

开源 EDA 使用手册 3.0

目录

| | |
|----------------------------------|----|
| 后端设计流程介绍..... | 2 |
| 开源 EDA 流程 iFlow 介绍..... | 16 |
| 一、 Build iFlow..... | 16 |
| 二、 iFlow 目录结构..... | 17 |
| 三、 iFlow command..... | 18 |
| 四、 iFlow 顶层脚本介绍..... | 19 |
| 1、 顶层脚本..... | 19 |
| 2、 配置脚本..... | 20 |
| 五、 iFlow 流程介绍..... | 23 |
| 3、 综合..... | 23 |
| 4、 布局..... | 25 |
| 5、 CTS..... | 32 |
| 6、 filler..... | 33 |
| 7、 布线..... | 34 |
| 8、 版图..... | 36 |
| 开源 EDA 流程 iFlow 使用示例——更换设计..... | 39 |
| 一、 gcd 设计..... | 39 |
| 二、 uart 设计..... | 39 |
| 三、 aes_cipher_top 设计..... | 41 |
| 开源 EDA 流程 iFlow 使用示例——更换工艺库..... | 43 |
| 一、 nangate45..... | 43 |
| 二、 asap7..... | 45 |

后端设计流程介绍

一、芯片设计流程（从功能定义到回片）

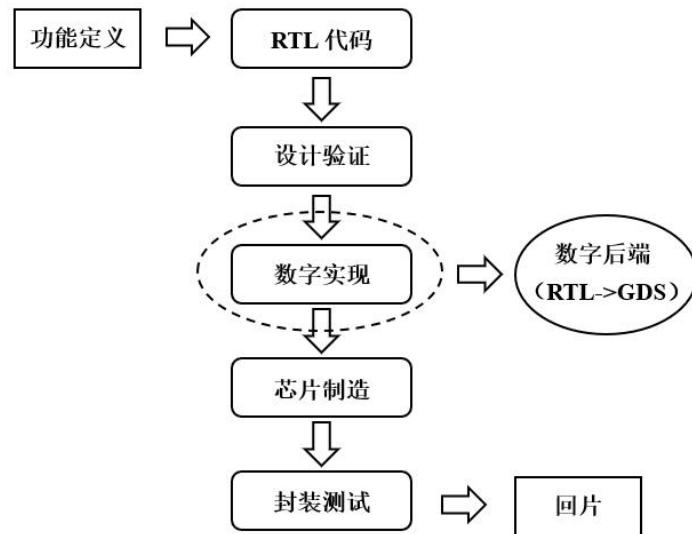


图 1 芯片设计流程

数字 IC 后端设计流程：（从综合后的网表到 GDS）

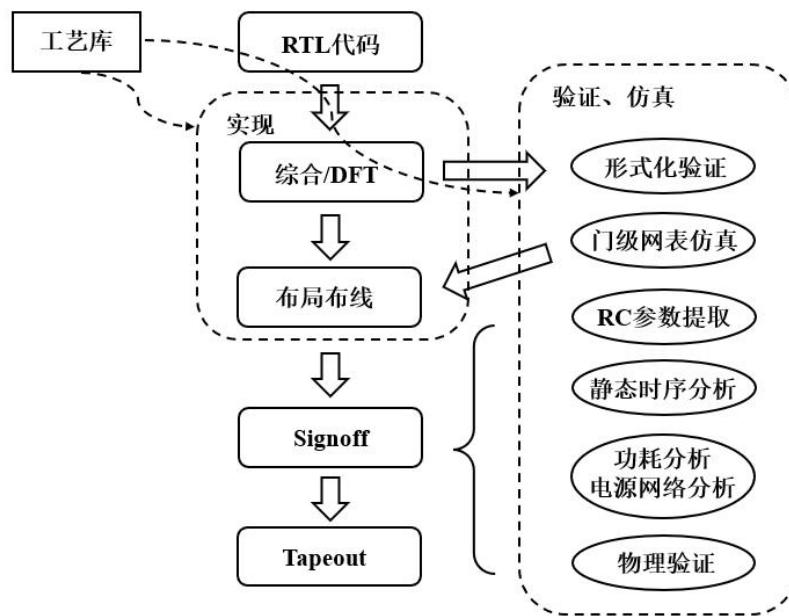


图 2 数字 IC 后端设计流程

二、综合（Synthesis）

1 逻辑综合：

将前端的 RTL 代码映射到特定的工艺库上，添加约束信息，对 RTL 代码进

行逻辑优化生成门级网表。

综合工具：yosys。

综合过程：Translation+ Optimization + Mapping

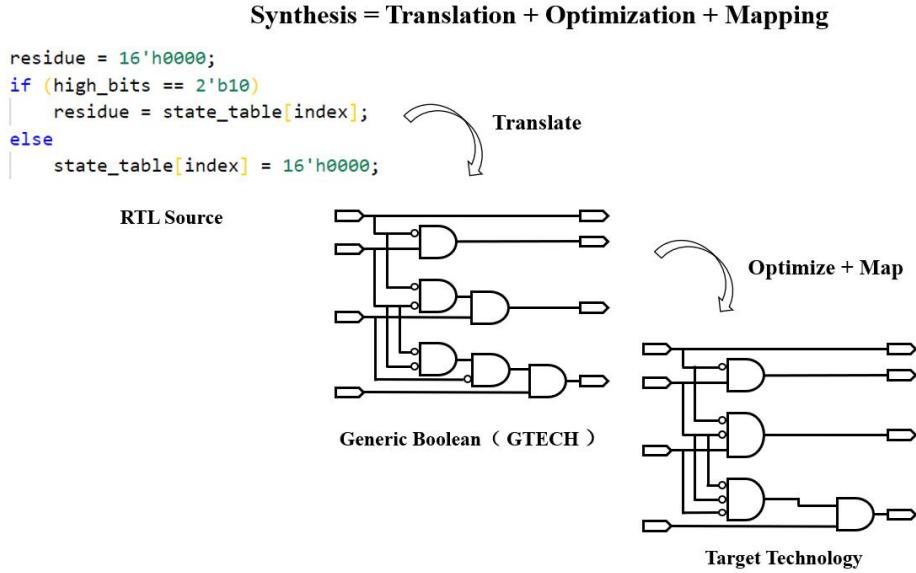


图 3 综合过程

