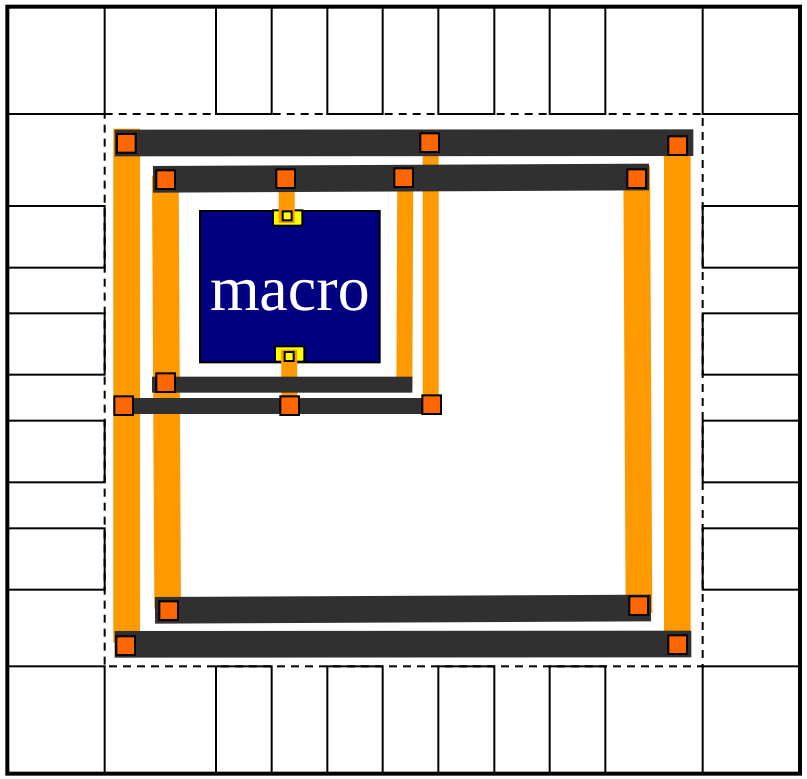
Channel
(Area：**400** μm^2)

Standard Cell
(Area：**800** μm^2)

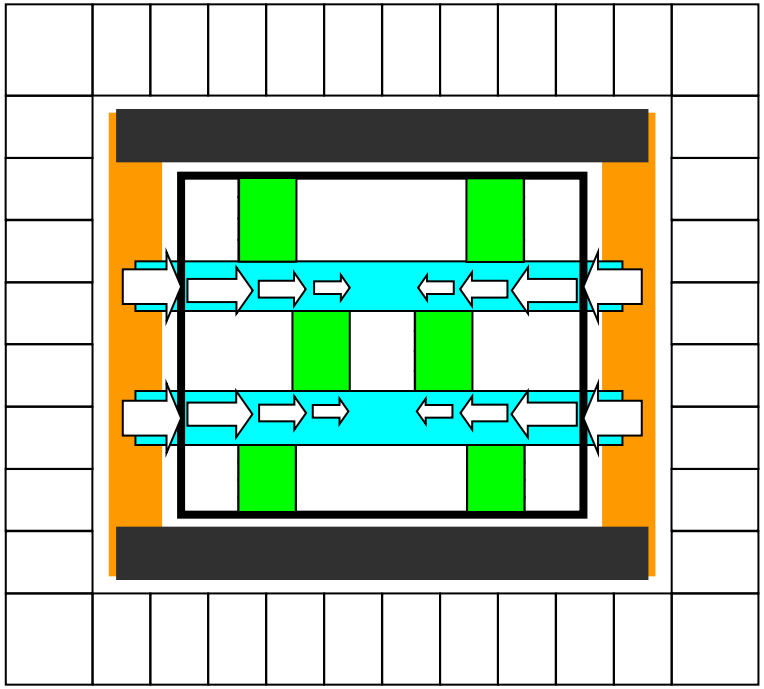
Aspect ratio: 40/50 = 0.8
Core Utilization: 800/2000 = 0.4
Core to Left: 10
Core To Right: 11
Core To Bottom: 12
Core To Top: 13



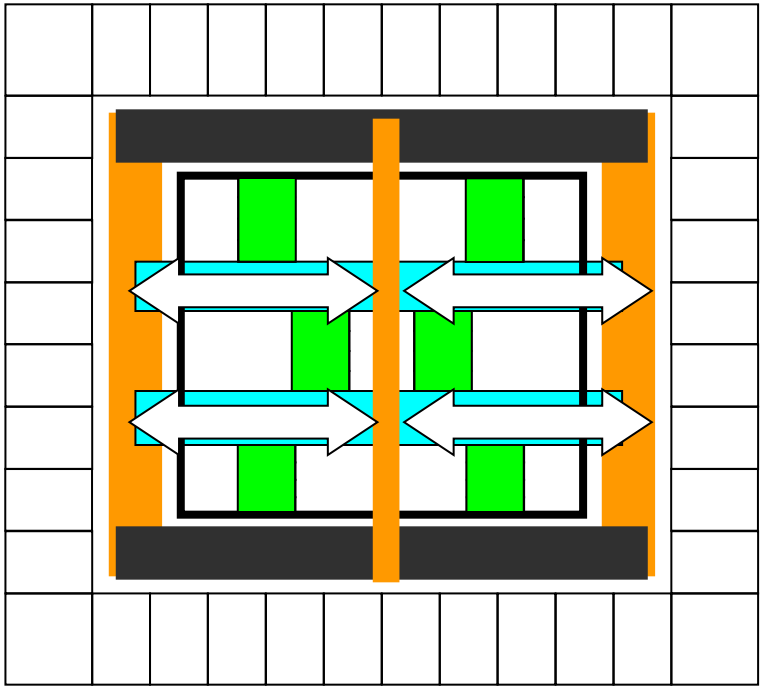
Floorplan - Power Mesh (Ring)



Floorplan - Power Mesh (Strap)

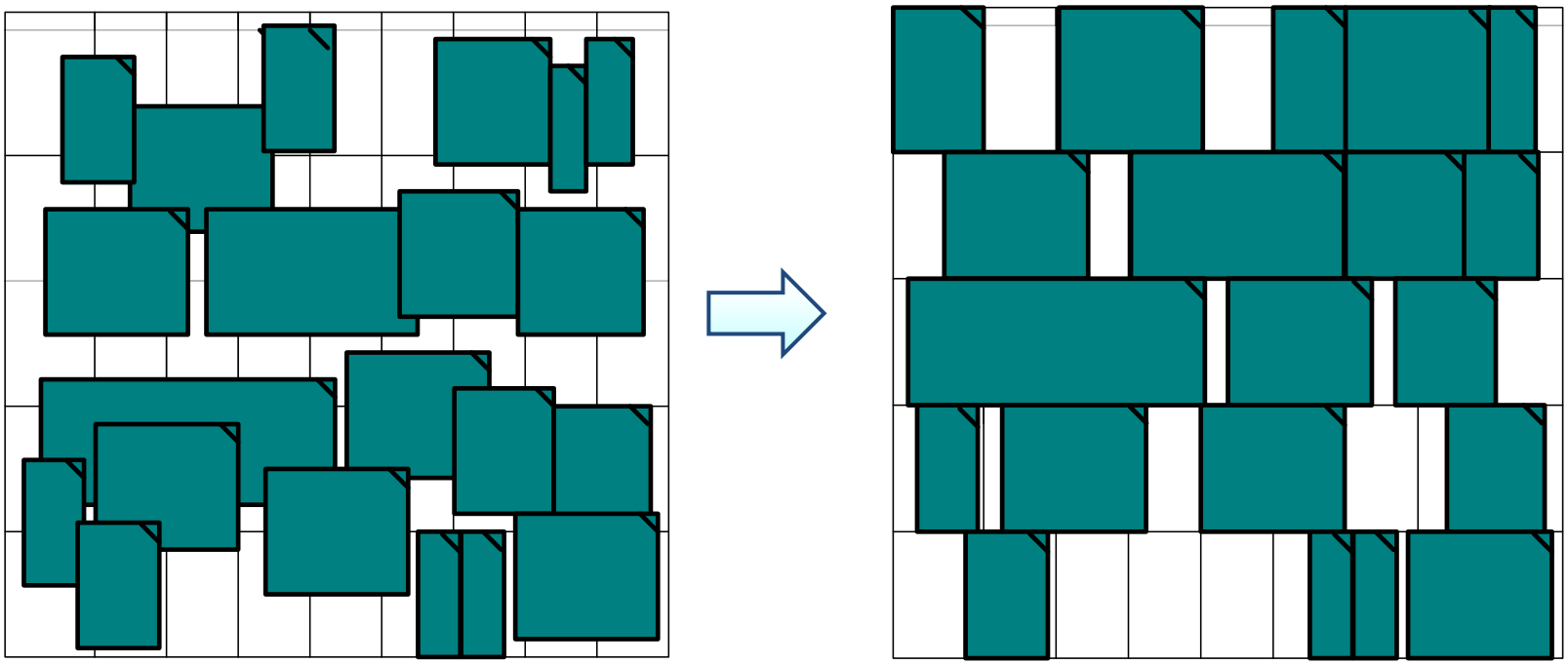


without strap (or stripe)



with strap (or stripe)

