集成电路 EDA 设计精英挑战赛
INTEGRATED CIRCUIT EDA ELITE CHALLENGE

共圆芯梦想 2023(第五届)

集成电路EDA设计精英挑战赛总决赛

2023年12月22日-24日　南京江北新区

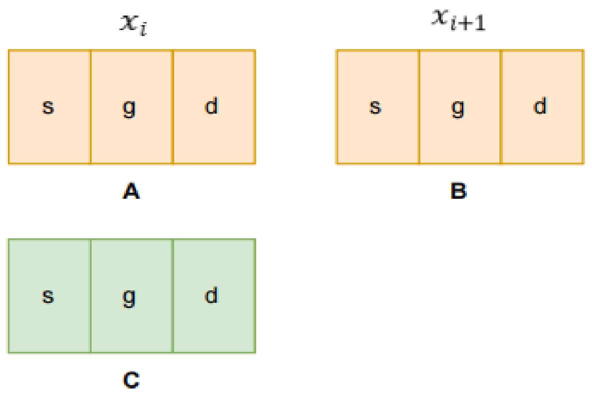
EDA230855

CONTENTS

Part I: Problem Formulation

集成电路EDA设计精英挑战赛 INTEGRATED CIRCUIT EDA ELITE CHALLENGE

- description



A. d==B. s
A. g==C. g

Figure.1 layout

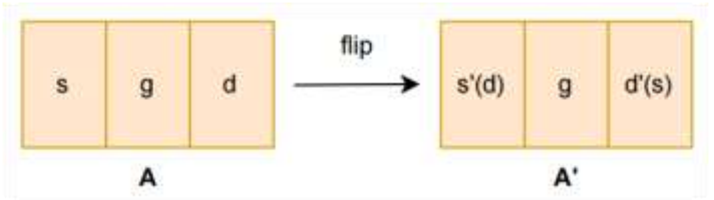


Figure.2 mos flip

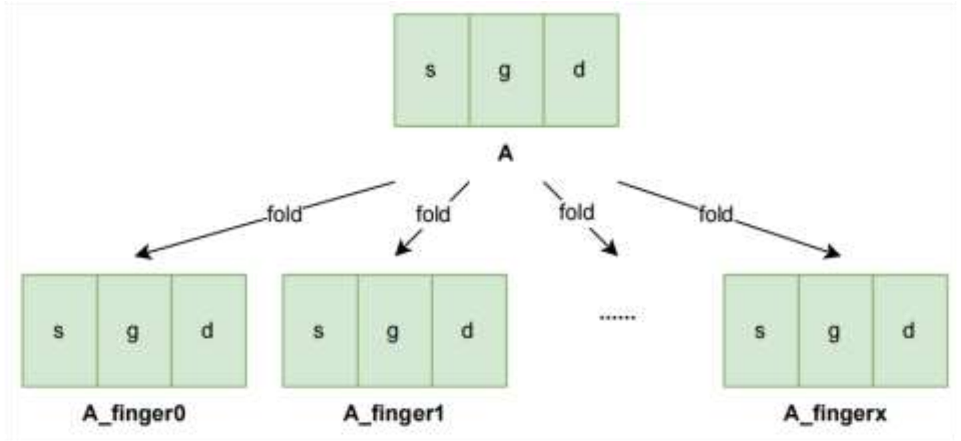


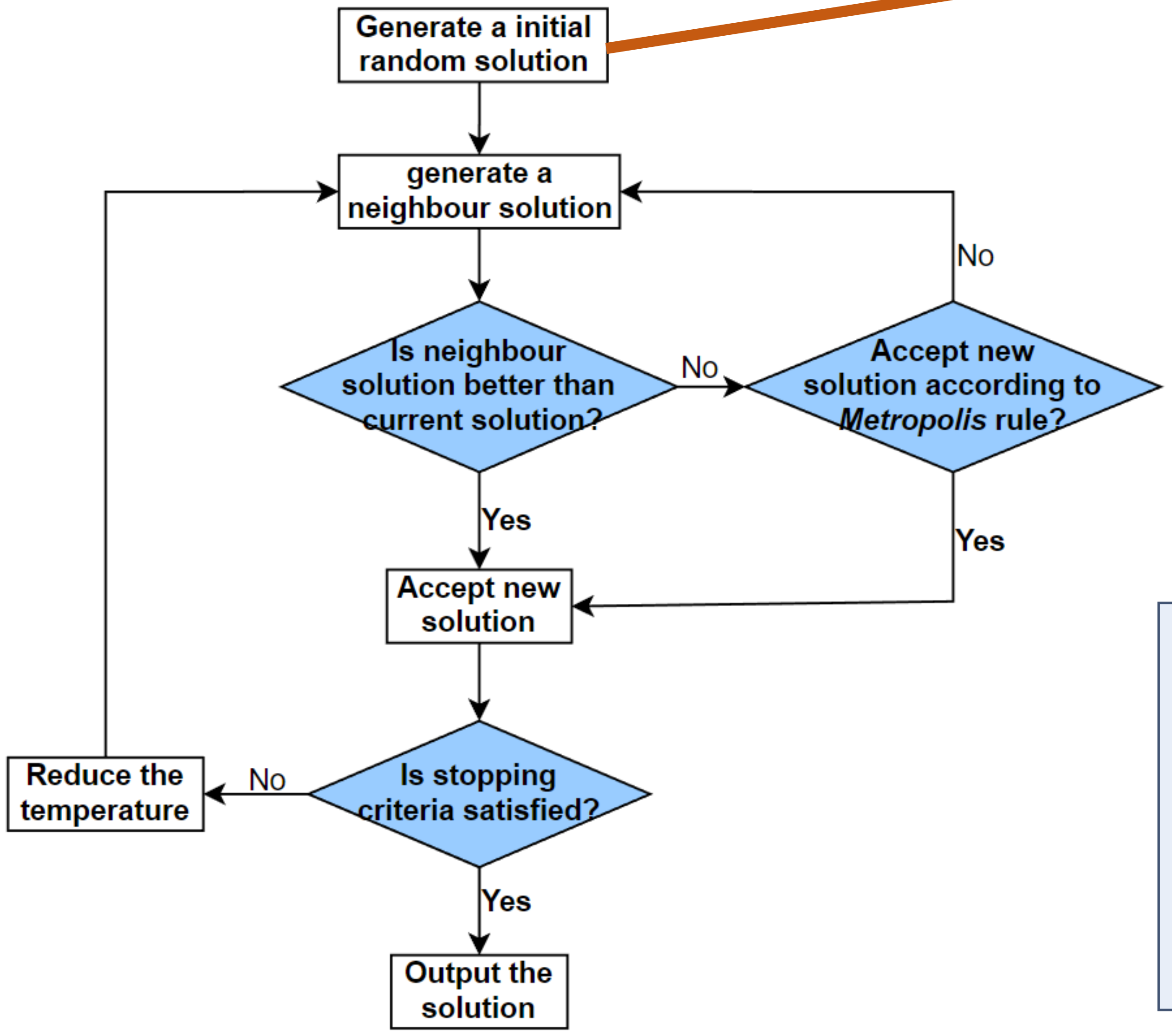
Figure.3 mos folded

For the layout of the MOS transistors within a given standard cell, legal layout results are generated based on optimization criteria. The main rules are:

(1) The source and drain of adjacent MOS transistors are the same.
(2) The gate of PMOS and NMOS transistors with the same x-coordinate is the same.
(3) Folding and flipping operations can be applied to MOS transistors.
(4) Optimization criteria mainly include layout width, wiring complexity, symmetry, pin density, DRC rules, and runtime.

Part II: Algorithms & Frameworks

- **Simulated annealing algorithm**



Generate a initial random solution
generate a neighbour solution
Is neighbour solution better than current solution?
No
Accept new solution according to *Metropolis* rule?
No
Yes
Yes
Accept new solution
Is stopping criteria satisfied?
No
Reduce the temperature
Yes
Output the solution

Generate a intial random solution:

Read the information of NMOS and PMOS from the file one by one and store them in the corresponding NMOS_list or PMOS_list.

First, place the first NMOS at position 0, then place a PMOS, and check if position 0 is legal for the PMOS (i.e., if the source, gate, and drain are in compliance, and the placement will not cause a source region notch).

