2023 集成电路 EDA 设计精英挑战赛

作品设计报告

赛题八：标准单元电路的版图自动生成

参赛队 ID：eda230855

二〇二三年十一月

一、 赛题实验结果

1. 结果展示

a. 标准单元 AN2D2

布局结果：

<table>
<tr>
<td>"placement": {
"M6": {
"x": "0",
"y": "1",
"source": "VDD",
"gate": "A1",
"drain": "net26",
"width": "170",
},</td>
<td>"M7":{
"x": "1",
"y": "1",
"source":"net26",
"gate": "A2",
"drain":"VDD",
"width": "170",
},</td>
<td>"M4": {
"x": "2",
"y": "1",
"source": "VDD",
"gate":"net26",
"drain": "Z",
"width": "170" ,
},</td>
<td>"M4_2": {
"x": "3",
"y": "1",
"source": "Z",
"gate": "net26",
"drain": "VDD",
"width": "170",
}</td>
</tr>
<tr>
<td>"M9": {
"x": "1",
"y": "0",
"source": "net14",
"gate": "A2",
"drain": "VSS",
"width": "140" ,
},</td>
<td>"M8": {
“x”:”0”,
"y": "0",
"source": "net26",
"gate": "A1",
"drain": "net14",
"width": "140",
},</td>
<td>"M5": {
"x": "3",
"y": "0",
"source": "Z",
"gate": "net26" ,
"drain": "vsS",
"width": "140",
},</td>
<td>"M5_2": {
"x": "2",
"y": "0",
"source": "VSS",
"gate": "net26",
"drain": "Z",
"width": "140" ,
}</td>
</tr>
</table>

评分结果：

PS E: \eda_project> python evaluator.py AN2D2.json AN2D2 cells.spi

Cell score 96.685586 (width: 4, bbox: 3.500000, pin access: 0.062500, symmetric: 10, drc: 10, runtime: 0s)

b. 标准单元 AOI222D2

布局结果：

"placement": {
"M4": {
"x": "10",
"y": "1",
"source": "net37",
"gate": "C1",
"drain": "VDD",
"width": "170" ，
},
	"M4_2": {
	"x": "1",
	"y": "1",
	"source": "net37",
	"gate": "C1",
	"drain": "VDD",
	"width": "170"，
	}
		"M5":{
		"x": "2",
		"y": "1",
		"source": "VDD",
		"gate": "C2",
		"drain": "net37",
		"width": "170",
		},
			"M5_2": {
			"x": "11",
			"y": "1",
			"source": "VDD",
			"gate": "C2",
			"drain": "net37",
			"width": "170" ,
			}

"M7": {
"x":"7",
"y": "1",
"source": "ZN",
"gate": "A2",
"drain": "net25",
"width": "170",
}
	"M7_2": {
	“x”:”4”,
	"y": "1",
	"source": "net25",
	"gate": "A2",
	"drain": "ZN",
	"width": "170",
	}
		"M6":{
		"x": "6",
		"y": "1",
		"source": "net25",
		"gate": "A1",
		"drain": "ZN",
		"width": "170",
		},
			"M6_2": {
			"x": "5",
			"y": "1",
			"source": "ZN",
			"gate": "A1",
			"drain": "net25",
			"width": "170"，
			}

"M2": { "x": "0", "y": "1", "source": "net25", "gate": "B1", "drain": "net37", "width": "170" , },	"M2_2": { "x": "9", "y": "1", "source": "net25", "gate": "B1", "drain": "net37", "width": "170" , },	"M3":{ "x": "12", "y": "1", "source": "net37", "gate": "B2", "drain": "net25", "width": "170" },	"M3_2": { "x": "13", "y": "1", "source": "net25", "gate": "B2", "drain": "net37", "width": "170", },
"M14"{: "x": "0", "y": "0",	"M15": { "x": "12", “y”:”0”,	"M16":{ x": "1", "y": "0",	"M17": { "x": "2", "y": "0",

"source": "net075", "gate": "B1", "drain": "ZN", "width": "140", }	"source": "vsS", "gate": "B2", "drain": "net075", "width": "140", }	"source": "ZN", "gate": "C1", "drain": "net067", "width": "140", }	"source": "net067", "gate": "C2", "drain": "VSS", "width": "140", }

"M18": {
"x":"7",
"y":"0"
"source": "net063",
"gate": "A2",
"drain": "VSS",
"width": "140",
},
	"M19": {
	"x": "6",
	"y": "0",
	"source": "ZN",
	"gate": "A1",
	"drain": "net063",
	"width": "140"
	},
		"M12": {
		"x": "10",
		"y": "0",
		"source": "ZN",
		"gate": "C1",
		"drain": "net38",
		"width": "140" ,
		},
			"M13": {
			"x": "11",
			"y": "0",
			"source": "net38",
			"gate": "C2",
			"drain": "VSS",
			"width": "140",
			}