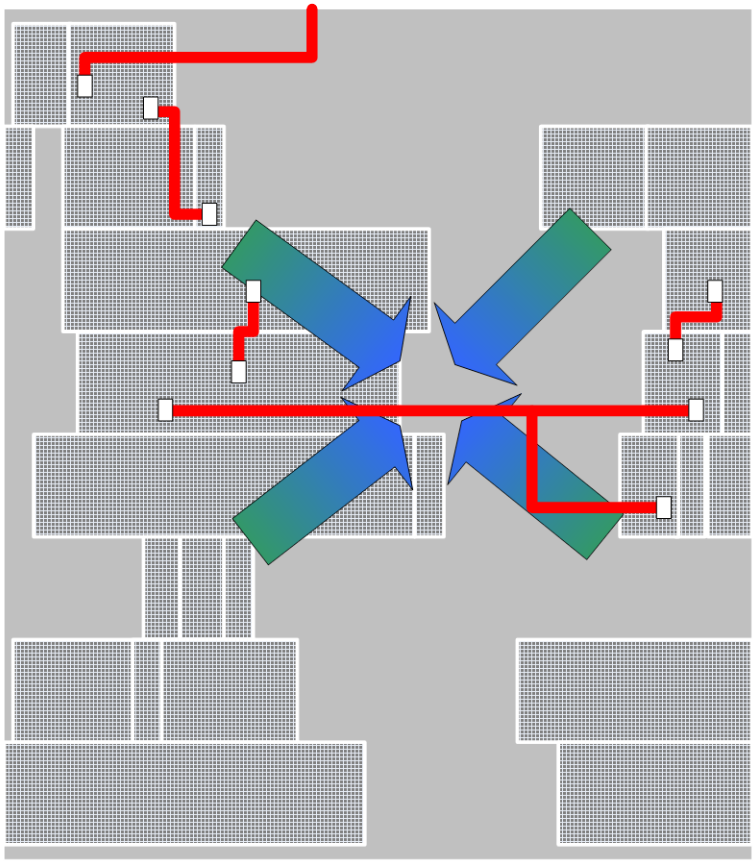
“Place” the Standard Cell



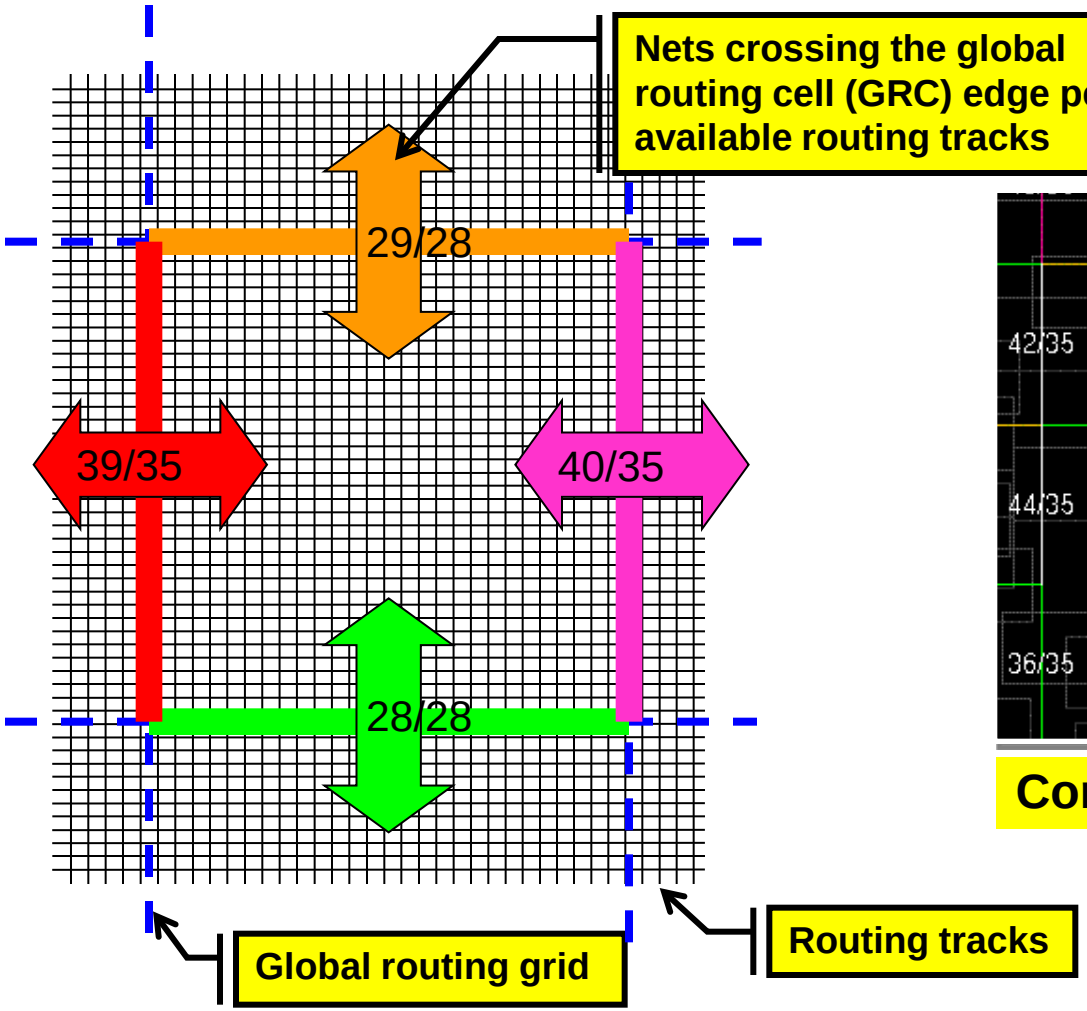
- Timing driven
- Congestion driven

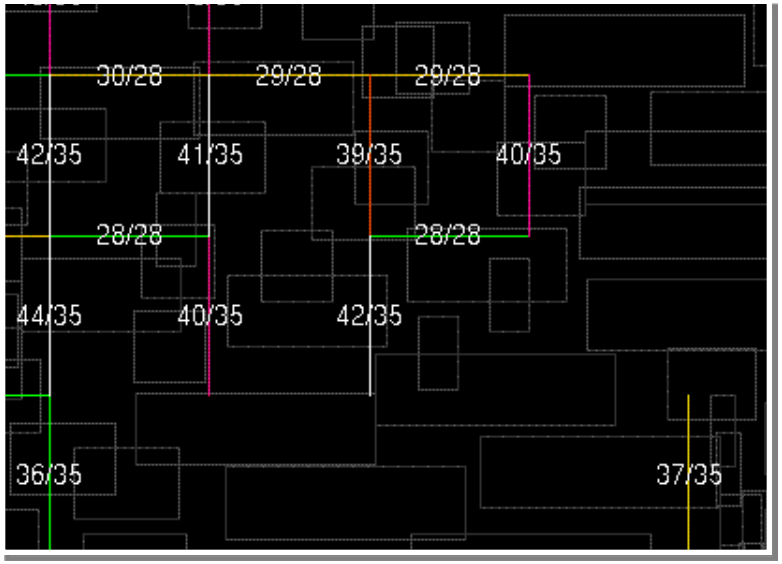
Timing-Driven Placement

- Timing-driven placement tries to *place cell along timing-critical path close together* to reduce net RCs and meet setup timing



Understanding the Congestion Calculation

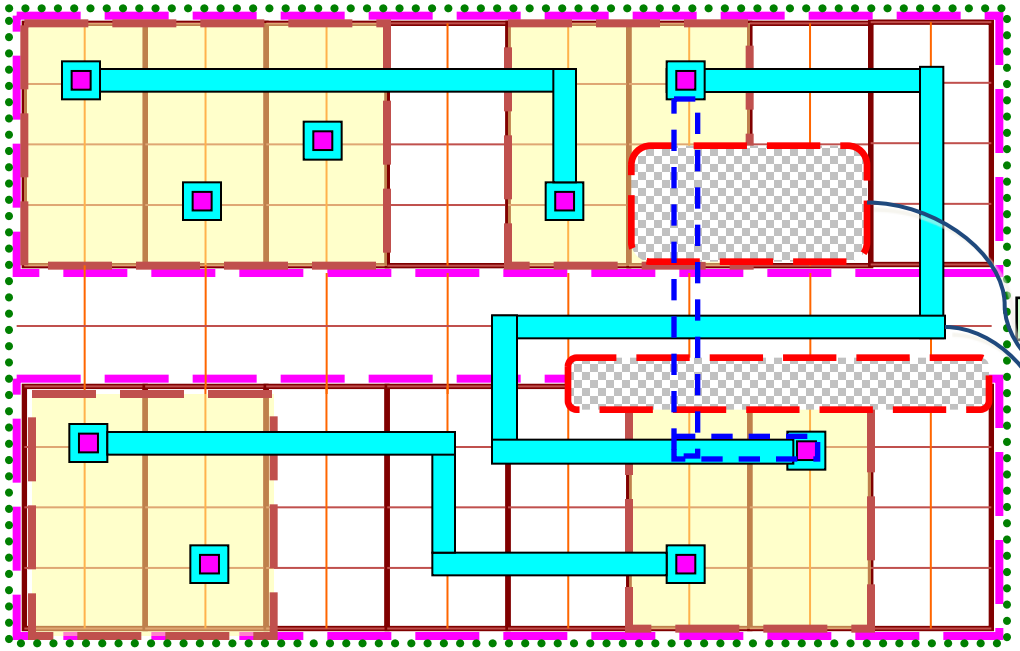




Congestion map (heat map)

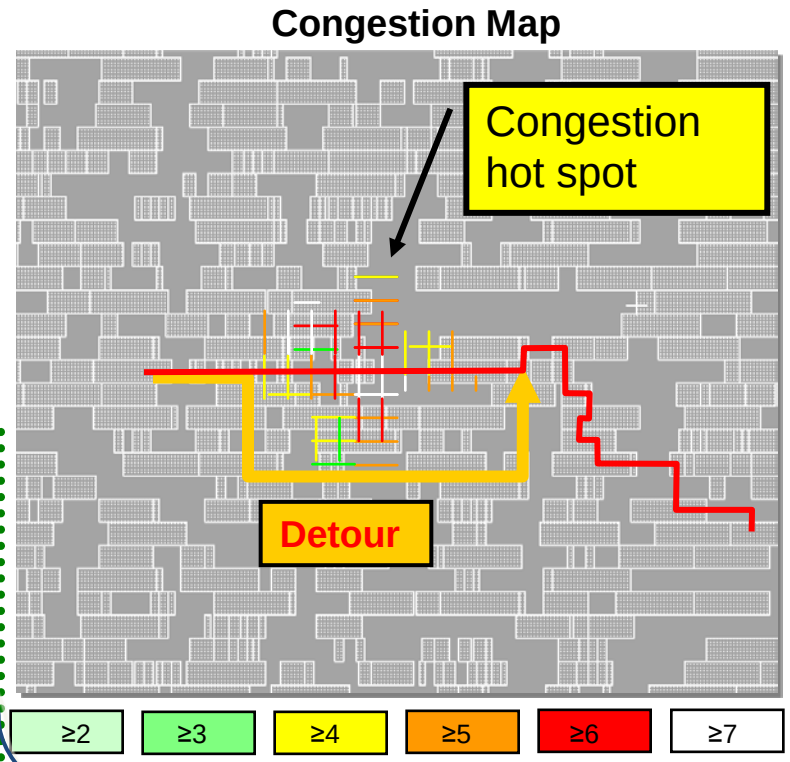
One Problem with Congestion…

- If congestion is not too severe, the actual route can be detoured around the congested area
- However, the detoured nets will have worse RC delay.



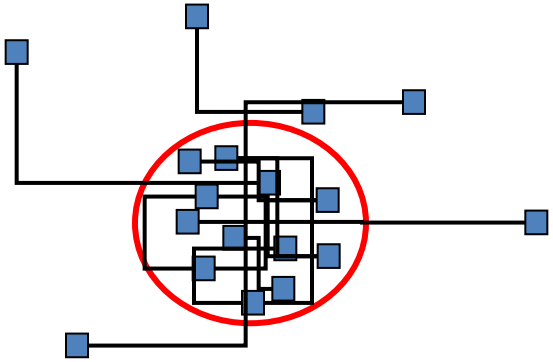
congestion area

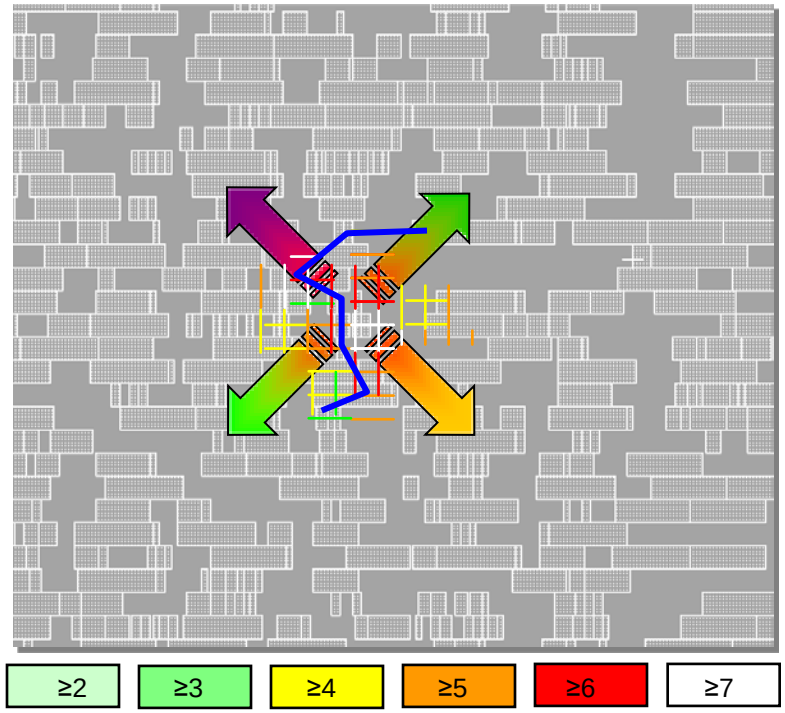
detoured net



What does Congestion-Driven Placement do?

Spreads apart cells that contribute to high congestion.