Translation: yosys 利用其内部的 IP 库对 RTL 代码进行结构级和逻辑级的优化，生成 GTECH 格式的网表。（与工艺库无关）

Optimization: 根据约束信息（时序、面积、功耗约束）对单元进行结构优化。

Mapping: 将单元映射成工艺库中对应的门级电路。

综合的基本策略：

(1) **Top-down:** 以顶层模块为当前设计模块，一次性完成整个设计的综合。

优点：中规模设计优化效果好，模块边界不需额外处理。

缺点：对于超大规模设计综合速度慢，甚至无法收敛。

(2) **Bottom-up:** 先综合底层模块，顶层模块再调用综合生成的子模块，进而完成整个综合过程。

优点：降低对内存的需求，适合超大规模设计。

缺点：模块边界需额外处理。

2 输入文件：

2.1 RTL 代码（包括 Verilog、VHDL）。

2.2 库文件 (.lib 文件)，包含所有标准单元和宏单元的信息：

- (1) cell 的信息：功能、面积、功耗等。
- (2) 连线负载模型：电阻、电容。
- (3) 工作环境：工艺、电压、温度。
- (4) 涉及约束规则：最大最小电容、最大最小转换时间、最大最小扇出。

2.3 约束文件 (.sdc 文件)，包含设计中所有的时序约束：PVT (选 worst case)、Input drives (驱动能力)、transition times (转换时间)、Capacitive output loads (驱动电容负载)、内部寄生的 RC (线负载模型)

(1) 环境条件 PVT (process、voltage、temperature)：工艺、电压和温度等周围环境对器件延迟的影响。（fast、typical、slow）温度越高，速度越慢；电压越高，速度越快。

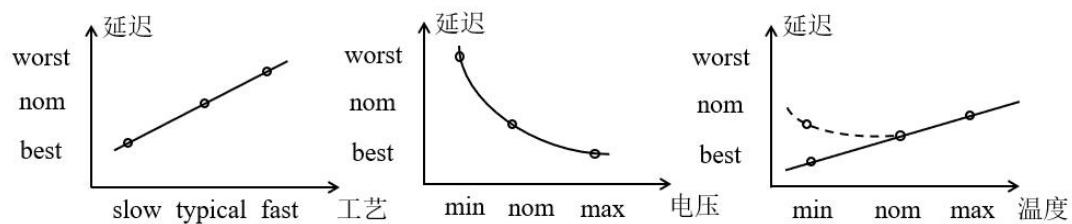


图 4 不同 PVT 条件对器件延迟的影响

- (2) 线负载模型：为了精确计算路径延迟，除了门单元延迟还有连线延迟。
- (3) 驱动强度：添加了驱动单元，输入会有一个斜率，（也就是告诉 DC 这个输入端口是由一个真实的外部单元驱动的，不是理想的）DC 就知道到达输入端口的转换时间，可以精确计算输入电路的延迟。
- (4) 电容负载：指定端口上的外部电容负载，可以精确计算输出电路的延迟。
- (5) 最大转换时间 (Max transition)：信号从 0->1 或 1->0 变化的时间。