"M9": {
"x": "3",
"y": "0",
"source": "VSS",
"gate": "A2",
"drain": "net50",
"width": "140",
},
	"M8": {
	"x": "5",
	"y": "0",
	"source": "net50",
	"gate": "A1",
	"drain": "ZN",
	"width": "140",
	}
		"M0": {
		"x": "9",
		"y": "0",
		"source": "net62",
		"gate": "B1",
		"drain": "ZN",
		"width": "140",
		}
			"M1":{
			"x": "8",
			"y": "0",
			"source": "VSS",
			"gate": "B2",
			"drain": "net62",
			"width": "140",
			}

评分结果：

PS E:\eda_project> python evaluator.py AOI222D2.json AOI222D2 cells.spi

Cell score 84.246612 (width: 14,bbox:86.000000, pin access: 0.189731, symmetric: 6, drc: 10, runtime: 0s)

c. 标准单元 SDQNV4

布局结果：

<table>
<tr>
<td>"placement": {
"M1": {
"x": "2",
"y": "1",
"source": "net134",
"gate": "D",
"drain": "net131",
"width": "200"，
},</td>
<td>"M27": {
"x": "1",
"y": "1",
"source": "VDD",
"gate": "SE",
"drain": "net134",
"width": "200"，
}</td>
<td>"M29": {
"x": "3",
"y": "1",
"source": "net131",
"gate": "SEN",
"drain": "net126",
"width": "200"，
}</td>
<td>"M28": {
"x": "4",
"y": "1",
"source": "net126",
"gate": "SI",
"drain": “VDD",
"width": "200"，
}</td>
</tr>
<tr>
<td>"M13": {
"x": "12",
"y": "1",
"source": "c",
"gate": "cn",
"drain": "VDD",
"width": "200"
},</td>
<td>"M11": {
"x": "10",
"y": "1",
"source": "VDD",
"gate": "CK",
"drain": "cn",
"width": "200"
},</td>
<td>"M23": {
“:x”:9,
"y": "1",
"source": "QN",
"gate": "net162",
"drain": "VDD",
"width": "200"
},</td>
<td>"M23_finger1": {
"x":"6",
"y":"1",
"source":"QN",
"gate":"net162",
"drain":"VDD",
"width": "200"
}</td>
</tr>
<tr>
<td>"M21":{
"x":"7",
"y": "1",
"source": "VDD",
"gate": "net162",
"drain": "net167",
"width": "120"
},</td>
<td>"M5": {
"x": "13",
"y": "1",
"source": "VDD",
"gate": "net199",
"drain": "net163",
"width": "200"
}</td>
<td>"M15": {
"x": "16",
"y": "1",
"source": "net162",
"gate": "cn",
"drain": "net163",
"width": "200"
},</td>
<td>"M20": {
"x": "15",
"y": "1",
"source": "net94",
"gate": "c",
"drain": "net162",
"width": "200"
}</td>
</tr>
</table>

"M7": {
"x": "21",
"y": "1",
"source": "net90",
"gate": "cn",
"drain": "net199",
"width": "120"
},
	"M19":{
	"x": "0",
	"y": "1",
	"source": "net94",
	"gate": "net167",
	"drain": "VDD",
	"width": "200"
	},
		"M8":{
		"x": "20",
		"y": "1",
		"source": "VDD",
		"gate": "net163",
		"drain": "net90",
		"width":"120"
		},
			"M24": {
			"x": "19",
			"y": "1",
			"source": "SEN",
			"gate": "SE",
			"drain": "VDD",
			"width": "200",
			}

"M33":{
"x": "22",
"y": "1",
"source": "net199",
"gate": "c",
"drain": "net131",
"width": "200"
}
	"M31": {
	"x":"4",
	"y": "0",
	"source": "net191",
	"gate": "SI",
	"drain": "net187",
	"width": "200"
	}
		"M12": {
		"x": "8",
		y:"0",
		"source": "vSS",
		"gate":"cn",
		"drain": "c",
		"width": "200"
		}
			"M25": {
			"x": "19",
			"y": "0",
			"source": "VSS",
			"gate": "SE",
			"drain": "SEN",
			"width": "200"
			}