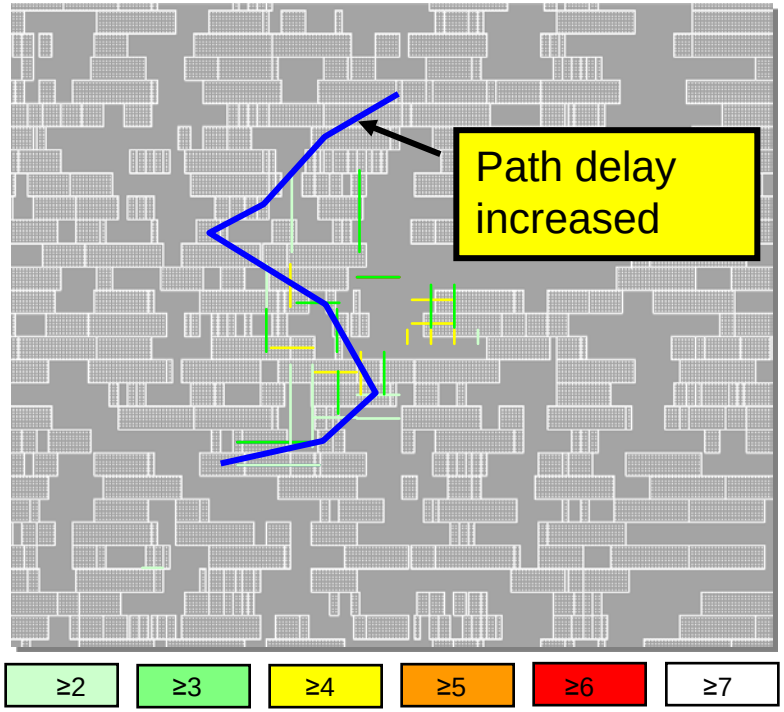


What happens to timing when the connected cells are moved apart?

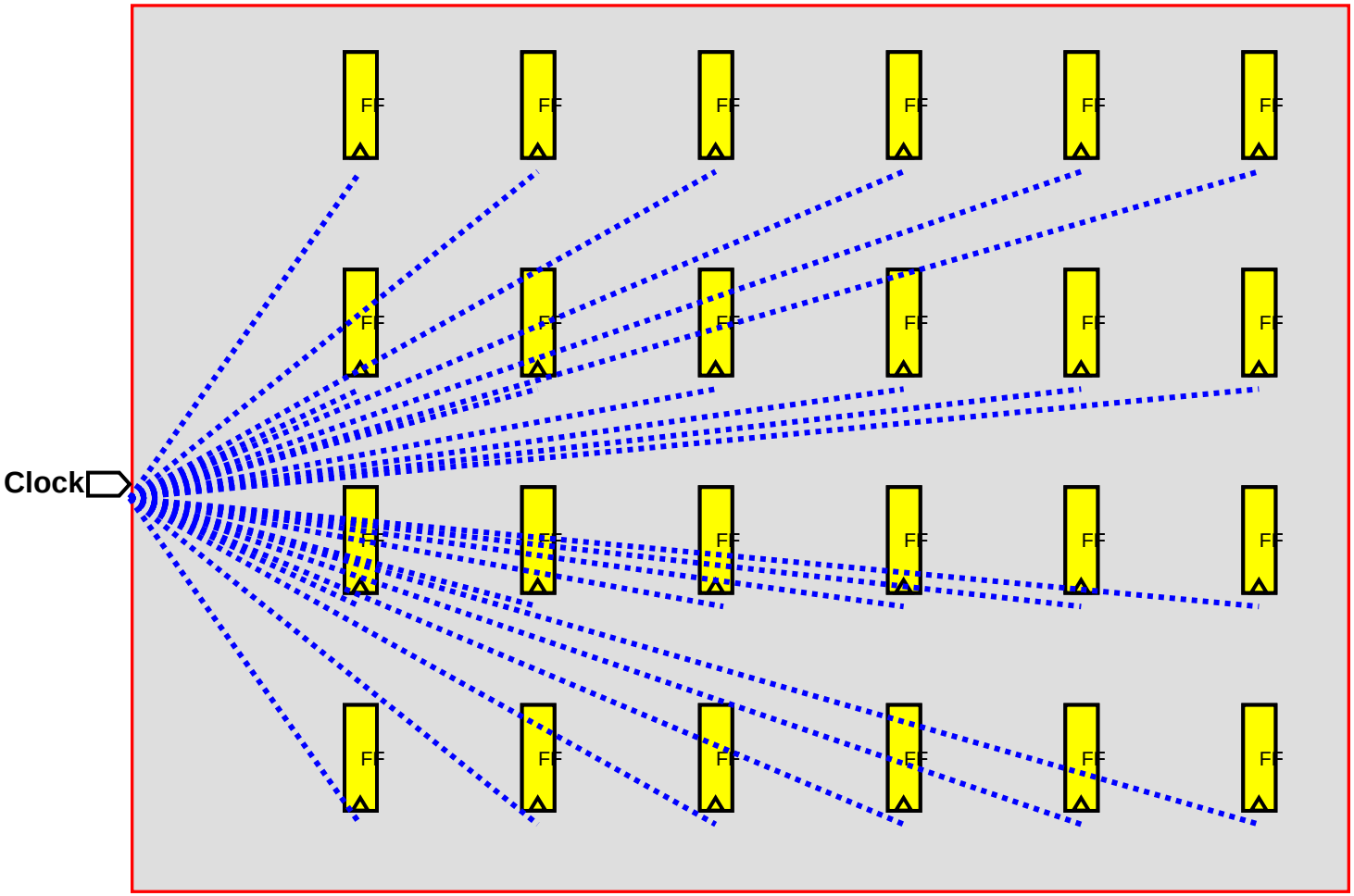
Congestion vs. Timing Driven Placement

CONGESTION ⟷ **TIMING**

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

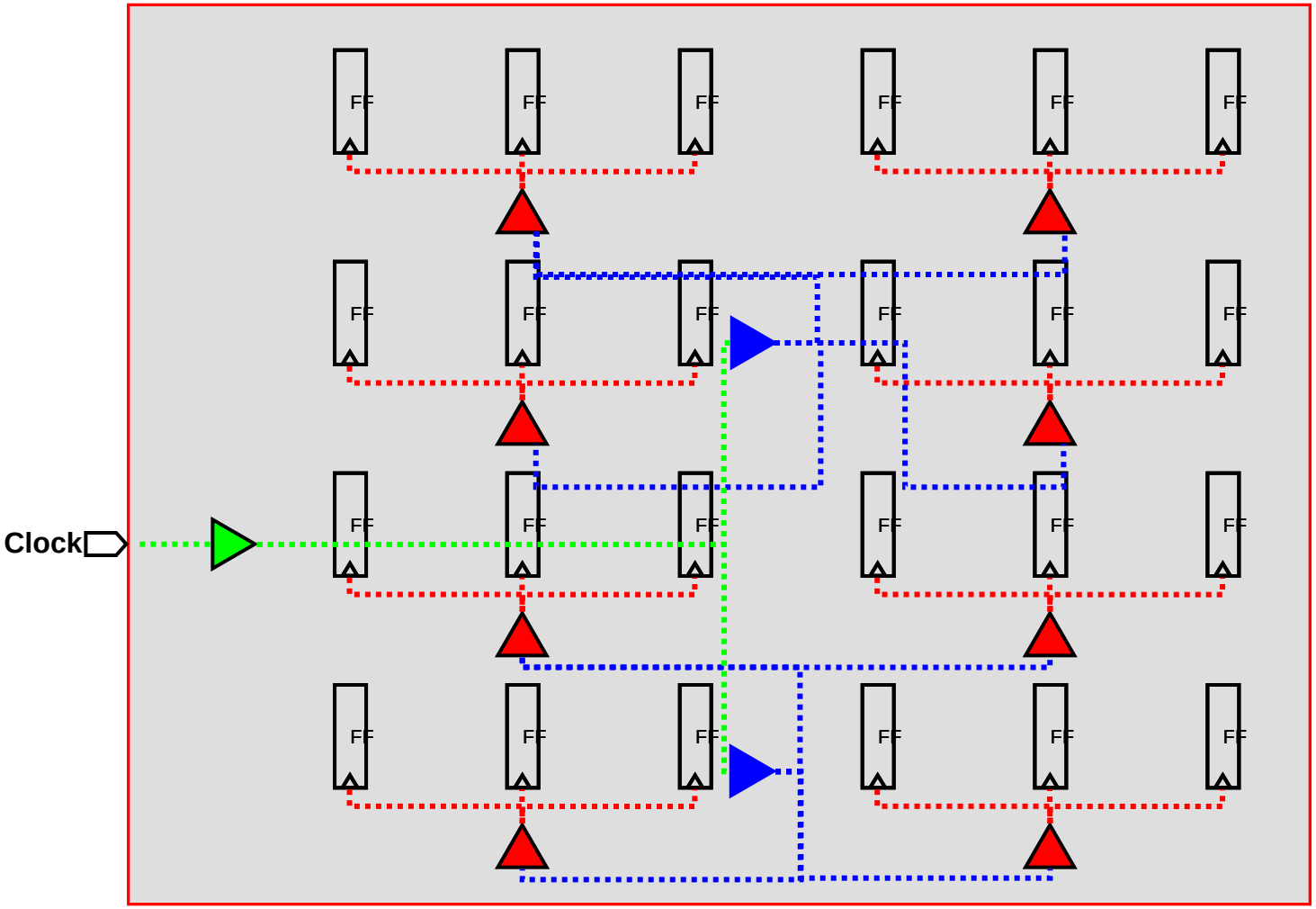


Starting Point before CTS



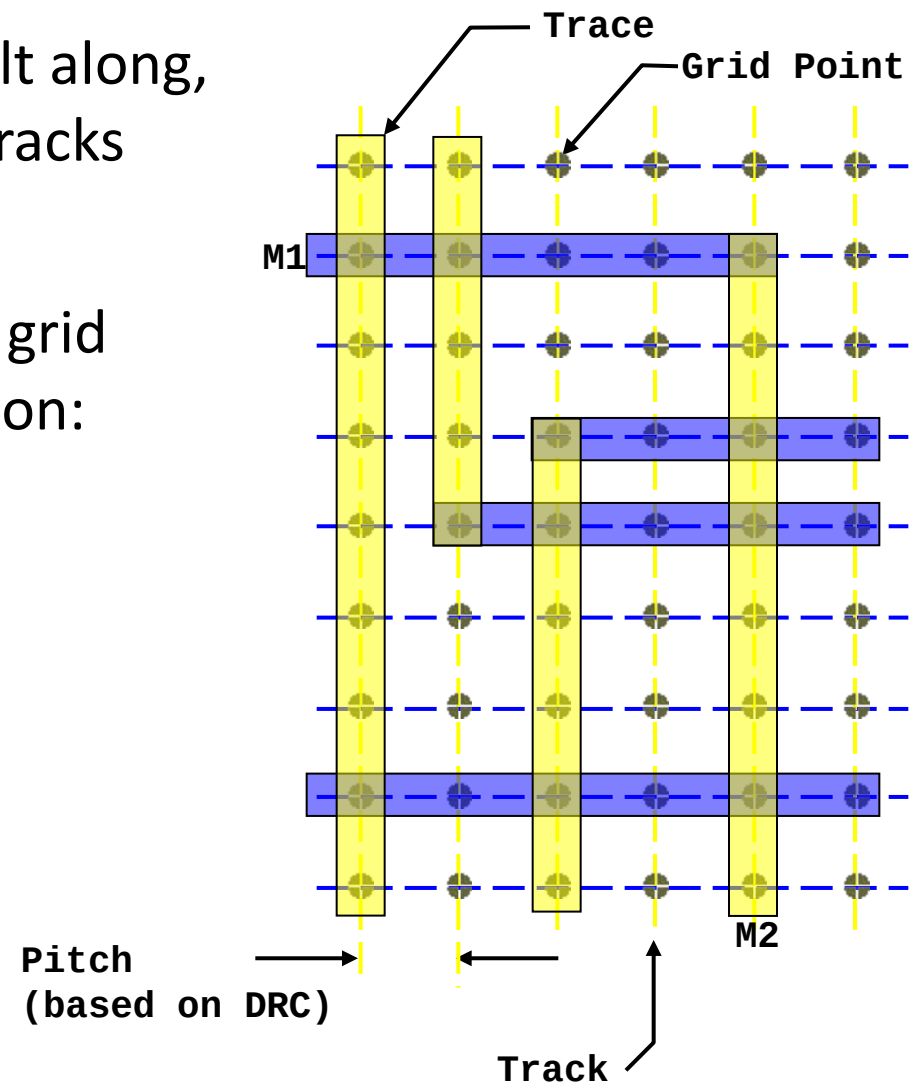
All clock pins are driven by a single clock source.