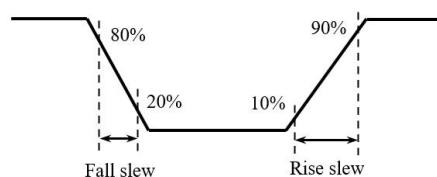


图 5 转换时间

最大扇出 (Max fanout)：单个逻辑门直接驱动的最大数目，图 6 中 BUF1

的扇出为 3。

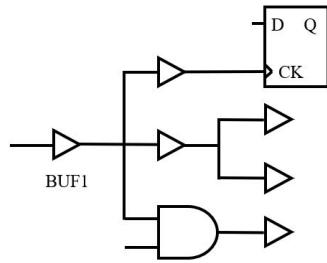


图 6 最大扇出示例

最大电容（Max capacitance）：输出可以驱动的最大负载值，图 7 中 BUF2 的 capacitance 值为 $0.05+0.03+0.02+0.07=0.17$ 。

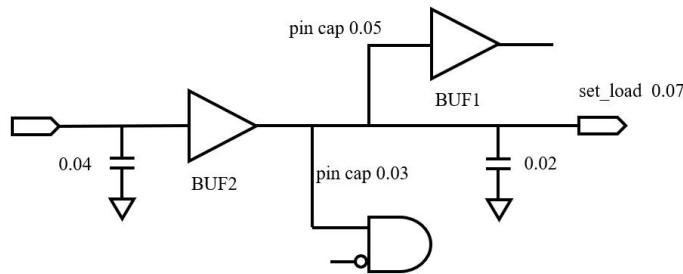


图 7 最大电容示例

(6) 时序约束：时序路径是点到点的数据通路，数据沿着时序路径进行传递。

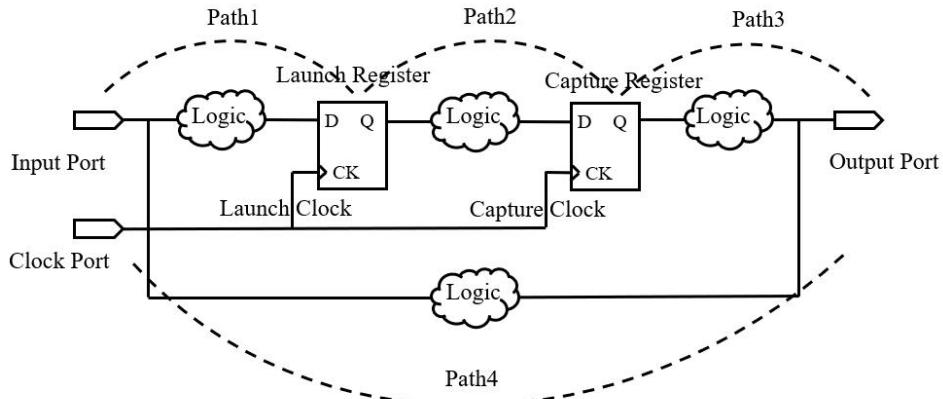


图 8 四种时序路径

Path1：输入端口到寄存器

假设外部输入电路延迟为 4ns，时钟周期 T_{clk} 为 10ns，则输入端到寄存器（内部逻辑）最大延迟为 $10 - 4 - T_{setup}(\text{ns})$ ，式中 T_{setup} 为建立时间。

Path2：寄存器到寄存器

延时应满足 $T_{comp} < T_{clk} - T_{c_q} - T_{setup}$ ，式中 T_{comp} 为组合逻辑延迟， T_{c_q} 为寄存

器 CK 端到 Q 端的延迟。

Path3：寄存器到输出端口

假设外部输出路径延迟为 4ns，时钟周期 T_{clk} 为 10ns，内部逻辑最大延迟为 $10 - 4 - T_{c_q}$ 。

Path4：输入端口到输出端口

组合逻辑延迟： $T_{clk} - T_{input_delay} - T_{output_delay}$

三、形式化验证 (Formal Verification)

1 形式化验证：

通过逻辑抽象的方法将两个设计进行对比，保证功能一致（只比较逻辑，不检查时序）。如图 9 所示，形式化验证包括 RTL vs netlist（综合后），netlist（综合后） vs netlist（PR 后），由于 yosys 综合后缺少商业工具 formality 需要的 svf 文件，因此无法进行 RTL vs netlist（综合后）的对比。