"M32": {
"x": "'16",
"y": "0",
"source": "net199",
"gate": "cn",
"drain": "net191",
"width": "200"
}
	"M26": {
	"x": "18",
	"y": "0",
	"source": "net195",
	"gate":"D",
	"drain": "vSS",
	"width": "200"
	}
		"M2": {
		"x": "17",
		"y":"0",
		"source": "net191",
		"gate": "SEN",
		"drain": "net195",
		"width": "200"
		}
			"M10": {
			"x":"10",
			"y":"0",
			"source":"cn",
			"gate":"CK",
			"drain":"VSS",
			"width": "200"
			},

"M22": {
"x": "7",
"y": "0",
"source": "QN",
"gate": "net162",
"drain": "VSS",
"width": "200"
}
	"M22_finger1":{
	"x": "6",
	"y": "0",
	"source": "VSS",
	"gate": "net162",
	"drain": "QN",
	"width": "200"
	}
		"M18": {
		"x": "11",
		"y": "0",
		"source": "VSS",
		"gate": "net162",
		"drain": "net167",
		"width": "120"
		}
			"M4": {
			"x": "13",
			"y": "0",
			"source": "net163",
			"gate": "net199",
			"drain": "VSS",
			"width": "200"
			}

"M30": {
"x": "5",
"y": "0",
"source": "net187",
"gate": "SE",
"drain": "VSS",
"width": "200"
}
	"M14": {
	"x": "22",
	"y": "0",
	"source": "net162",
	"gate": "c",
	"drain": "net163",
	"width": "200"
	. }
		"M16": {
		"x":"0",
		"y" :"0",
		"source": "net155",
		"gate": "net167",
		"drain": "'vsS",
		"width": "200"
		}
			"M9":{
			"x": "14",
			"y": "0",
			"source": "VSS",
			"gate": "net163",
			"drain": "net151",
			"width": "120"
			}

"M17": {
"x": "21",
"y": "0",
"source": "net155",
"gate":"cn",
"drain": "net162",
"width": "200"
}
	"M6":{
	"x": "15",
	"y": "0",
	"source": "net151",
	"gate": "c",
	"drain": "net199",
	"width": "120"
	}		

评分结果：

PS E:\eda_project> python evaluator.py SDQNV4.json SDQNV4 cells.spi

Cell score 39.988629 (width: 23, bbox: 194.500000, pin access: 0.171939, symmetric: -2, drc: 10, runtime: 0s)

2. 结果分析

上面展示的标准单元中，AN2D2 含有 8 个 MOS 管，算法生成的布局结果中布局宽度为 4，布线复杂度为 3.5，引脚密度为 0.00625,所有 MOS 管均产生栅极共用，没有出现 notch。AOI222D2 含有 24 个 MOS 管，算法生成的布局结果中布局宽度为 14，布线复杂度为 86.0，引脚密度为 0.189731，有 4 个 MOS 管未产生栅极共用，未出现 notch。标准单元 SDQNV4 含有 32 个 MOS 管，在部分晶体管折叠之后有 34 个 MOS 管，生成的布局结果中，布局宽度为 23，布线复杂度为 194.5，引脚密度为 0.171939，有 12 个 MOS 管未产生栅极共用，布局未产生 notch。

算法自动生成的布局中均避免出现有源区凹槽，并且通过上述分析可知，对于小规模的

标准单元，算法生成的布局结果各项指标较优，对于大规模的标准单元，算法生成的布局结果不够好，有部分晶体管未产生有源区共用。

3. **算法运行时间**

Cell:AN2D2 width:4 bbox:3.5 pin_access:0.0625 symmetric:10 drc:10 runtime:0.s

Cell:AN3D2 width:5 bbox:4.5 pin_access:0.0471404520791 symmetric:10 drc:10 runtime:0.s

Cell:AN4D2 width:7 bbox:7.5 pin_access:0.0927884361198 symmetric:8 drc:10 runtime:5.656s

Cell:AO21D2 width:6 bbox:12.5 pin_access:0.117851130198 symmetric:8 drc:10 runtime:4.s

Cell:AO211D2 width:7 bbox:10.5 pin_access:0.154647393533 symmetric:8 drc:10 runtime:5.062s

Cell:AOI222D2 width:15 bbox:87. pin_access:0.177664362916 symmetric:4 drc:10 runtime:15.907s

Cell:CKAN2D2 width:6 bbox:9.5 pin_access:0.208333333333 symmetric:8 drc:10 runtime:0.015s

Cell:CKLHQD2 width:16 bbox:71.5 pin_access:0.123773277482 symmetric:6 drc:10 runtime:19.86s

Cell:CLKNAND2V4 width:4 bbox:8. pin_access:0.117851130198 symmetric:10 drc:10 runtime:0.062s