CTS for Design Example



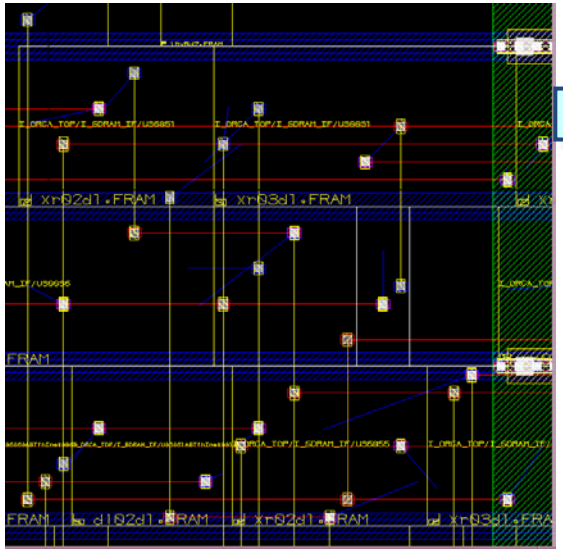
Grid-based Routing System

- Metal traces (routes) are built along, and centered upon routing tracks based on a grid.
- Each metal layer has its own grid and preferred routing direction:
  - VHV
    - M1: Vertical
    - M2: Horizontal, etc…
  - HVH
    - M1: Horizontal
    - M2: Vertical, etc…



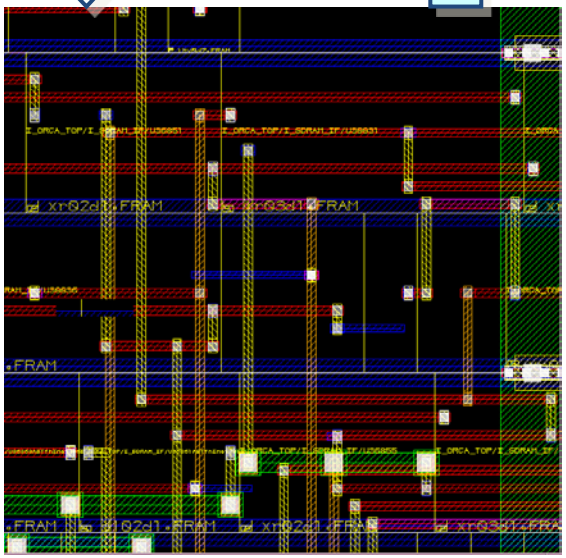
Routing for Design Example

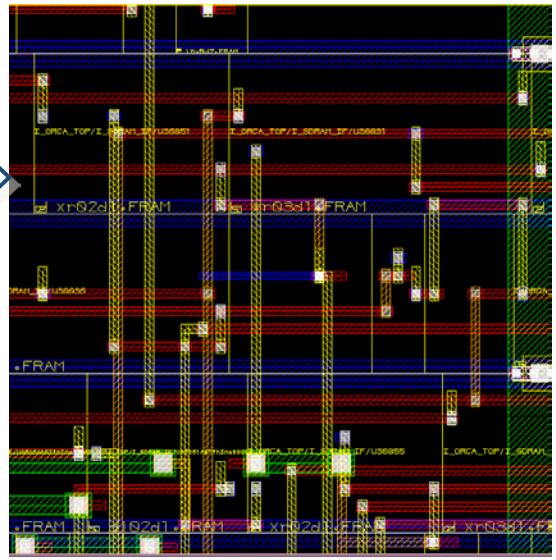
- After three stage all of the net connection with metal.