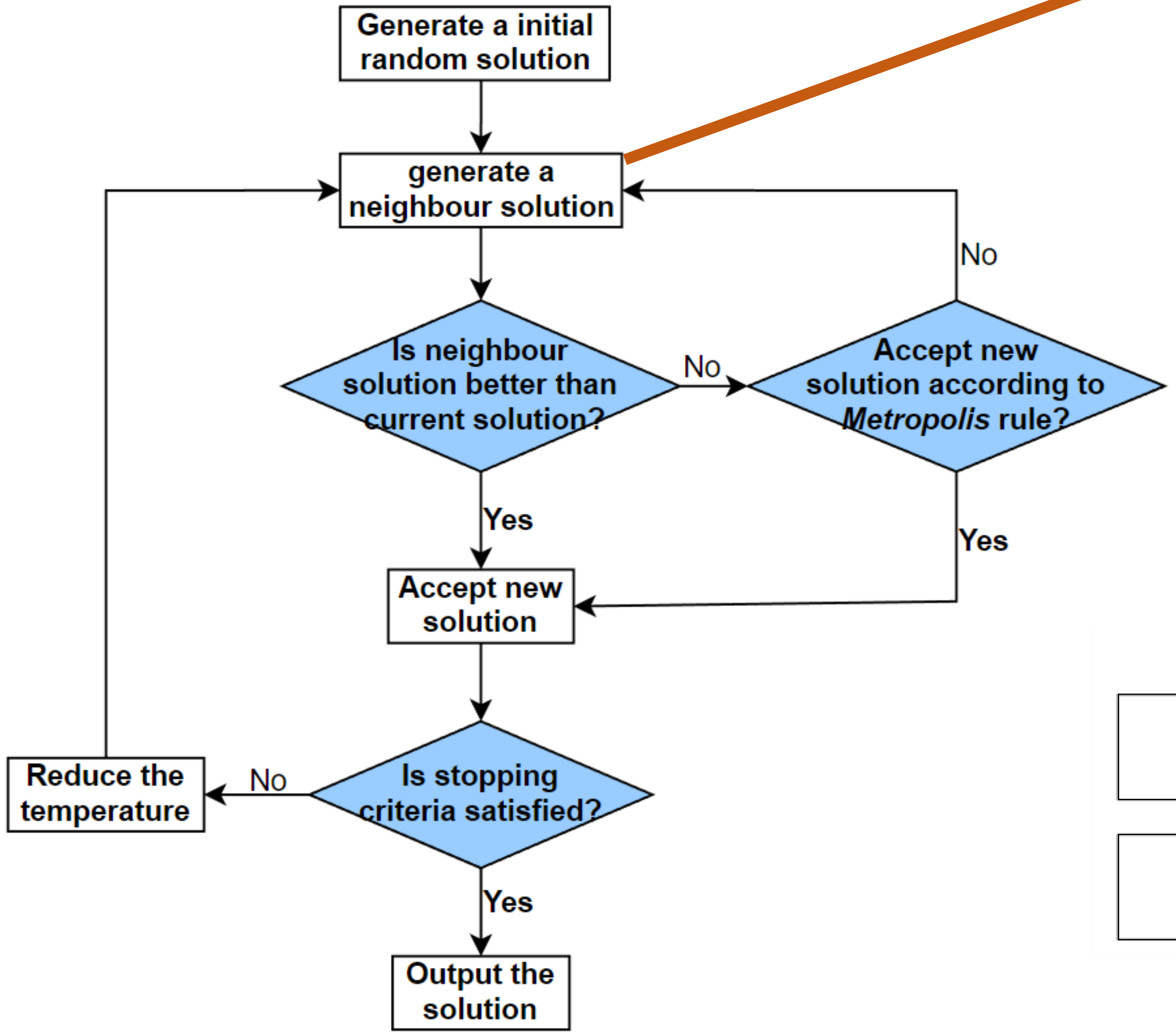
After the placement, continue to place the next NMOS... alternating the placement until all MOS transistors have been placed.

We **use a list to record the arrangement of MOS transistors from positions 0 to n+2** (where n is the maximum relative position of the MOS transistors in the current layout). If there is a MOS transistor at a position, the number of the transistor is filled into that position in the list; otherwise, None is filled in to represent an empty position.

Part II: Algorithms & Frameworks

集成电路EDA设计精英挑战赛 INTEGRATED CIRCUIT EDA ELITE CHALLENGE

- Simulated annealing algorithm


Generate a initial random solution
generate a neighbour solution
Is neighbour solution better than current solution?
No
Accept new solution according to *Metropolis* rule?
No
Yes
Yes
Accept new solution
Is stopping criteria satisfied?
No
Reduce the temperature
Yes
Output the solution