Cell:SDQNV4 width:22 bbox:118.pin_access:0.155071890079 symmetric:0 drc:10 runtime:26.766s

Cell:AO22D2 width:7 bbox:12.5 pin_access:0.154647393533 symmetric:8 drc:10 runtime:6.469s

Cell:AOI211OPTREPBD2 width:19 bbox:75. pin_access:0.16368230664 symmetric:-5 drc:10 runtime:19.937s

Cell:AOI21D2 width:6 bbox:17. pin_access:0.207289049397 symmetric:10 drc:10 runtime:0.016s

Cell:AOI21OPTREPBD2 width:11 bbox:44. pin_access:0.175391468224 symmetric:6

drc:10 runtime:14.672s

Cell:AOI221D2 width:14 bbox:78. pin_access:0.202730790071 symmetric:2 drc:10 runtime:15.718s

Cell:AOI22D2width:8 bbox:32. pin_access:0.182859235479 symmetric:10 drc:10 runtime:0.016s

Cell:AOI31D2width:10 bbox:34. pin_access:0.146287388383 symmetric:6 drc:10 runtime:11.625s

Cell:AOI32D2width:10 bbox:45. pin_access:0.156569118567 symmetric:10 drc:10 runtime:0.125s

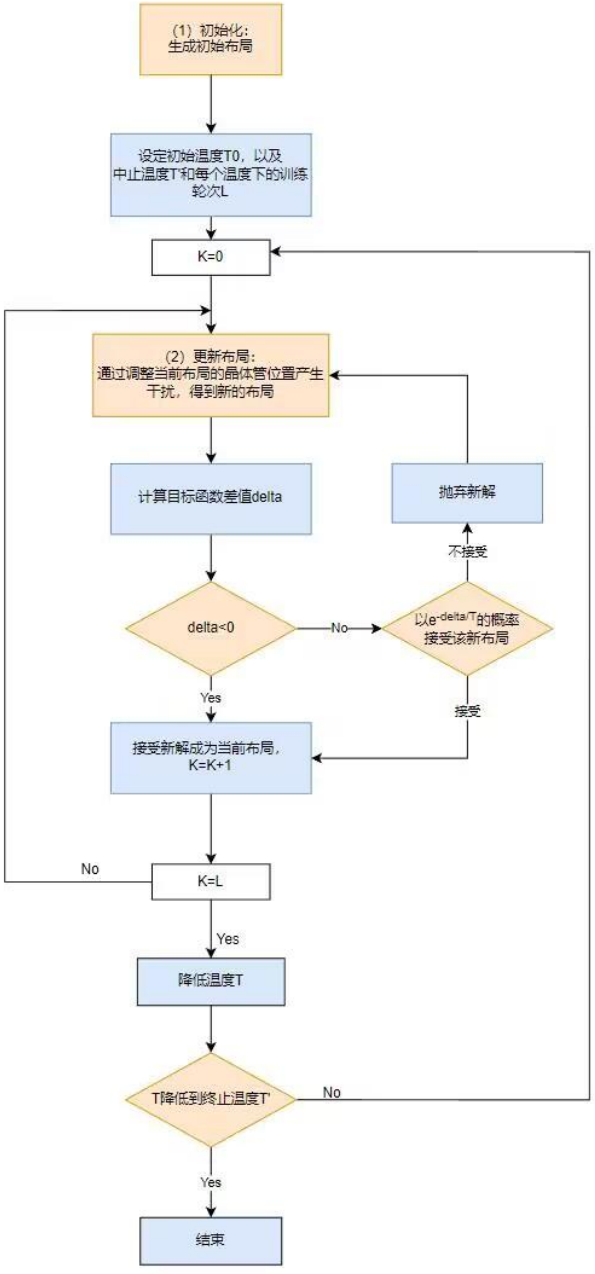
Cell:AOI33D2width:13 bbox:58. pin_access:0.079242885175 symmetric:8 drc:10 runtime:20.641s

Cell:BUFFD2 width:3 bbox:2.5 pin_access:0. symmetric:10 drc:10 runtime:0.015s

二、 主要技术路线及实现方式

1. 算法简介

我们使用了模拟退火算法来进行标准单元的版图自动生成。算法具体流程如下图所示。首先初始温度为T_0,在每个温度下训练 L 次。对于每轮次，更新当前布局并计算新布局的目标函数的值，如果大于当前布局的值，则一定更新为新布局；否则，以$e^{-\frac{c(s_1)-c(s_2)}{T}}$的概率接受新布局，其中 T 为当前温度，$c(s_2)$为新布局下目标函数的值，$c(s_1)$为当前布局下目标函数的值。