Global Route

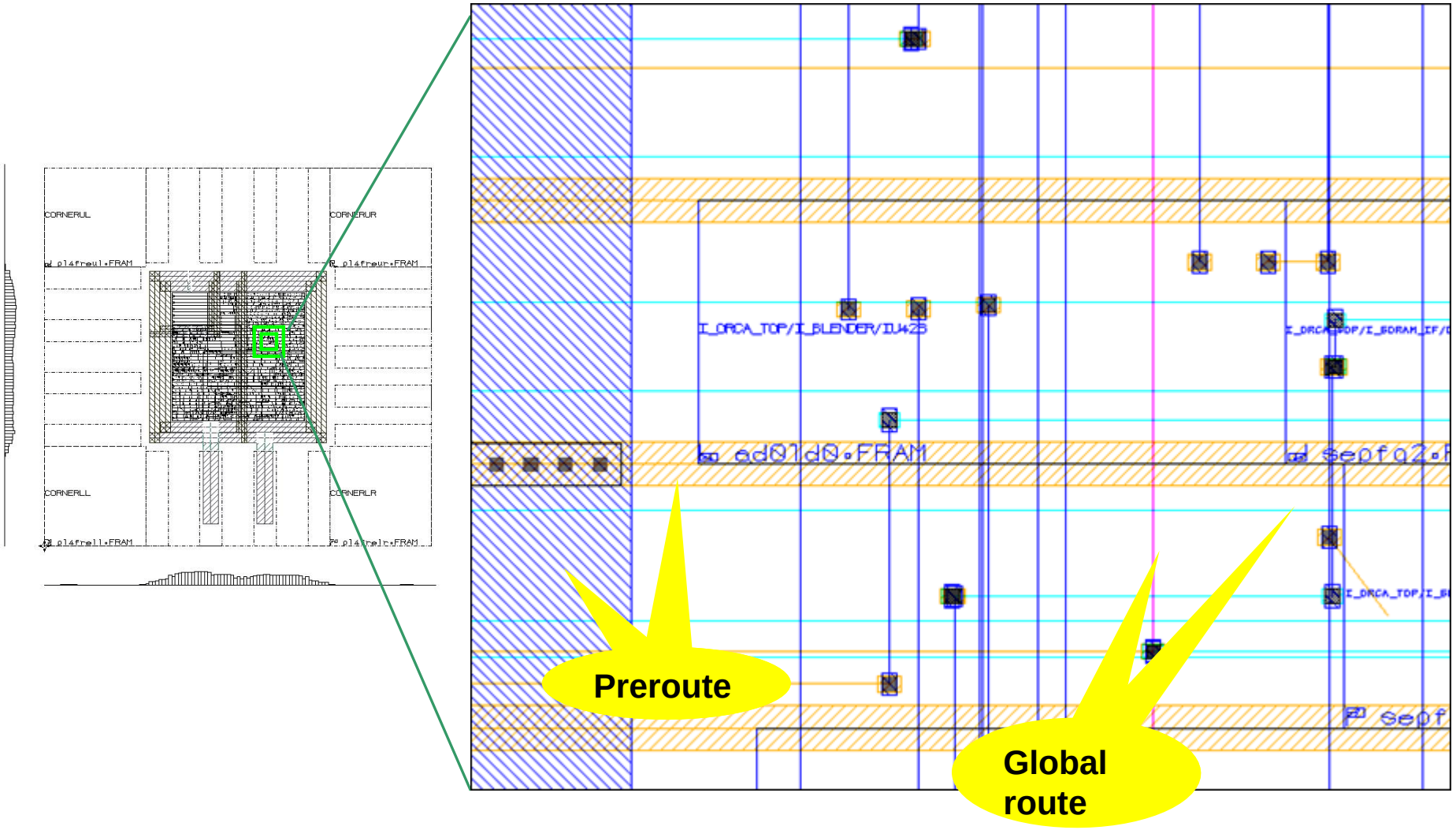
Track Assignment



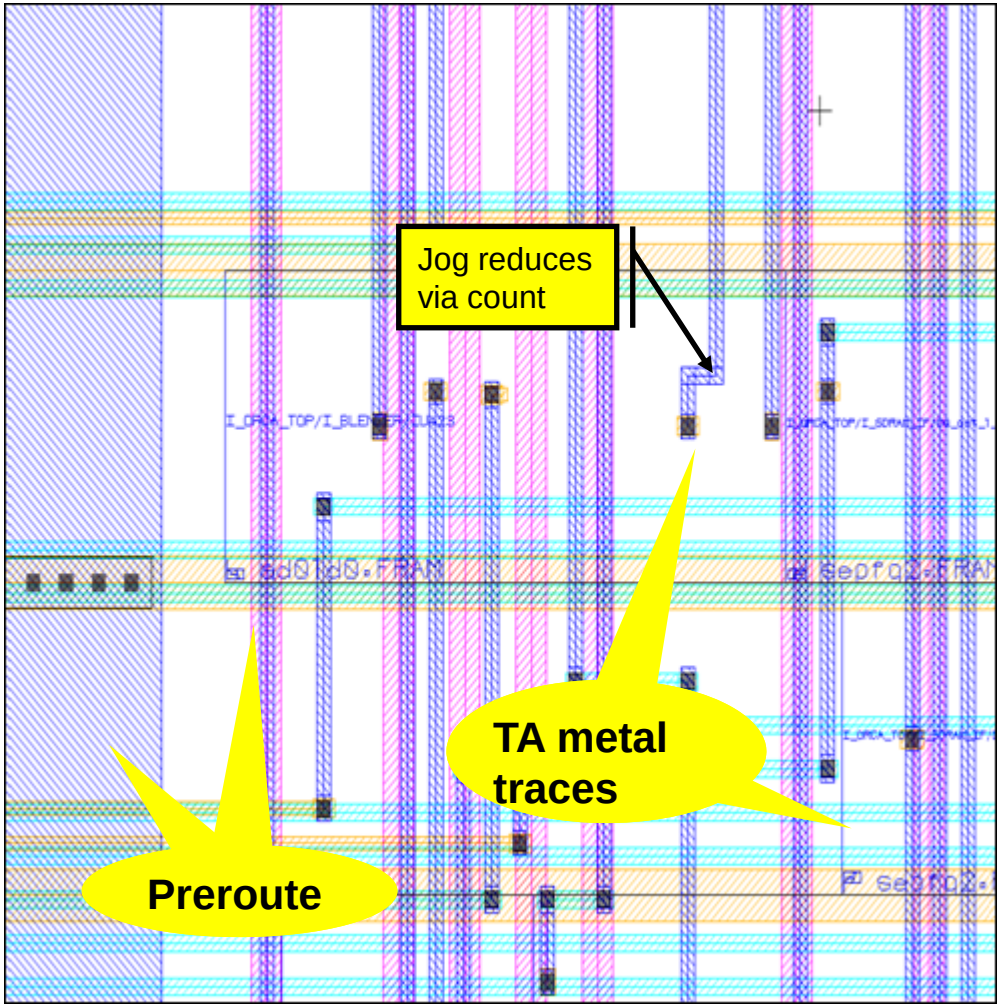


Detail Route

Routing - Global Route

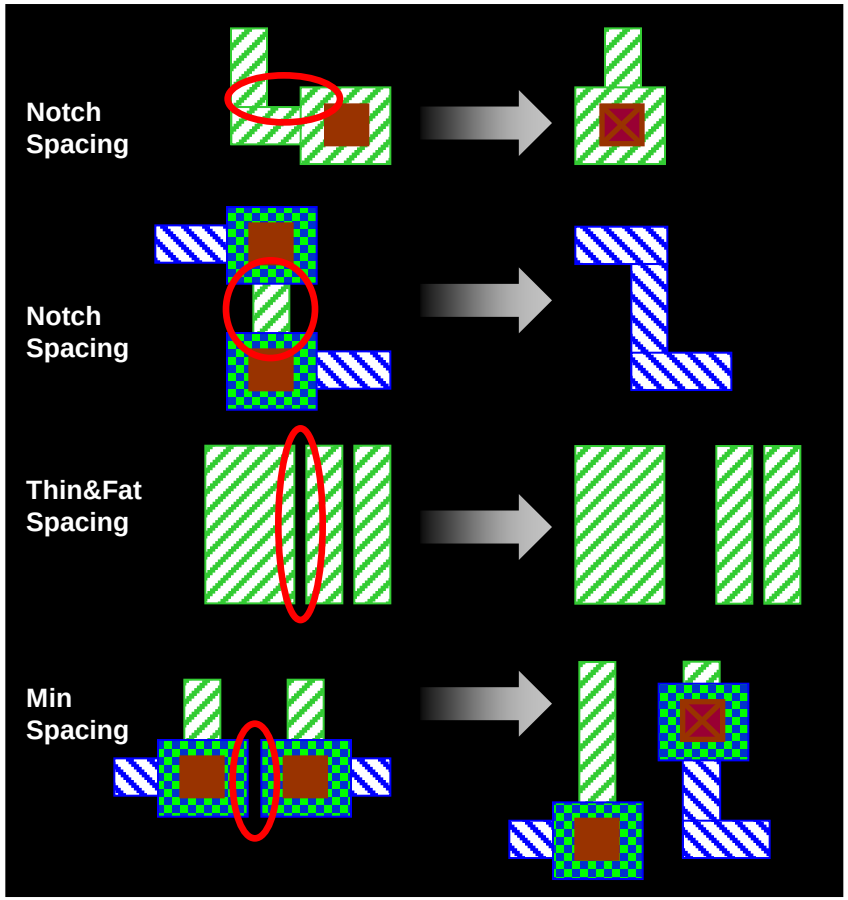


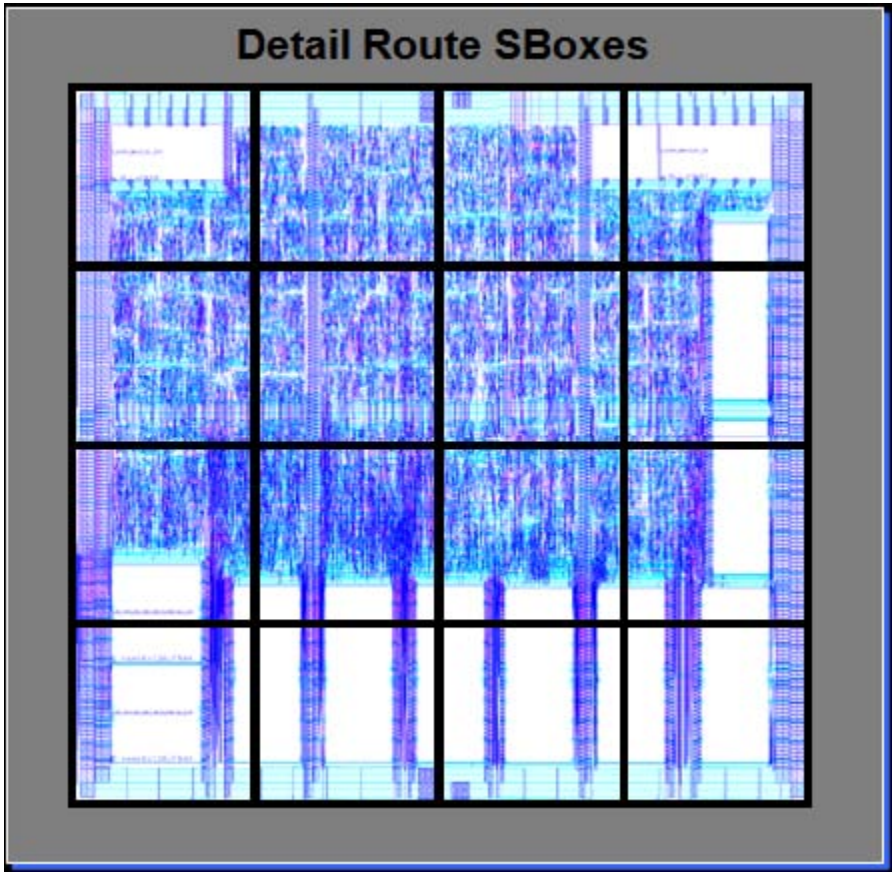
Routing - Track Assignment



Routing - Detail Route

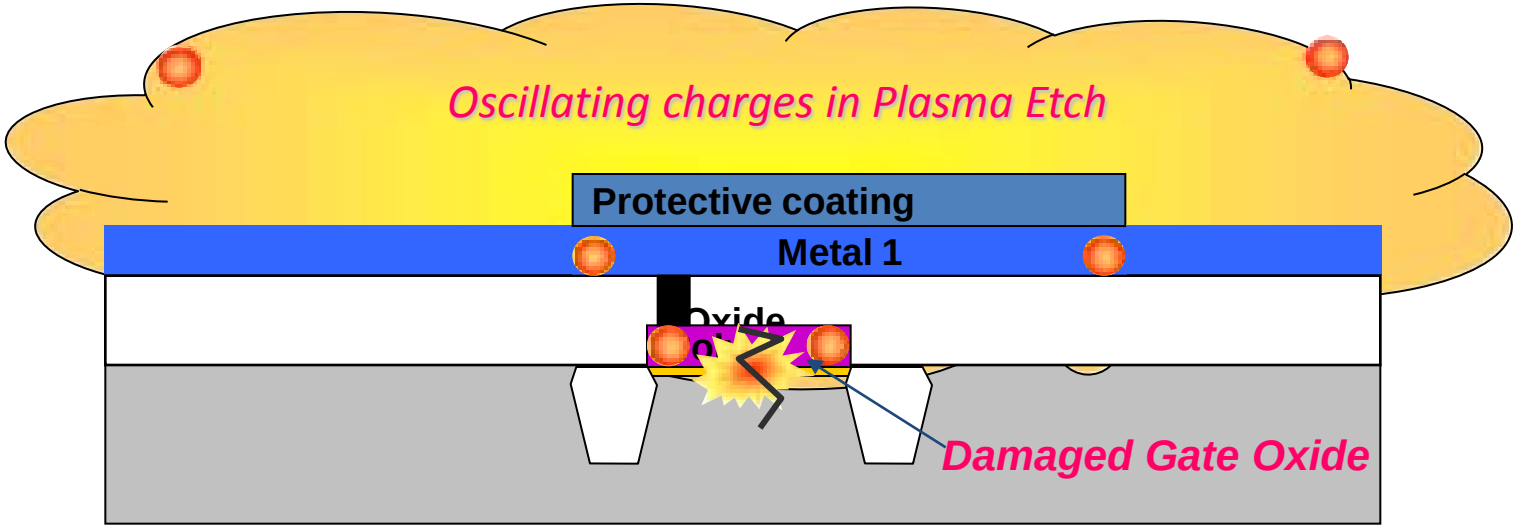
- Detail route attempts to clear DRC violations using a fixed size Sbox



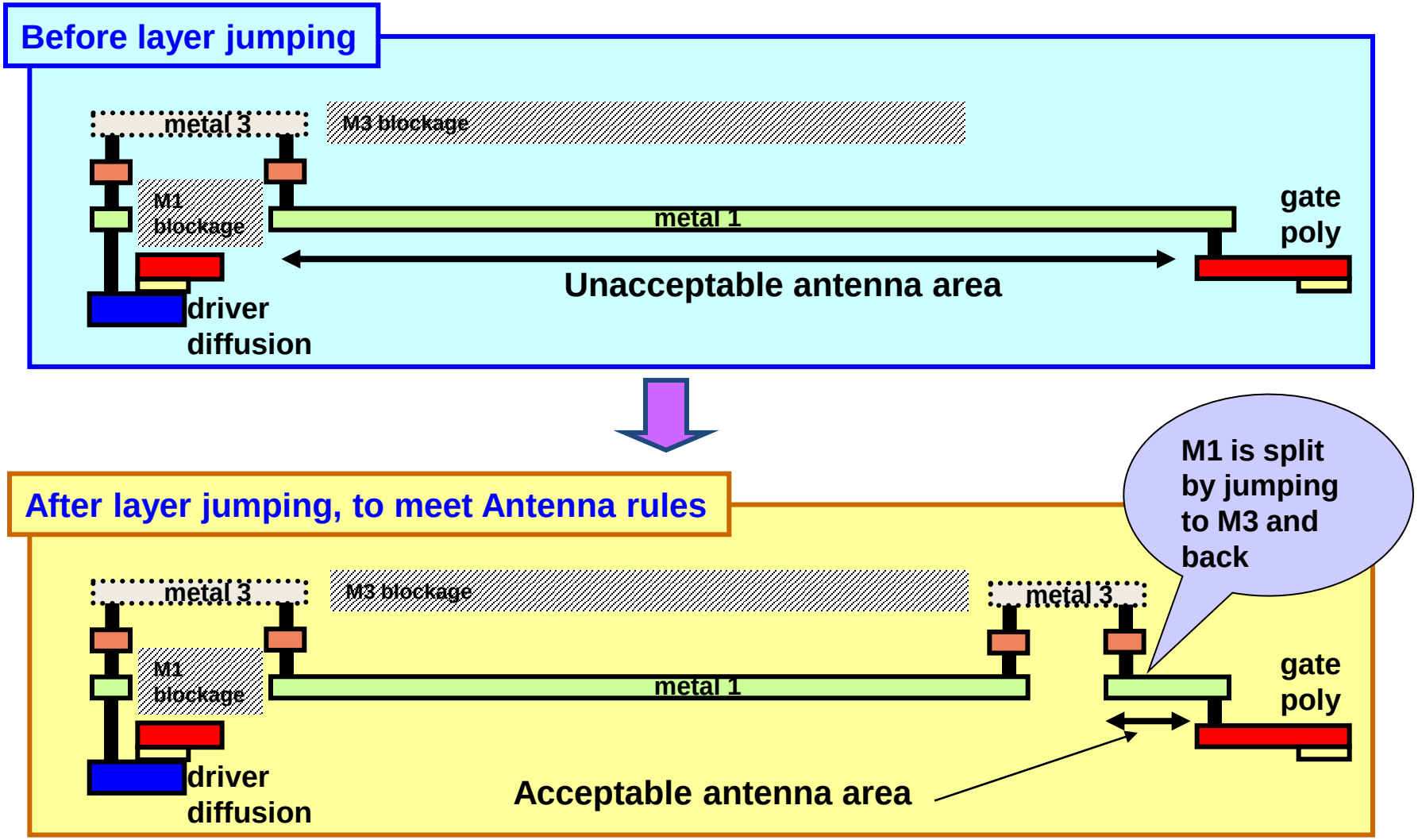


DFM Problem: Gate Oxide Integrity

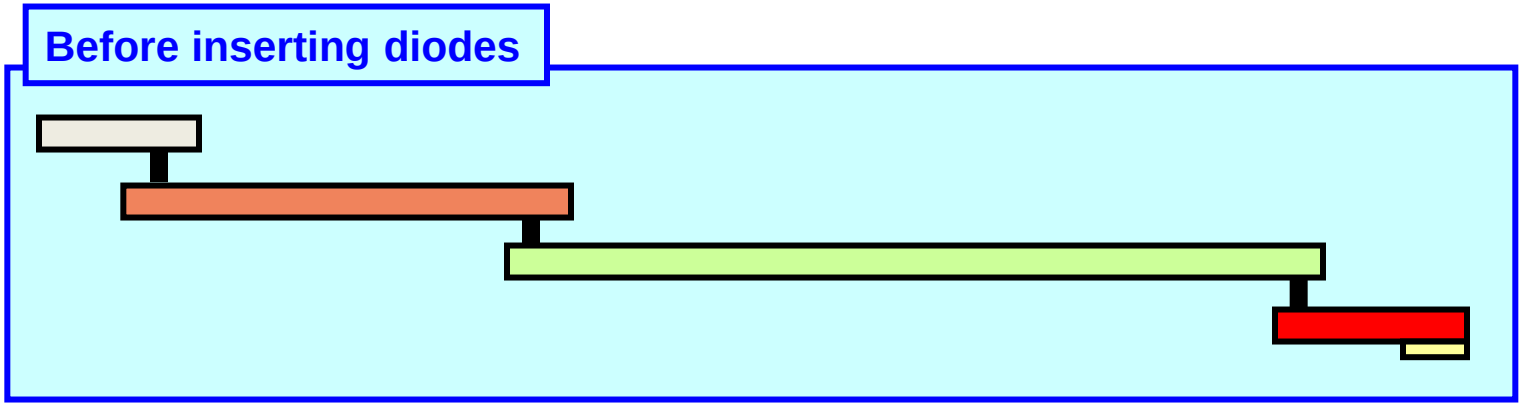
- Metal wires (antennae) placed in an EM field generate voltage gradients
- During the metal etch stage, strong EM fields are used to ionize the plasma etchant