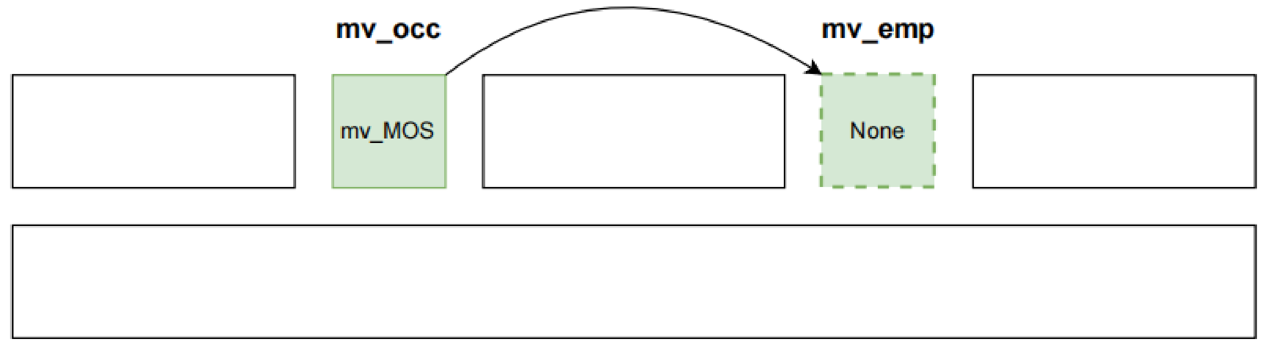
Generate a neighbour solution:

The update steps are as follows:

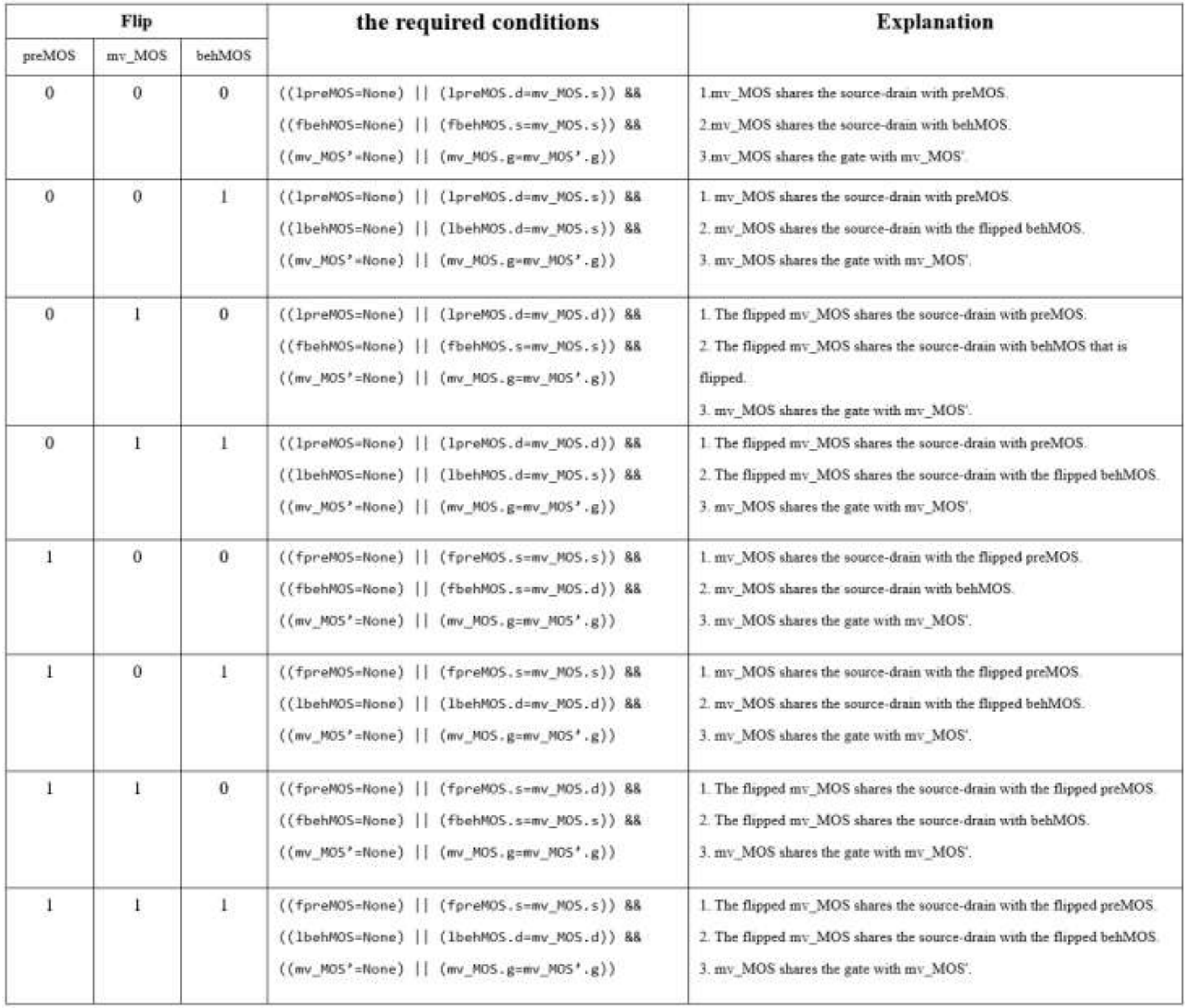
1. For an NMOS transistor, randomly select a transistor M and an empty position x.