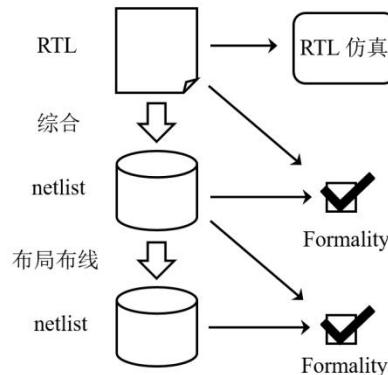


图 9 形式化验证流程

2 比较原理

(1) 将设计划分成多个 Logic Cone 和 Compare Point 组合。（Logic Cone：一组输入最终收敛到一个比较点的锥形逻辑，可以是寄存器输出、端口输入、黑盒的输出。Compare Point：比较点，包括寄存器输入、端口输出、黑盒的输入）

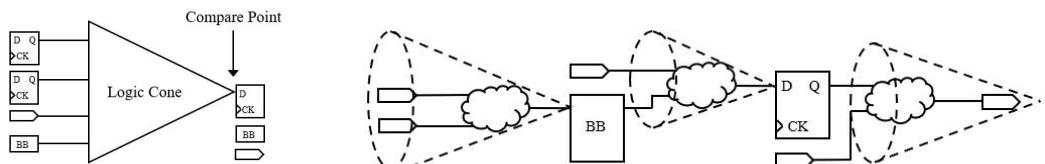


图 10 形式化验证比较原理

(2) 逻辑接口的比对，比对 reference design 和 implementation design 的

Compare point 是否匹配（该过程叫 match）。

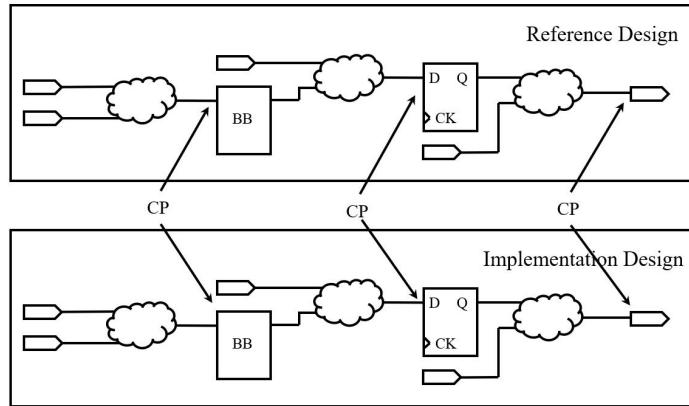


图 11 match 过程

(3) 逻辑功能的比对，给 Logic cones 激励，看 Compare point 输出结果是否一致（该过程叫 verify）。

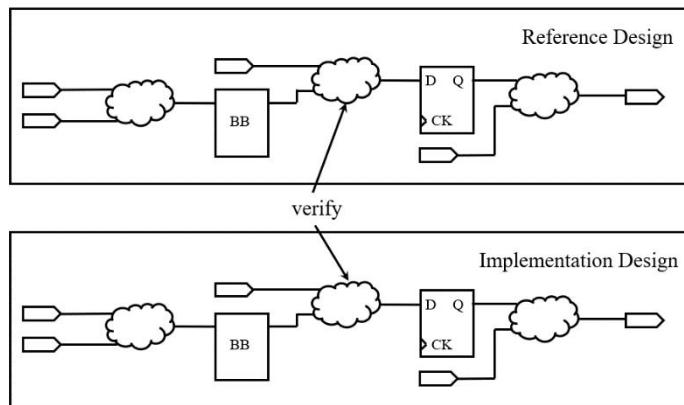


图 12 verify 过程

3 导致 unmatch 的原因

表 1 形式化验证 unmatch 的原因及解决办法

| 表现 | 可能原因 | 解决办法 |
|--------------------------|-------------------------------|---|
| ref 和 imp 不匹配的点数量不一样 | 设计被重命名了 | -手动设置 user match -打开 signature analysis 选项 |
| ref 中 unmatch 数量比 imp 中多 | 综合时逻辑优化掉冗余寄存器 | 不需特殊处理 |
| | 有些 missing cell 产生了 Black box | 读入 missing cell |
| ref 中 unmatch 数量比 imp 中少 | 综合过程中产生了额外逻辑 | 检查逻辑映射 |