2. 一些参数及函数说明

（1）温度

初始温度$T_0 = N \times 1000$，下一温度更新方法为 $T = T \times 0.8$。

（2）训练轮次

每个温度下训练轮次 $L = N \times 100$。

（3）目标函数

目标函数 C(s):计算方法与 evaluator.py 文件提供的一致。

3. 算法描述

（1）初始化

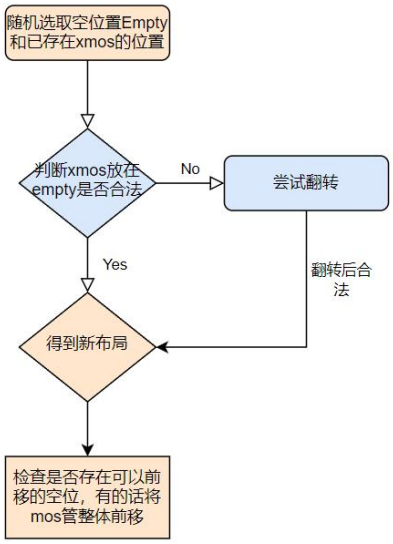
从文件中依次读取 NMOS 和 PMOS 的信息并分别存入到对应的 NMOS_list 或 PMOS_list 的列表中。

首先将第一个 NMOS 摆放到位置 0，接着摆放一个 PMOS，检查位置 0 对于该 PMOS 是否合法（也就是源珊漏极符合要求，且摆放不会带来源区凹槽）。摆放完成后，接着摆放下一个 NMOS……交叉着进行摆放，直到所有 MOS 管都被摆放完成。

（2）更新布局

对于 NMOS 和 PMOS，更新的步骤如下图所示。

具体来说，分别执行以下操作：遍历 MOS 管的坐标列表，分别找到所有空位置和所有已被占用的位置，分别放到两个列表中，然后分别从两个列表中随机选一个空位置 Empty 和已有晶体管 xMOS 的位置 Occupied，检查 xMOS 放在位置 Empty 是否合法，如果合法，则将 xMOS 放到空位置 Occupied，否则尝试翻转 xMOS，或者 Empty 左右的 mos 管集合（指位置连续的一个或几个 mos 管），来尝试找到合法布局。



三、 遇到的主要问题及解决方法

1. 数据结构的设计

需要有存储引脚、线网、MOS 管的数据结构，分别用两个数组存储 PMOS 和 NMOS 集合，用两个数组存储 PMOS 和 NMOS 的物理布局，将两种 MOS 管能够统一操作，增强代码的复用性。

2. 算法设计

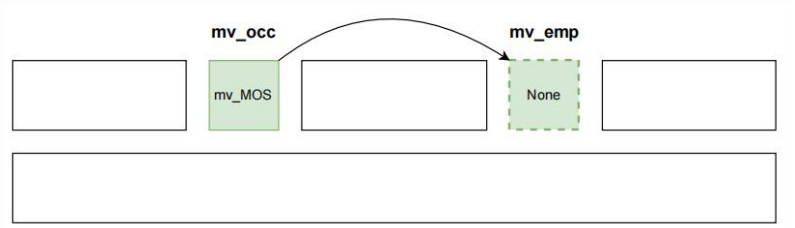
（1）布局初始化

初始化布局不考虑过多的优化，直接按照网表文件中晶体管的顺序进行排布，放置一 MOS 管时，能够产生共用的有源区则将 MOS 管相邻放置，否则相隔一个单位长度放置。在算法执行过程中，需要保证 PMOS 的布局数组与 NMOS 的布局数组长度相同，以检查相同位置的 PMOS 与 NMOS 的栅极。开始时，初始化并没有考虑晶体管折叠的情况，样例的 PMOS 数量和 NMOS 数量相同，因此我们得到结论：布局宽度最差的情况是两种 MOS 管的数量之和，统计的晶体管数量是未产生折叠情况的数量。在进一步的测试样例中，需要对晶体管进行折叠，折叠后产生新的 MOS 管，此时标准单元内部 MOS 管数量变多，在先前的结论下，得到的 PMOS、NMOS 布局数组长度不相同，造成了数组下标越界的错误，所以修改结论为，假设晶体管折叠后的 PMOS 数量为n_1，NMOS 数量为n_2，则布局宽度最差的情况为 $2 \times \max(\mathrm{n}_1, \mathrm{n}_2)$。