2. Check if placing M at position x is legal (source-gate-drain matching, and no norch will occur). If it is legal, place the transistor M at position x; if it is not legal, attempt to **flip**.

3. For a PMOS transistor, execute steps 1 and 2 in the same manner.



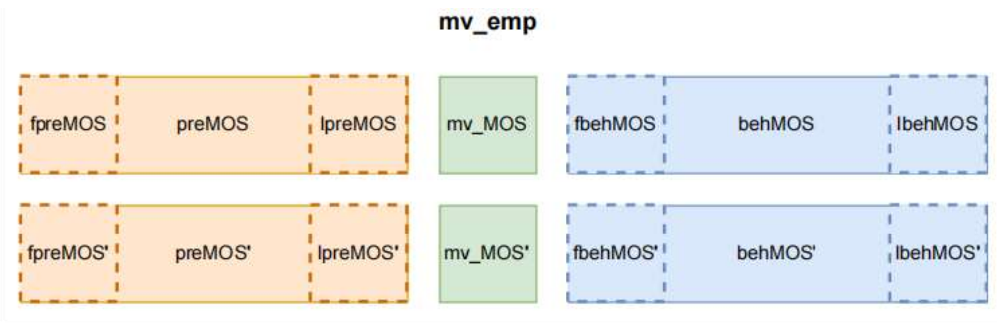
Part II: Algorithms & Frameworks



Flip			the required conditions	Explanation
preMOS	mv_MOS	behMOS		
0	0	0	((lpreMOS=None) || (lpreMOS.d=mv_MOS.s)) && ((fbehMOS=None) || (fbehMOS.s=mv_MOS.s)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1.mv_MOS shares the source-drain with preMOS. 2.mv_MOS shares the source-drain with behMOS. 3.mv_MOS shares the gate with mv_MOS'.
0	0	1	((lpreMOS=None) || (lpreMOS.d=mv_MOS.s)) && ((lbehMOS=None) || (lbehMOS.d=mv_MOS.s)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. mv_MOS shares the source-drain with preMOS. 2. mv_MOS shares the source-drain with the flipped behMOS. 3. mv_MOS shares the gate with mv_MOS'.
0	1	0	((lpreMOS=None) || (lpreMOS.d=mv_MOS.d)) && ((fbehMOS=None) || (fbehMOS.s=mv_MOS.s)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. The flipped mv_MOS shares the source-drain with preMOS. 2. The flipped mv_MOS shares the source-drain with behMOS that is flipped. 3. mv_MOS shares the gate with mv_MOS'.
0	1	1	((lpreMOS=None) || (lpreMOS.d=mv_MOS.d)) && ((lbehMOS=None) || (lbehMOS.d=mv_MOS.s)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. The flipped mv_MOS shares the source-drain with preMOS. 2. The flipped mv_MOS shares the source-drain with the flipped behMOS. 3. mv_MOS shares the gate with mv_MOS'.
1	0	0	((fpreMOS=None) || (fpreMOS.s=mv_MOS.s)) && ((fbehMOS=None) || (fbehMOS.s=mv_MOS.d)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. mv_MOS shares the source-drain with the flipped preMOS. 2. mv_MOS shares the source-drain with behMOS. 3. mv_MOS shares the gate with mv_MOS'.
1	0	1	((fpreMOS=None) || (fpreMOS.s=mv_MOS.s)) && ((lbehMOS=None) || (lbehMOS.d=mv_MOS.d)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. mv_MOS shares the source-drain with the flipped preMOS. 2. mv_MOS shares the source-drain with the flipped behMOS. 3. mv_MOS shares the gate with mv_MOS'.
1	1	0	((fpreMOS=None) || (fpreMOS.s=mv_MOS.d)) && ((fbehMOS=None) || (fbehMOS.s=mv_MOS.s)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. The flipped mv_MOS shares the source-drain with the flipped preMOS. 2. The flipped mv_MOS shares the source-drain with behMOS. 3. mv_MOS shares the gate with mv_MOS'.
1	1	1	((fpreMOS=None) || (fpreMOS.s=mv_MOS.s)) && ((lbehMOS=None) || (lbehMOS.d=mv_MOS.d)) && ((mv_MOS'=None) || (mv_MOS.g=mv_MOS'.g))	1. The flipped mv_MOS shares the source-drain with the flipped preMOS. 2. The flipped mv_MOS shares the source-drain with the flipped behMOS. 3. mv_MOS shares the gate with mv_MOS'.