四、布局布线

1 布局布线：

布局布线是将电路网表转换成物理版图的过程，其设计流程如图 13 所示。

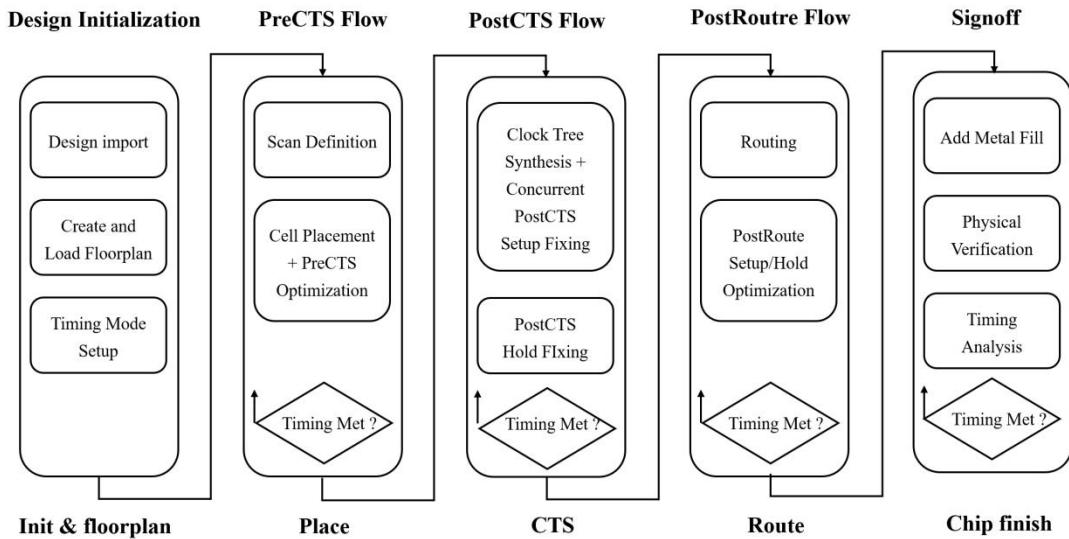


图 13 布局布线设计流程

2 Init

输入数据：

- (1) 综合或 DFT 后的门级网表。
- (2) 物理库：techlef 和 cell lef。
- (3) 时序库：.lib，商业工具还会用到.db。

3 Floorplan

- (1) 面积规划

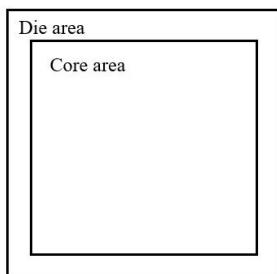


图 14 芯片面积规划

Die area：整个 layout 占的面积。

Core area：可用于摆放单元的面积。

标准单元利用率 = 标准单元总面积 / (Core area - 宏单元面积) , 初始经验值在 70%-80% 之间, 由于开源 EDA 工具布局布线功能尚未成熟, 利用率过高可能会导致影响绕线, 可通过降低利用率解决。

(2) 宏单元摆放位置规划

考虑的问题: 时序最优 (反复迭代) 、布线不阻塞 (反复迭代) 、供电是否可行、宏单元摆放导致的狭窄通道、宏单元 Port 位置。

宏单元摆放预留的窄通道一方面可以放标准单元, 另一方面有利于宏单元 Port 布线, 减少拥堵。

(3) Port 摆放位置规划

一般根据 Port 功能及信号走向分组摆放。

(4) 电源规划

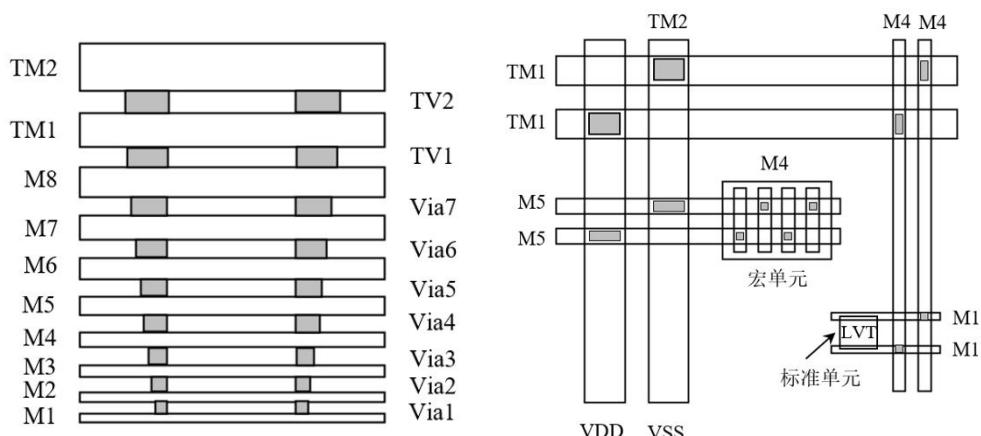


图 15 不同金属层的电源线与通孔

如图 15 所示, 电源线的奇数层用于横向布线, 偶数层用于纵向布线。TM1、TM2 用于设计电源主网络, M2-M8 用于次级电源网络, M1 为标准单元库电源网络。