（2）产生干扰

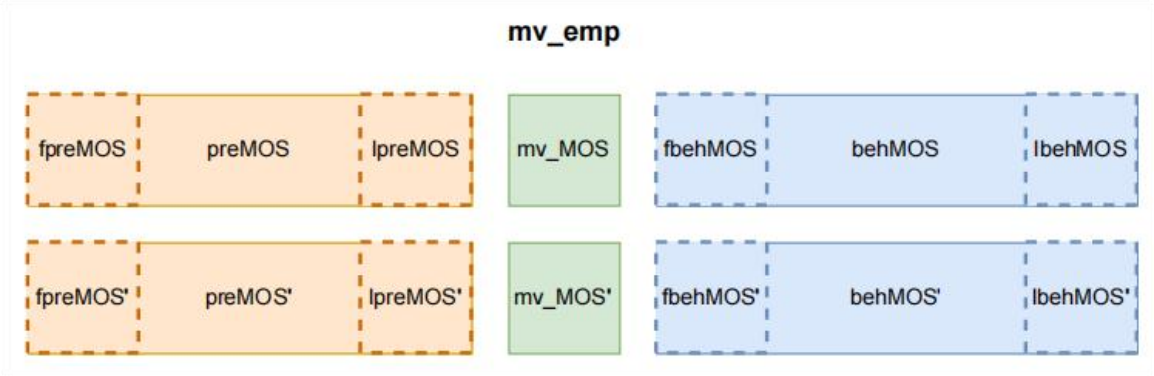
优化算法选择模拟退火算法，在设计如何产生干扰以及干扰的合法性检测时遇到了许多问题。

关于产生干扰的设计，我们给出了许多方案，最终采用了思路最清晰的方案：将布局数组位置状态分为占用、空这两类，统计布局中占用位置以及空位置，然后随机的选择一个占用位置和一个空位置，交换占用位置和空位置，并检测交换后的布局是否合法，尝试将占用位置的 MOS 管移动到空位置上，完成一次干扰。其中，交换两个占用位置的动作可以视作交换占用位置和空位置的动作组合。



开始时，对于干扰合法性检测的算法是：占用位置的 MOS 管的源极必须和空位置左边的同类 MOS 管漏极相同，其漏极必须和空位置右边的同类 MOS 管源极相同，此外，其栅

极必须和与空位置在相同位置的不同类 MOS 管栅极相同，这样严格的条件使得优化效果并不理想，在对调试过程中，发现许多轮次的优化都是不合法的，因此需要拓广合法检测的范围，我们对合法性检测算法进行了如下调整：



计算空位置 mv_emp 之前已经产生源漏共用的 MOS 管集合 preMOS，空位置 mv_emp 之后已经产生源漏共用的 MOS 管集合 behMOS，以及与 preMOS 位置相同的另一类 MOS 管集合 preMOS'，与 behMOS 位置相同的另一类 MOS 管集合 behMOS'，由于 MOS 管可以翻转，因此可以通过翻转 preMOS、mv_MOS 以及 behMOS 尽量产生源漏共用，因此这三者是否翻转的情况有：

翻转组合			满足条件	解释
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 与 preMOS 源漏共用 2.mv_MOS 与 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用
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 与 preMOS 源漏共用 2.mv_MOS 与翻转的 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用
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.翻转的 mv_MOS 与 preMOS 源漏共用 2.翻转的 mv_MOS 与 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用
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.翻转的 mv_MOS 与 preMOS 源漏共用 2.翻转的 mv_MOS 与翻转的 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用

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 与翻转的 preMOS 源漏共用 2.mv_MOS 与 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用
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 与翻转的 preMOS 源漏共用 2.mv_MOS 与翻转的 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用
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.翻转的 mv_MOS 与翻转的 preMOS 源漏共用 2.翻转的 mv_MOS 与 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用
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.翻转的 mv_MOS 与翻转的 preMOS 源漏共用 2.翻转的 mv_MOS 与翻转的 behMOS 源漏共用 3.mv_MOS 与 mv_MOS'栅极共用

注：

① MOS.s 表示 MOS 管源极，MOS.g 表示 MOS 管栅极，MOS.d 表示 MOS 管漏极

② 对于翻转组合的情况，0 表示不进行翻转，1 表示进行翻转

并且需要翻转同位置的 MOS'集合。若 MOS 集合内的 MOS 管数量为 0 或 1 个，则无需翻转；否则翻转相同位置的 MOS″集合，并判断能否与 mv_MOS'产生源漏共用。