- **Flip**

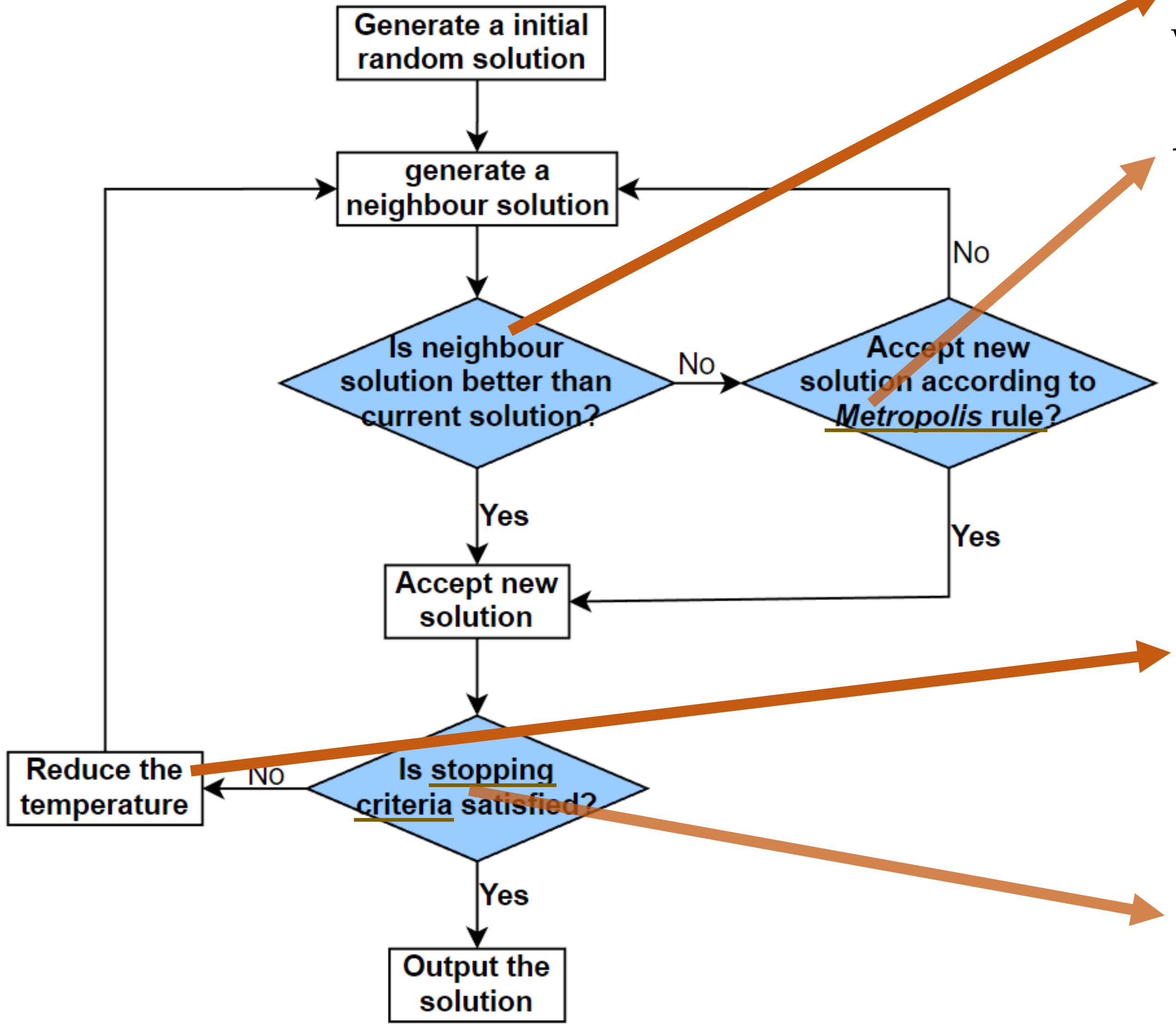
Calculate the set of MOS transistors preMOS that have already produced source-drain sharing before the empty position mv_emp, the set of MOS transistors behMOS that have already produced source-drain sharing after the empty position mv_emp, as well as another set of MOS transistors preMOS' at the same position as preMOS, and another set of MOS transistors behMOS' at the same position as behMOS. Since MOS transistors can be flipped, it is possible to maximize source-drain sharing by flipping preMOS, mv_MOS, and behMOS. Therefore, the situations in which these three are flipped are shown in the left figure.