供电能力满足要求:

$$I_{sup(TM2)} > P_{total}/V_{sup}$$

$$I_{sup(TM1)} > P_{total}/V_{sup}$$

$$I_{sup(M4)} > P_{stdcel}/V_{sup}$$

$$I_{sup(M5)} > P_{macrocel}/V_{sup}$$

式中, P_{total} 为整个设计的总功耗, P_{stdcel} 为标准单元总功耗, $P_{macrocel}$ 为宏单

元总功耗， V_{sup} 为供电电压。

电源规划需要考虑的因素：

- (1) 布线资源：可用于实现电源网络的金属层及最大供电能力。
- (2) 供电需求：给定电压下，最大电流需求。
- (3) 元件电源 PIN 脚：需了解如宏单元以及标准单元 VDD,VSS 的 PIN 及其与电源网络的大致连接方式。
- (4) 窄通道：需要特别关注窄通道标准单元的供电。

4 CTS 时钟树综合（Clock Tree Synthesis）

时钟树综合是保证从 Clock 的 root 点长到各个 sink 点的 clock buffer/inverter tree，时钟信号到达各个寄存器时钟端的时间偏差（skew）尽可能小。

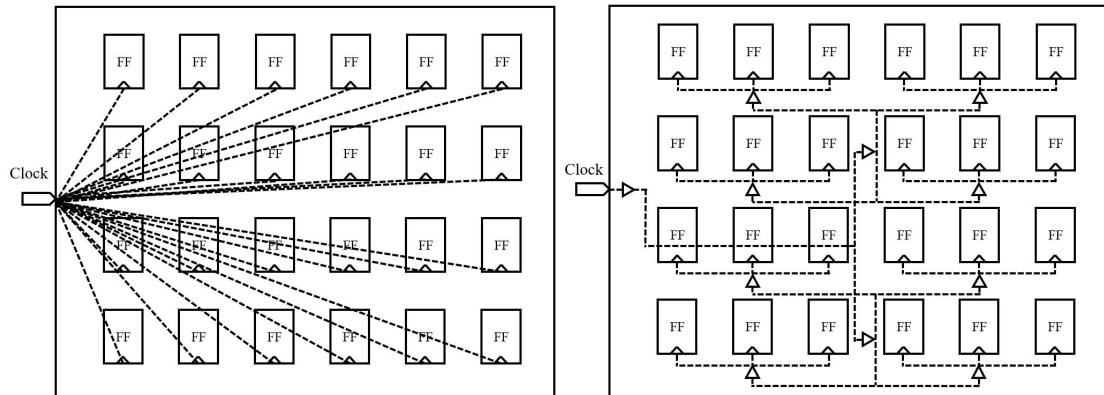


图 16 CTS 前后时钟线的布局

如图 16 所示，时钟树综合之前是一个时钟源扇出到很多寄存器的时钟端，时钟树综合之后是由多级的 buffer 构成了一个时钟树。

(1) 时钟源

外部晶振+内部时钟发生器+内部 PLL 产生的高频时钟+内部分频产生各种频率的时钟。

先从晶振或时钟发生器产生一定频率的时钟（如 25MHz），再经过 PLL 产生倍频时钟（高频时钟），最后再经过分频电路产生各种频点的时钟送到各个功能模块。

(2) 锁相环（PLL）数量

PLL 所占面积较大，因此 PLL 数量尽可能少，先对各个功能模块的时钟频率需求进行统计，设计分频器，最后计算出 PLL 的数量。

(3) PLL 位置

PLL 的位置决定了时钟树的长度（Clock Tree Latency），需要理清各路时钟的复用关系，PLL 倍频后的时钟供给哪些模块以及这些模块的位置。

(4) 时钟约束

第一部分是晶振 -> PLL

第二部分是 PLL -> clock gen 模块（产生分频时钟信号）

第三部分是分频器输出 -> 各个功能模块

CTS 步骤：

- (1) 生长时钟树
- (2) 优化时钟树及时序
- (3) 时钟树绕线
- (4) 手动调节时钟树