若能够通过某一翻转情况的条件，则可以将 mv_MOS 放置在空位置。

在翻转 MOS 集合时，需要注意的是：需要交换每个内部 MOS 的源极和漏极，在 MOS 集合的 MOS 管数量大于等于 2 时，也需要反转 MOS 集合。

四、 主要创新点

在我们的工作中，使用了 OCmal 函数式编程来解决标准电路单元版图的自动生成，其高阶函数、函数式数据结构、模式匹配、静态类型系统、并发性、开发周期短等特点在我们的工作中发挥了巨大优势。与此同时，我们使用了模拟退火算法并修改其参数，让其能够在我们的模型上获得不错的得分。同时，我们将两种 MOS 管统一操作，增强了代码的复用性。以下，将详细阐述我们的创新和优势。

1. 统一操作接口

分别用两个数组存储 PMOS 和 NMOS 集合，用两个数组存储 PMOS 和 NMOS 的物理布局，将两种 MOS 管能够统一操作，增强了代码的复用性，有助于提高代码的通用性和可维护性。

2. 改进模拟退火算法

（1）布局最差宽度

晶体管折叠后的 PMOS 数量为n_1，NMOS 数量为n_2，则布局宽度最差的情况为 $2 \times \max(\mathrm{n}_1, \mathrm{n}_2)$。

（2）修改参数

设置初始温度为$T_0 = N \times 1000$,在每个温度下训练 $\mathrm{L} = \mathrm{N} \times 100$ 次，每轮次更新当前布局并计算新布局的目标函数 $\mathrm{C(s)}$的值，设置下一温度更新方法为 $\mathrm{T} = \mathrm{T} \times 0.8$。

（3）产生干扰

将布局数组位置状态分为占用、空这两类，统计布局中占用位置以及空位置，然后随机的选择一个占用位置和一个空位置，交换占用位置和空位置，并检测交换后的布局是否合法，尝试将占用位置的 MOS 管移动到空位置上，完成一次干扰。

3. 采用 OCaml 语言

在我们的项目中，使用了 OCaml 函数式编程。在实现标准单元版图的自动生成中，使用 OCaml 函数式编程语言有以下优势：

高阶函数	OCaml 支持高阶函数，可将函数作为参数传递给其他函数，或将函数作为返回值。
函数式数据结构	在 OCaml 中，数组是不可变的，因此在删除元素时，不会直接修改原始数组，而是创建了新的数组来代表删除元素后的结果。这种不可变性可以避免一些常见的错误，比如在命令式编程中可能会因为修改原始数组而导致意外的副作用。
模式匹配	在 OCaml 中，模式匹配是一种强大的工具，可以用来处理各种数据结构，包括数组、列表、元组等。在这段代码中，使用了模式匹配来获取数组中指定位置的元素，以及将数组分割成左右两部分。

静态类型系统	OCaml 的静态类型系统有助于在编译时捕获类型错误，提高了代码的稳定性。
并发性	OCaml 的轻量级并发模型适用于处理并发任务，可以为标准单元版图自动生成任务完成加快速度。
表达能力	OCaml 提供了高度抽象的编程能力，可以更自然地表达复杂的算法和逻辑。
开发周期短	OCaml 强大的类型推导、模式匹配、高阶函数和函数式编程、并发性和轻量级线程等特点，缩短开发周期。

为了更好地理解 OCaml 在标准单元电路版图的自动生成中的优势，如下是我们为实现赛题写的一段 OCaml 代码，其功能是定义一个函数 ary_drop，它接受一个数组 ary 和一个位置 pos 作为参数，然后从数组中删除索引为 pos 的元素。我们将 OCaml 语言分别与 C 语言、Python 语言进行对比。

Ocaml 语言	C 语言	Python 语言

let ary_drop ary pos = (*删除索引为 pos 的元素*)
let left_ary = Array.sub ary 0 pos
and right_ary = Array.sub ary (pos+1) (Array.length ary-pos-1) in
Array.append left_ary right_ary
	void ary_drop(int ary[], int size, int pos) {
	if (pos < 0 || pos >= size) {
	// 无效位置，不执行任何操作
	return;
	}
	// 创建左边
	int* left_ary = (int*)malloc(pos * sizeof(int));
	for (int i = 0; i < pos; ++i) {
	left_ary[i] = ary[i];
	}
	// 创建右边
	int* right_ary = (int*)malloc((size - pos - 1) * sizeof(int));
	for (int i = pos + 1, j = 0; i < size; ++i, ++j) {
	right_ary[j] = ary[i];
	}
	// 合并左边和右边
	int new_size = size - 1;
	int* new_ary = (int*)malloc(new_size * sizeof(int));
	for (int i = 0; i < pos; ++i) {
		def ary_drop(ary, pos):
		# 检查位置是否在有效范围内
		if 0 <= pos < len(ary):
		# 创建左侧子数组，包括指定位置之前的元素
		left_ary = ary[:pos]
		