Part II: Algorithms & Frameworks

集成电路EDA设计精英挑战赛 INTEGRATED CIRCUIT EDA ELITE CHALLENGE

- **Simulated annealing algorithm**



Generate a initial random solution
generate a neighbour solution
Is neighbour solution better than current solution?
No
Accept new solution according to Metropolis rule?
No
Yes
Yes
Accept new solution
Is stopping criteria satisfied?
No
Reduce the temperature
Yes
Output the solution

The **evaluation function** calculation method is consistent with the one provided in the evaluator.py file.

***Metropolis* rule:**

If the new layout is better than the current layout, it is definitely updated to the new layout.

otherwise, the new layout is accepted with a probability of $e^{-\frac{c(s_1)-c(s_2)}{T}}$ where T is the current temperature, $c(s_2)$is the value of the evaluation function for the new layout, and $c(s_1)$ is the value of the evaluation function for the current layout.

Temperature drop: exponential decay

$$T_0 = 20000 * N$$
$$T = T * 0.88$$

The number of **iterations** L at each temperature is $2000 * N$.

Stopping criteria:$T < 0.1$

Part III: Experimental results

集成电路EDA设计精英挑战赛
INTEGRATED CIRCUIT EDA ELITE CHALLENGE

- **Layout result**

Cell	width	bbox	Pin_access	symmetric	drc:10	runtime
AN2D2	4	3.5	0.0625	10	10	0.s
AN3D2	5	4.5	0.047140452	10	10	0.s
AN4D2	7	7.5	0.092788436	8	10	5.656s
AND2V4	7	10.5	0.088388348	10	10	93.641s
AND3V4	9	17	0.073950997	10	10	120.953s
AND4V4	8	15.5	0.114017543	10	10	255.281s
AO22V4	8	20	0.096953597	10	10	160.297s
AO32V4	11	27	0.055987901	10	10	344.234s
AO33V4	15	68	0.03029875	6	10	409.266s
DFCNQD2	45	466.5	0.121742294	-29	10	1091.828s
DFCSND2	53	677.5	0.134989989	-30	10	1728.422s
DFCSNQD2	44	497.5	0.073849077	-30	10	977.609s
DFKCND2	37	363	0.108027561	-22	10	644.515s

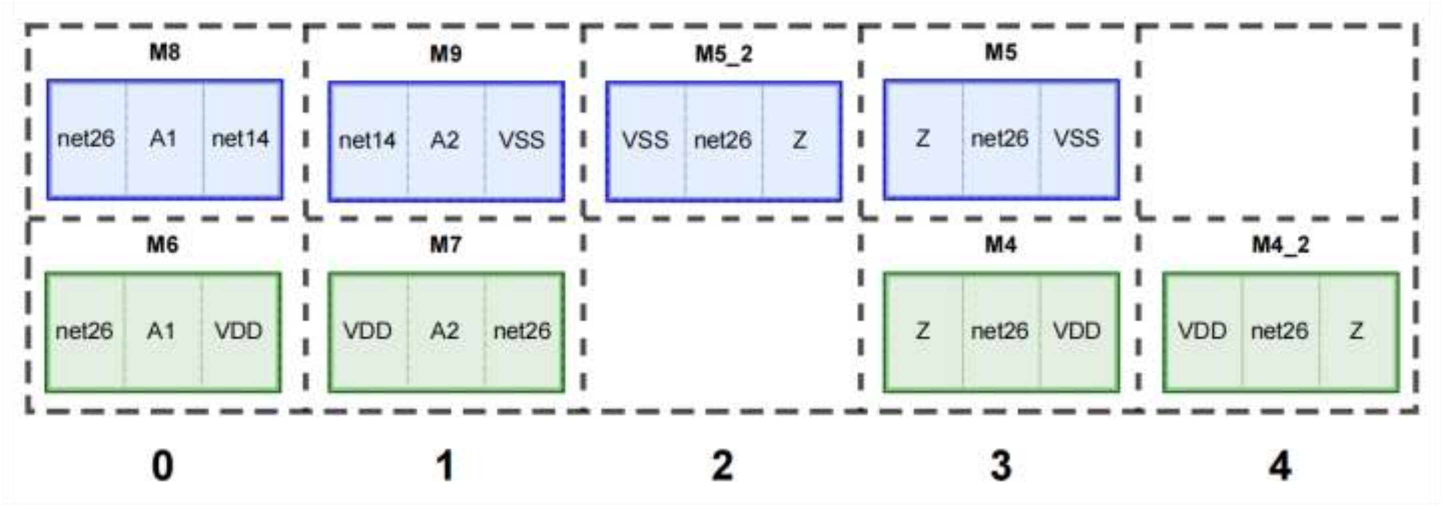
When the number of MOS tubes is less than 20, it can be seen that the indicators of the layout result are relatively good, and the program runs faster.