- (5) 查看时钟树报告，重复前面 4 个过程

5 Route

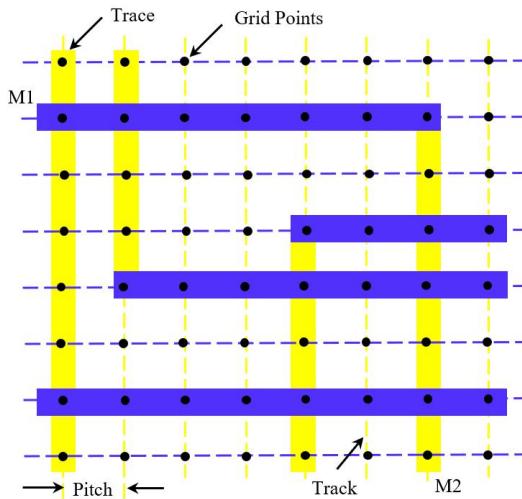


图 17 基于格点的布线图

track：黄色和蓝色的虚线，没有宽度，基于格点的布线要求所有的金属走线都要在 track 上。

pitch：两条 track 之间的间距。

trace：实际走在 track 上的金属走线，有宽度。

grid point：两条 track 的交点。

标准单元的高宽都是 pitch 的整数倍，布局时标准单元的 pin 都放到了 grid

point 上。

Route 的步骤：

(1) Global routing (全局布线)

全局布线是为了规划布线路径，确定大体位置及走向，并不会做实际的连线。

(2) Track assignment (track 分配)

把每一根线分配到 track 上，并对连线进行实际的布线，布线时尽可能使金属长而减少孔的数量，这个阶段不做 DRC 设计规则检查。

(3) Detail Routing (详细布线)

使用全局布线和 track 分配过程中产生的路径进行布线和布孔。由于 track 分配时只考虑尽量走长线，所以会有很多 DRC 违规产生，详细布线时使用固定尺寸的 sbox 来修复违规，sbox 是整个版图平均划分的小格子，小格子内部违规会被修复，但小格子边界的 DRC 违规就修复不了，这就需要在接下来的步骤中完成修复。

(4) Search and repair

修复在详细布线中没有完全消除的 DRC 违例，在此步骤中通过逐渐加大 sbox 的尺寸来寻找和修复 DRC 违例。

注：时钟树布线具有最高的优先权。

6 Insert fillers

使每一排的标准单元的 N 井连在一起，提高供电网络稳定性。

插冗余通孔：尽量将单通孔替换成双通孔提高成品率。

7 导出文件

导出版图 gds 文件和 Verilog 门级网表供后续流程使用。

五、静态时序分析 (Static Prime Analysis, STA)

静态时序分析是一种通过检查所有路径时序信息从而验证电路时序有效性的方法，其原理如图 18 所示。

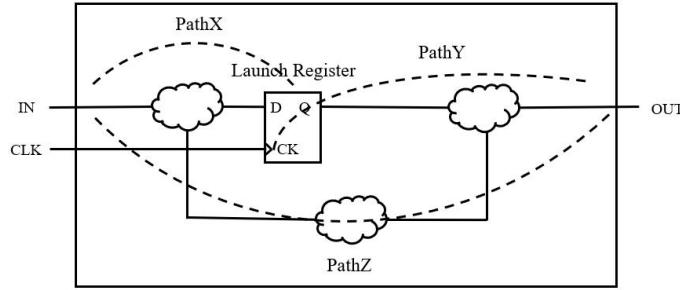


图 18 STA 原理

- (1) 把设计划分为若干路径
- (2) 分别计算每个路径的延迟
- (3) 检查每个路径的延迟是否满足要求

1 建立时间与保持时间

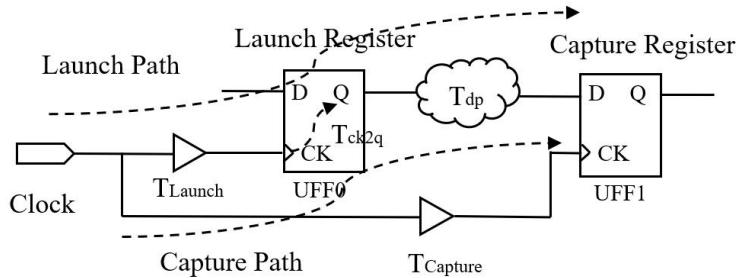


图 19 时序路径图

1.1 建立时间 T_{setup} :