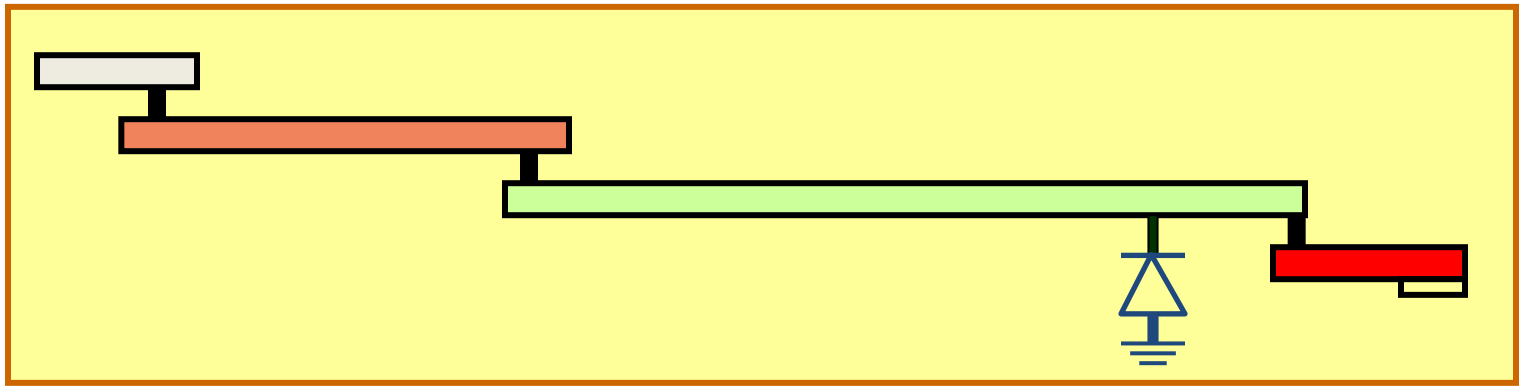
Solution 1: Splitting Metal or Layer Jumping



Solution 2: Inserting Diodes



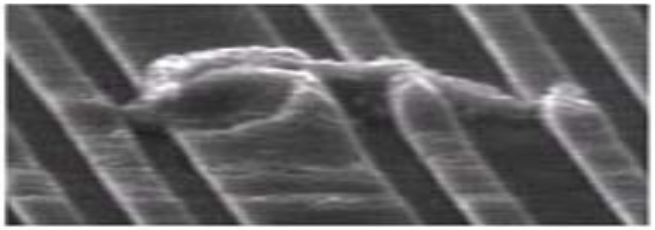
Diode Inhibits large voltage swings on metal tracks



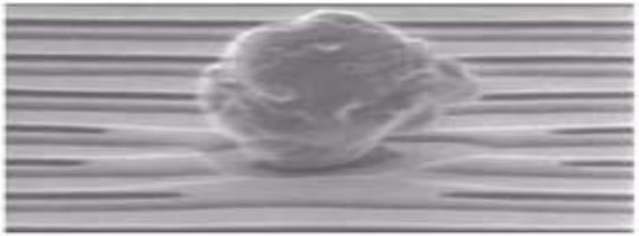
During etch phase, the diode clamps the voltage swings.

DFM Problem: Random Particle Defects

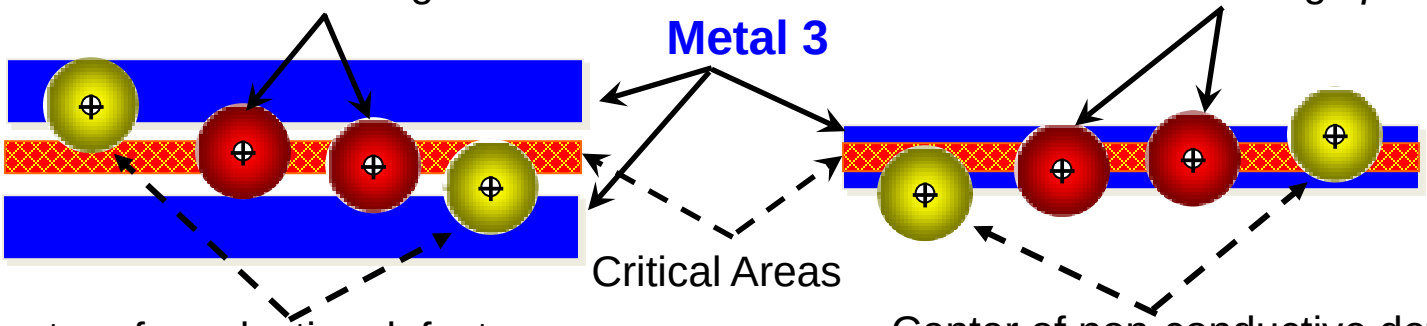
- Random particle defects during manufacturing may cause *shorts* or *opens* during the fabrication process



Center of conductive defects within critical area – causing *shorts*



Center of non-conductive defects within critical area – causing *opens*

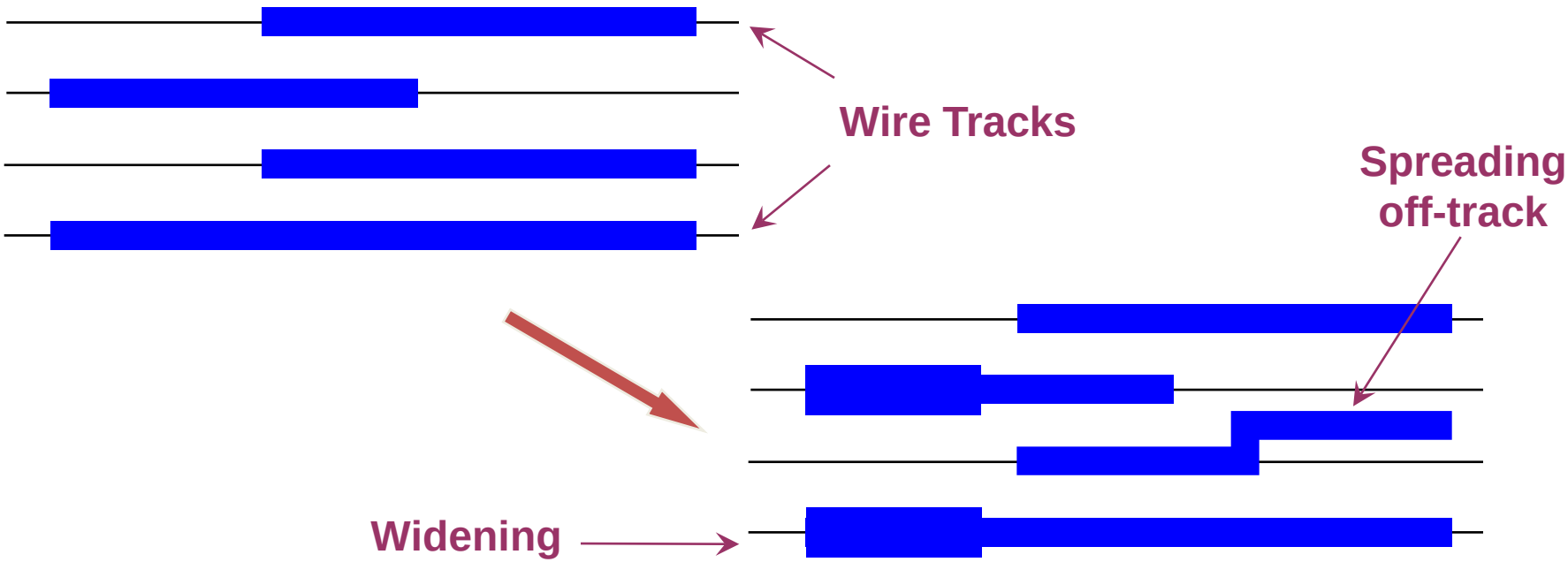


Center of conductive defects outside critical area – no shorts

Center of non-conductive defects outside critical area – no opens

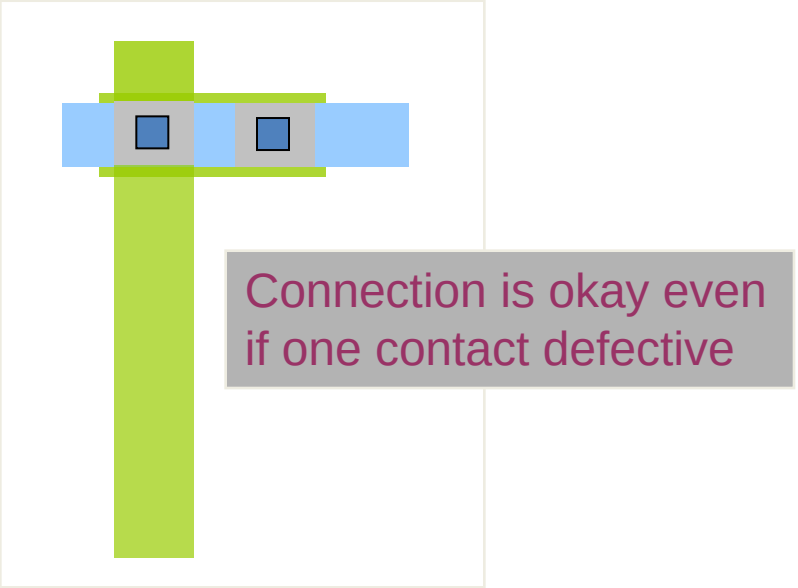
Solution: Wire Spreading + Widening

- Spread wires to reduce *short* critical area
- Widen wires to reduce *open* critical area



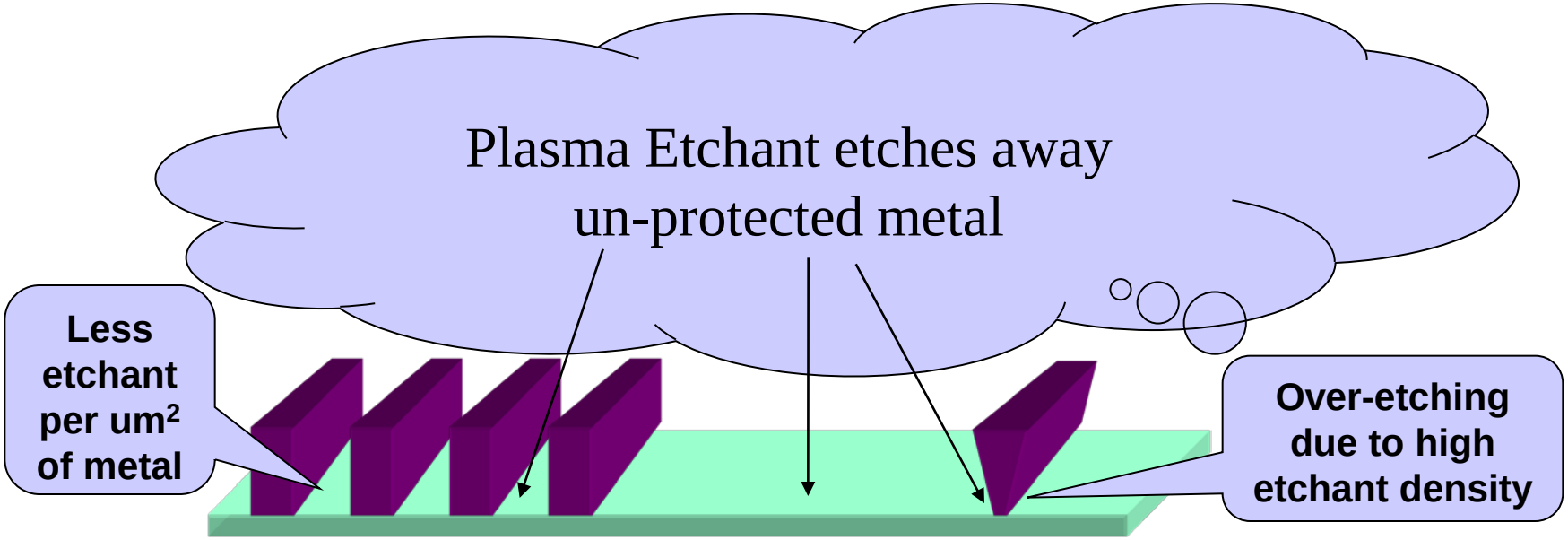
Voids in Vias during Manufacturing

- Voids in vias is a serious issue in manufacturing
- Two solutions are available:
  - Reduce via count:
  - Add backup vias: known as redundant vias