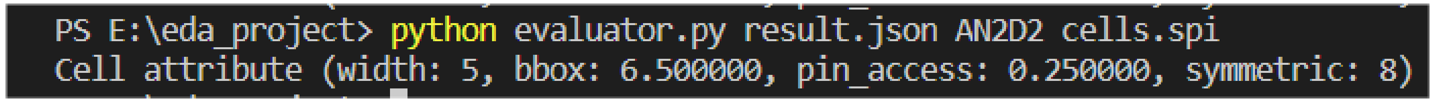
When the number of MOS tubes is large, the optimization effect of each index is not obvious, indicating that there are still many MOS tubes that do not share the active area, and the program runs for a relatively long time.

From the table, it can be seen that the layout results of all standard cells comply with the DRC rules. This is because considering the possibility of multiple occurrences of DRC, the algorithm design has avoided the generation of active area notches in the initial layout and optimization process.

Part III: Experimental results

- **An example**





```
PS E:\eda_project> python evaluator.py result.json AN2D2 cells.spi
Cell attribute (width: 5, bbox: 6.500000, pin_access: 0.250000, symmetric: 8)
```

This is the layout result for the standard cell AN2D2, and it can be seen that most of the MOS transistors share the active regions.

Part IV: Conclusion & Discussion

• **Why OCaml**

OCaml has features similar to C such as static type checking, advanced type system, and efficiency benefits, as well as Python-like features such as functional programming support, high-level abstraction, and development efficiency.

OCaml strikes a balance between maintaining the performance and control of the underlying language, C, while providing high-level abstraction and ease of development similar to that of Python.

Part IV: Conclusion & Discussion

OCaml

The main reasons

1. Powerful static type system (reliable)
2. Highly expressive(Easy to understand)
3. Lightweight concurrency model (Efficiency)
4. Pattern matching and type inference (Safety and Readability)

Part IV: Conclusion & Discussion

The main reasons

1. Powerful static type system

The static type system provides more accurate type information for integrated circuit design, making error location more accurate and faster.

2. Highly expressive

Through OCaml's **polymorphic type and module system**, it is easy to create abstract data structures and realize complex operations such as combinatorial logic and timing logic in circuit design.

Part IV: Conclusion & Discussion

The main reasons

3. Lightweight concurrency model

Multi-threading support is provided to execute multiple tasks in parallel. The use of multithreading allows different parts or tasks in a circuit design to be processed in parallel, improving computational performance and responsiveness.

4. Pattern matching /type inference

We can use **pattern matching** to describe and handle multiple states of a circuit, such as inputs, outputs, faults, and so on, and perform corresponding actions based on the matching situation.

Part IV: Conclusion & Discussion

- **Future improvements**

01
Concurrency of OCaml

OCaml's concurrency features allow for concurrent execution of multiple tasks, enabling parallel processing and improved performance.

02
Parameters of Algorithm

For the parameters of the simulated annealing algorithm, we will constantly adjust them to find the parameters that perform better in the algorithm.

03
Attenuation of Function

- Exponential Decay

$$T_k = T_0 \cdot \alpha^k$$

- Linear Decay

$$T_k = T_0 - \beta \cdot k$$

- Loagrithmic Decay

$$T_k = \frac{T_0}{1 + \alpha \cdot \log(1 + k)}$$

集成电路 EDA 设计精英挑战赛
INTEGRATED CIRCUIT EDA ELITE CHALLENGE

THANKS!