		# 创建右侧子数组，从指定位置的下一个位置开始到末尾
		right_ary = ary[pos + 1:]
		
		# 合并左右子数组，模拟删除指定位置的元素
		result_ary = left_ary + right_ary
		
		# 返回结果数组
		return result_ary
		else:
		# 如果位置无效，返回原始数组
		return ary

```
            new_ary[i] = left_ary[i];
        }
        for (int i = pos, j = 0; i < new_size;
        ++i, ++j){
            new_ary[i] = right_ary[j];
        }
        // 释放内存
        free(left_ary);
        free(right_ary);
        // 更新数组
        for (int i = 0; i < new_size; ++i) {
            ary[i] = new_ary[i];
        }
        // 释放内存
        free(new_ary);
}
```

在 OCaml 语言中，let left_ary = Array.sub ary 0 pos 这一行创建了一个子数组 left_ary，包含了原数组 ary 中索引从 0 到 pos-1 的元素。let right_ary = Array.sub ary (pos+1) (Array.length ary-pos-1)这一行创建了另一个子数组 right_ary，包含了原数组 ary 中索引从 pos+1 到数组末尾的元素。Array.append left_ary right_ary 最后一行使用 Array.append 函数将两个子数组连接起来，形成一个新的数组，即删除了原数组中索引为 pos 的元素。在 C 语言中，使用动态内存分配来创建左侧和右侧的子数组，并使用循环来填充这些子数组。最后，通过将左右两个子数组连接起来，创建一个新的数组来模拟删除指定位置的元素。在 Python 语言中，使用切片操作来创建左侧和右侧的子数组，并通过连接这两个子数组来实现删除指定位置的元素。

基于上述标准单元电路版图的自动生成代码实例，可以看到 OCaml 有以下优势：

	OCaml 语言	C 语言	Python 语言
函数式语言结构	OCaml 是一种函数式编程语言，ary_drop 函数表现出其简洁和表达力。使用 Array.sub 和 Array.append 函数反映了其函数式风格。	C 中的内存管理和数组操作涉及手动分配和释放，使得代码更冗长。	Python 是一种多范式语言，支持函数式编程，但更偏向命令式。在实例中的列表切片较为简洁，但缺乏 OCaml 中模式匹配的优雅性。
模式匹配	OCaml 具有强大的模式匹配，可在涉及数据结构的场景中	C 不支持本地模式匹配，实现类似功能通常需要多个条件	Python 通过元组解包等功能支持有限的模式匹配，但不具

	编写简洁而表达力强的代码。	语句和循环。	备 OCaml 中丰富的模式匹配能力。
静态类型系统	OCaml 的静态类型系统在编译时捕获许多错误，确保类型安全。函数签名明确指定了输入和输出的类型。	C 是静态类型语言，但可能需要更多手动类型处理。与类型相关的错误可能只在运行时显现。	Python 是动态类型语言，类型相关的错误可能只在执行时出现。
代码简短	由于其表达性语法、高级抽象和函数式编程特性，OCaml 代码通常简洁。	C 代码可能更冗长，特别是在处理手动内存管理和数组操作时。	Python 以代码可读性和简洁性著称，但在这个特定的例子中，OCaml 的函数式风格和模式匹配提供了类似的简洁性。
更好的性能	作为一种编译语言，OCaml 通常比解释性语言如 Python 表现更好。	作为低级语言，C 通常提供较高的性能。	Python 是解释性语言，相较于编译语言可能性能较低。

OCaml 在函数式编程特性、表达性语法、模式匹配功能、静态类型系统、模式匹配等方面有较强能力。总的来说，函数式编程语言拥有强大的抽象能力、不可变性和模式匹配等特性，使得代码更加简洁、可读性更高、潜在更安全，并且能够更容易地处理复杂的数据结构和逻辑。在完成标准单元版图自动生成时有不容小觑的优势。

五、 附录

1. 参考文献

[1] 马琪,王旭;CMOS 单元版图生成中的晶体管布局算法[J];电路与系统学报;2004 年 04 期.

[2] 马琪,罗小华,严晓浪;CMOS 单元版图生成算法综述[J];微电子学;2001 年 03 期.

[3] Schneider, Jan: Transistor-Level Layout of Integrated Circuits. - Bonn, 2014. - Dissertation, Rheinische Friedrich-Wilhelms-Universität Bonn.

[4] Kahng, A.B. et al. (2022) VLSI physical design: From graph partitioning to timing closure. Springer.