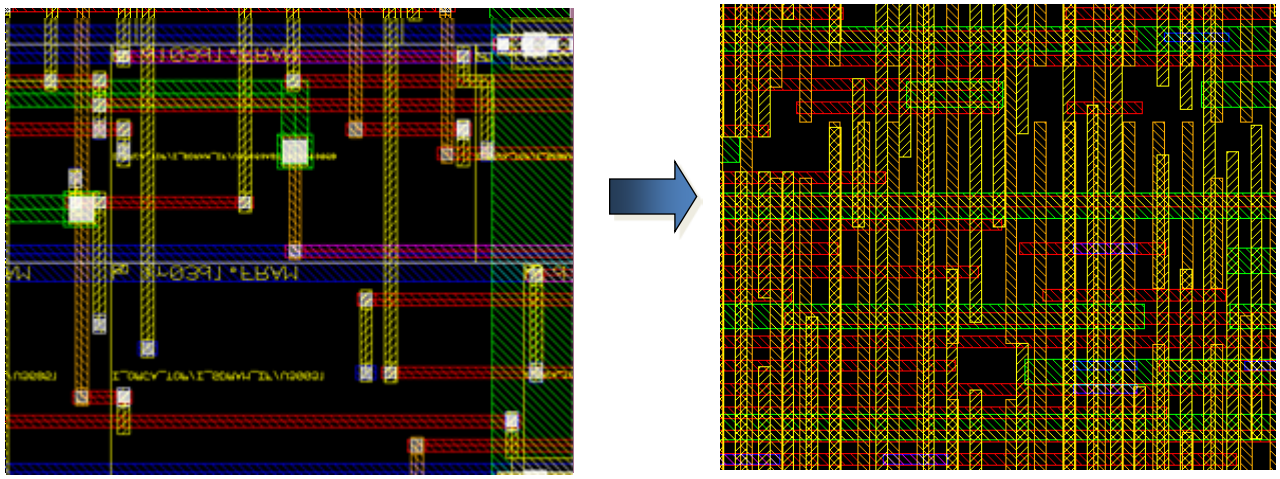
Problem: Metal Over-Etching

- A metal wire in low metal density region receives a higher ratio of etchant can get over-etched
- Minimum metal density rules are used to control this



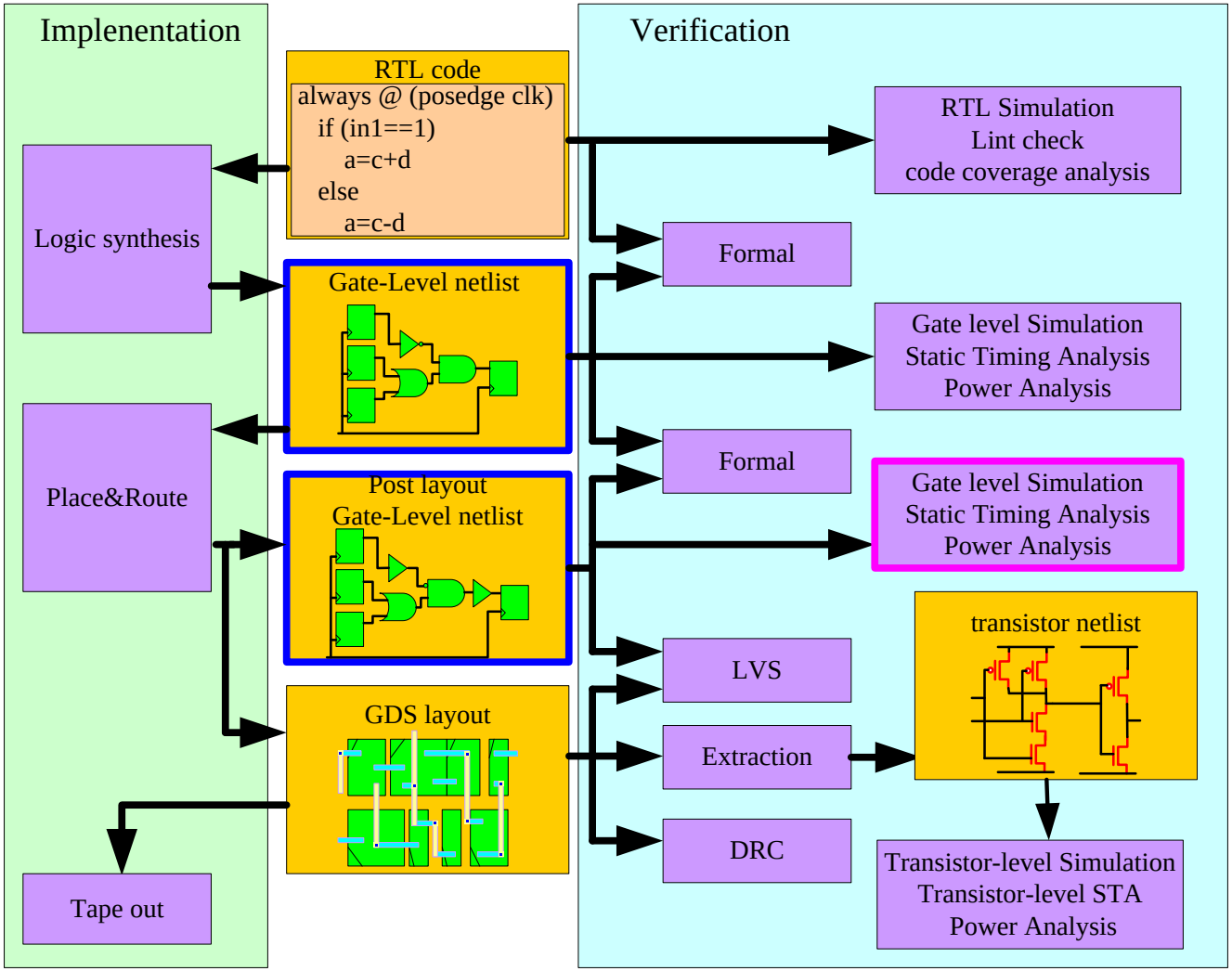
Solution: Metal Fill Insertion

- Fills empty tracks on all layers (default) with metal shapes to meet the minimum metal density rules



Cell-Based IC Design Flow

- Post-layout Verification

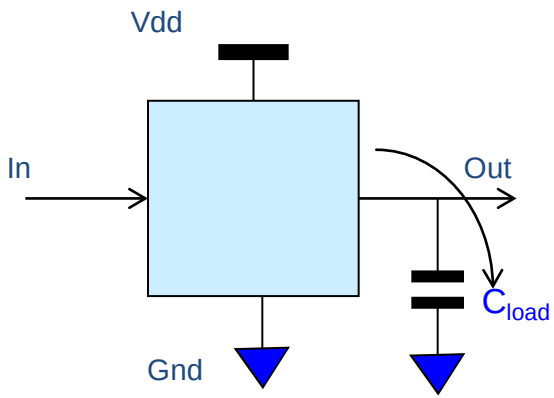


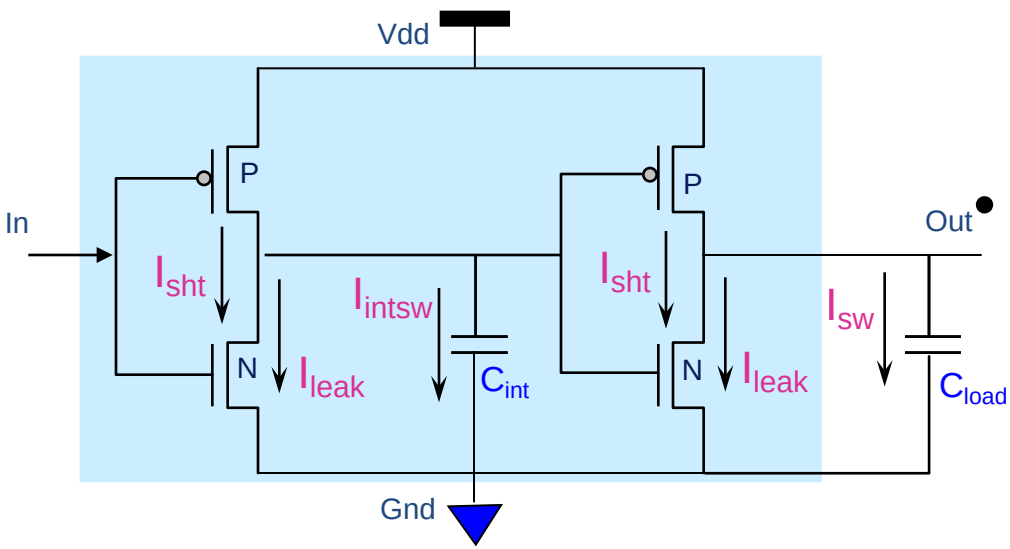
Post-layout Gate-level Verification

- Function Verification
  - Post-layout Gate-level Simulation
  - Formal Equivalence Checking
    - Pre-layout Gate-level HDL vs. Post-layout Gate-level HDL
- Timing Verification
  - Post-layout Gate-level Simulation
  - Post-layout Static Timing Analysis (STA)
- Power Verification
  - Power Analysis

Estimated Interconnect Parasitic
=> Real Interconnect Parasitic

Power Analysis



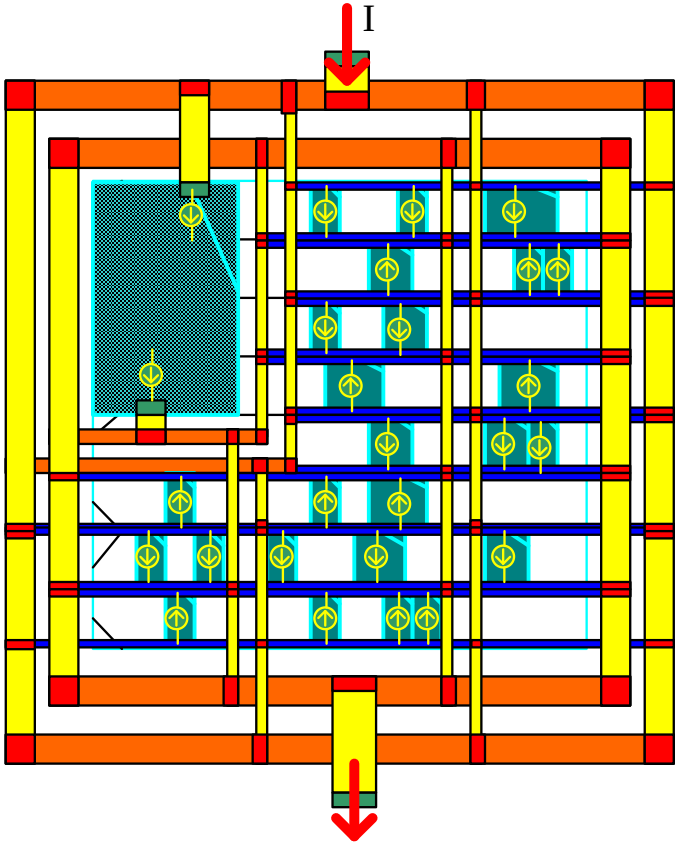


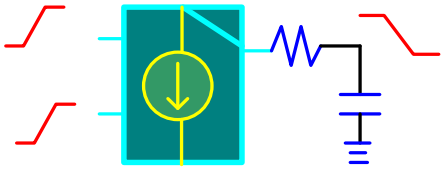
Total Power=

Dynamic Power + Static Power

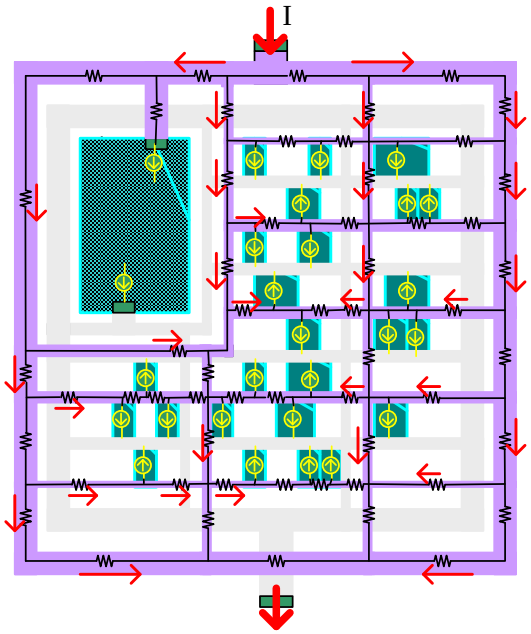
- Switching power (dynamic):
  - Charging output load
- Internal power (dynamic)
  - Short circuit
  - Charging internal load
- Leakage power (static)
  - Stable state

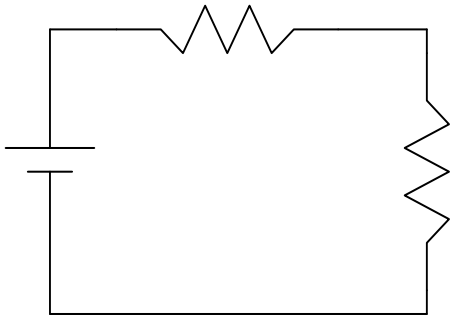
Power Analysis

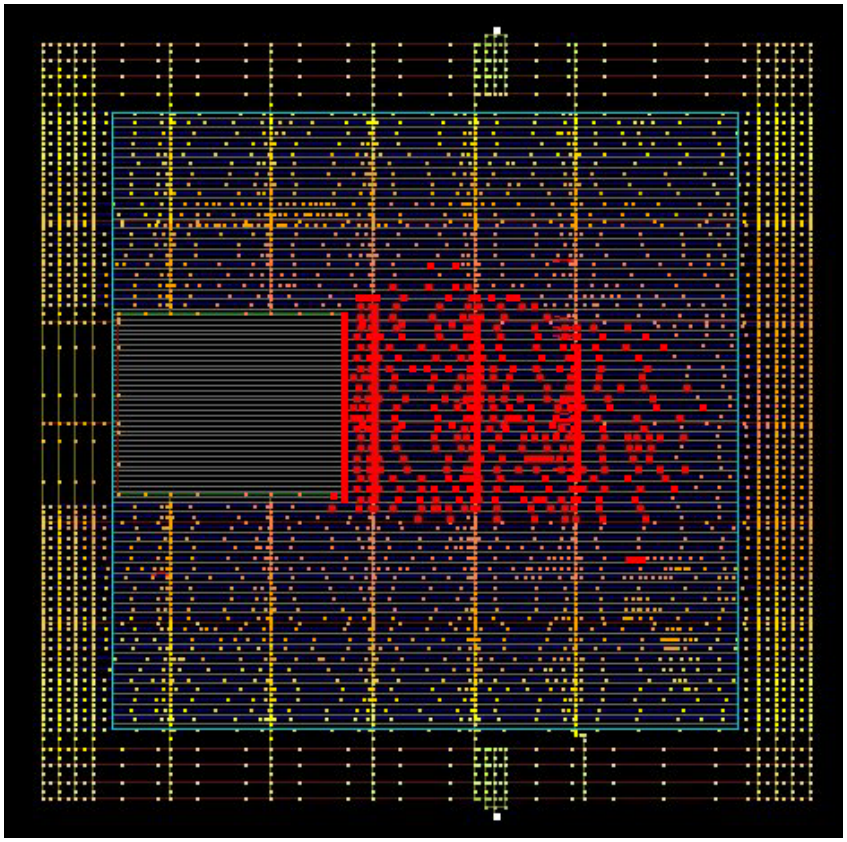




Rail Analysis

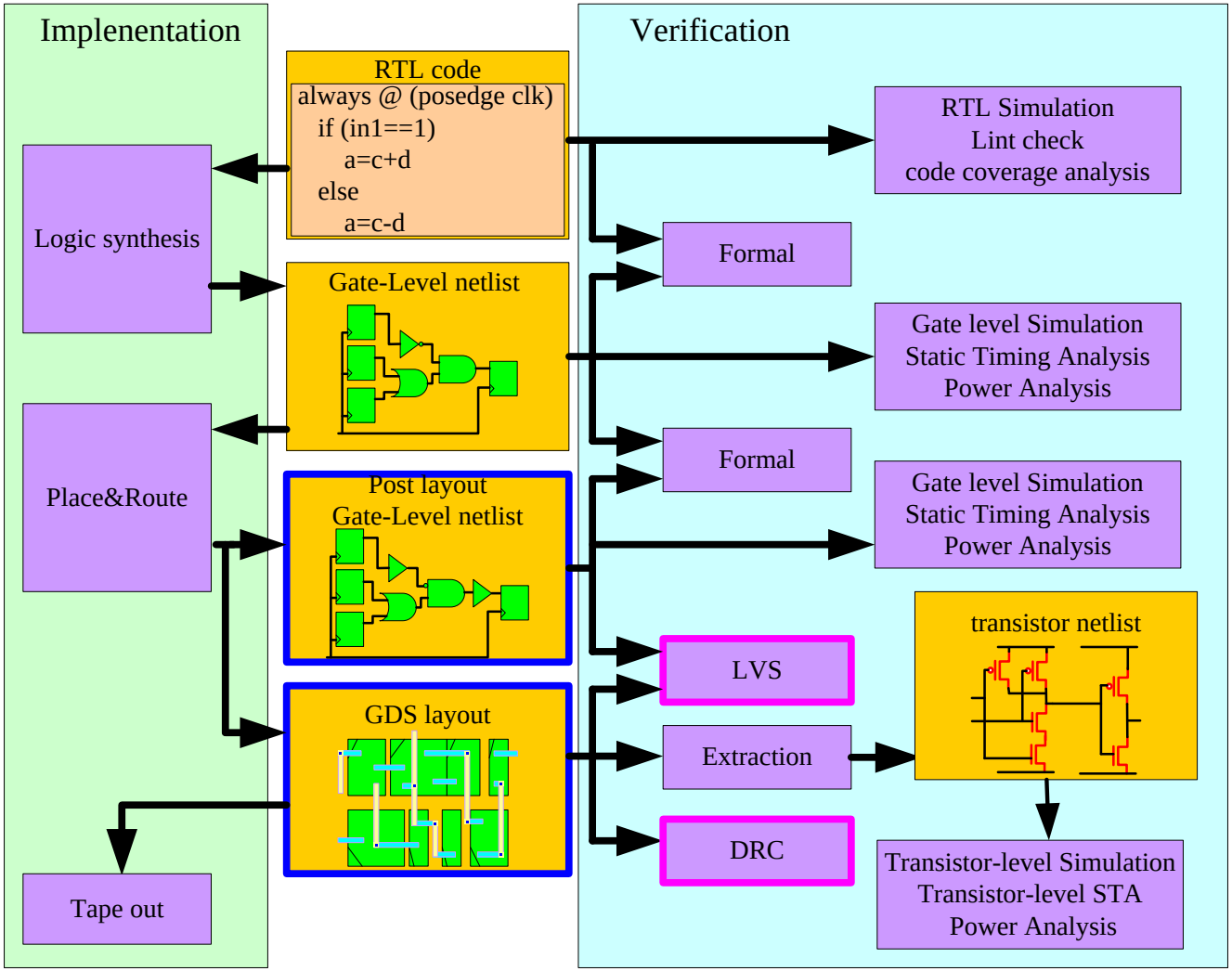






IR power graph

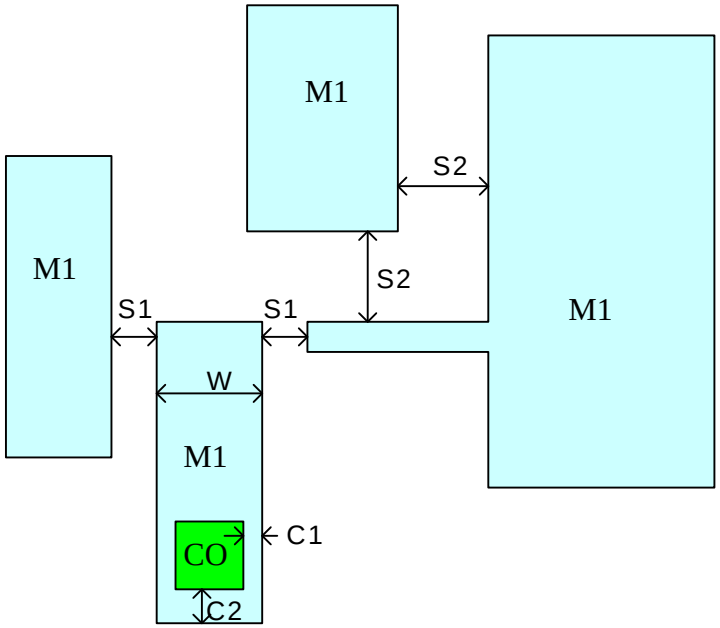
Cell-Based IC Design Flow - Physical Verification



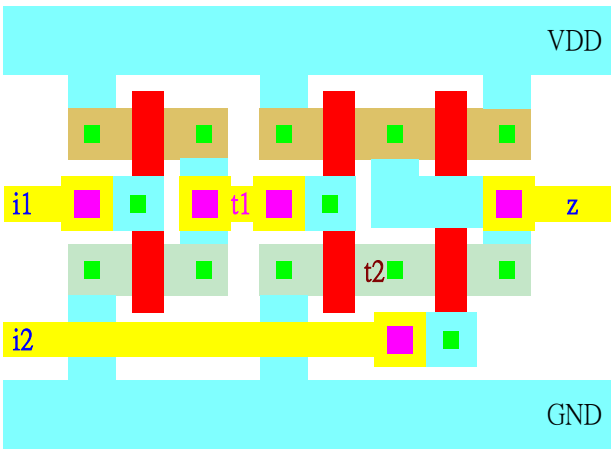
Physical Verification

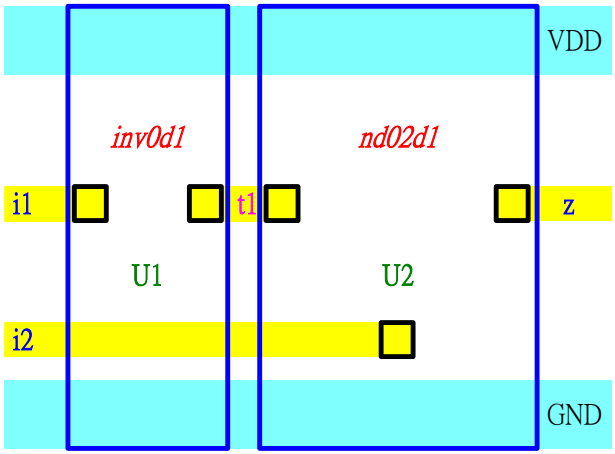
- For geometrical verification
  - Design Rule Check (DRC)
- For topological verification
  - Layout versus Schematic (LVS)

Design Rule Check



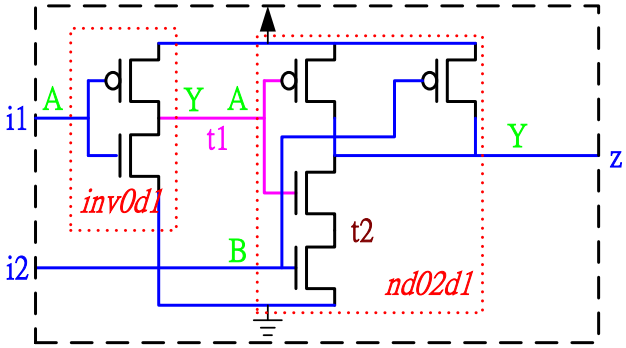
Black Box LVS

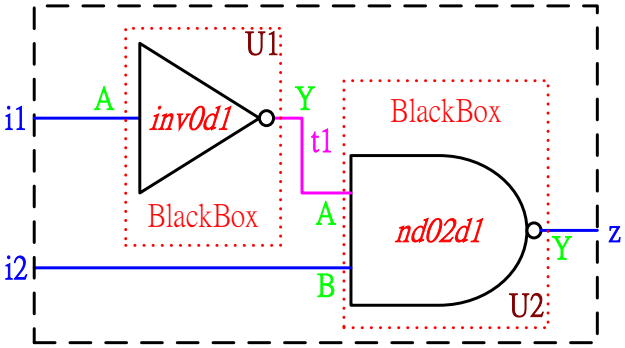




LVS

Black-box LVS





THE END

73