时钟上升沿到来之前数据保持稳定的时间。

数据到达 UFF1 D 端的时间 arrival time:

$$T_a = T_{\text{launch}} + T_{\text{ck2q}} + T_{\text{db}}$$

满足 setup 所允许的最长时间 required time:

$$T_r = T_{\text{capture}} + T_{\text{clk}} - T_{\text{setup}}$$

$$\text{Slack} = T_r - T_a > 0, \text{ 即 } T_{\text{capture}} + T_{\text{clk}} - T_{\text{setup}} - T_{\text{launch}} - T_{\text{ck2q}} - T_{\text{db}} > 0,$$

令 $T_{\text{capture}} - T_{\text{launch}} = T_{\text{skew}}$ ，整理得：

$$T_{\text{skew}} + T_{\text{clk}} > T_{\text{setup}} + T_{\text{ck2q}} + T_{\text{db}}$$

修复 T_{setup} 时序违例方法：

- (1) 增加 T_{clk} : 降频。
- (2) 降低 T_{db} : 优化组合逻辑、划分流水线、减少关键路径上的负载。
- (3) 降低 T_{ck2q} : 换更快的时序逻辑单元，如 HVT->LVT。

1.2 保持时间 T_{hold} :

时钟上升沿到来之后数据保持稳定的时间。

数据到达 DFF1 的 D 端时间 arrival time:

$$T_a = T_{launch} + T_{ck2q} + T_{db}$$

满足 hold 所允许的最长时间 required time:

$$T_r = T_{capture} + T_{hold}$$

$$Slack = T_a - T_r > 0, \text{ 即 } T_{launch} + T_{ck2q} + T_{db} - T_{capture} - T_{hold} > 0,$$

令 $T_{capture} - T_{launch} = T_{skew}$, 整理得:

$$T_{skew} + T_{hold} < T_{ck2q} + T_{db}$$

修复 T_{hold} 时序违例方法:

- (1) 增大 T_{db} : 增加组合路径延迟, 插 buffer。
- (2) 降低 T_{skew} : 甚至采用负的 skew。

2 输入文件

- (1) db 文件: 和综合的 db 文件一致, 并需要 ss、ff 等多 corner 下的库
- (2) 门级网表
- (3) 约束文件.db
- (4) 反标文件: sdf、spef

SDF (Standard delay format) : 标准延时格式, 描述了设计中的时序信息, 指明了模块管脚和管脚之间的延迟、时钟到数据的延迟和内部连接延迟, sdf 文件可直接用于电路后仿。

SPEF(standard parasitic exchange format) : 标准寄生交换格式, 从网表中提取出来的表示 RC 值信息, 在提取工具与时序验证工具之间传递 RC 信息的文件格式。SPEF 提供 RC 信息, 延时计算相对更准确。

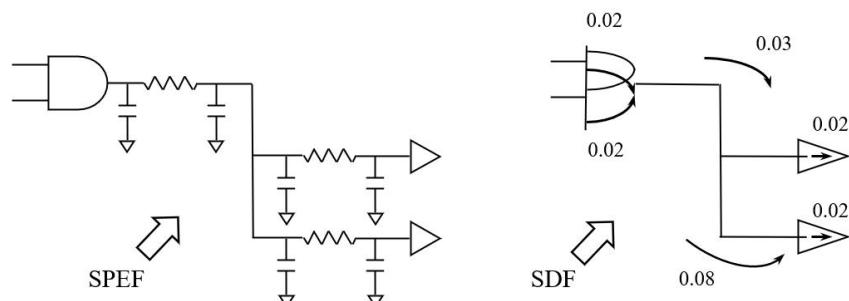


图 20 SDF 与 SPEF 对比

SDF 文件反标包括单元延时和连线延时，寄生 SPEF 反标描述了 RC 参数，
SDF 反标比 SPEF 反标运行速度快。

开源 EDA 流程 iFlow 介绍

一、Build iFlow

系统环境： iFlow 支持在 Ubuntu 20.04 下使用，不推荐使用 20.04 以下的版本。

安装依赖工具及库：

Tools

- * build-essential 12.8
- * cmake 3.16.3
- * clang 10.0
- * bison 3.5.1
- * flex 2.6.4
- * swig 4.0
- * klayout 0.26

Library

- * libeigen3-dev 3.3.7-2
- * libboost-all-dev 1.71.0
- * libffi-dev 3.3-4
- * libreadline-dev 8.0-4
- * libspdlog-dev 1.5.0
- * lemon 1.3.1

首先进入到要存放 iFlow 的目录下，输入命令：

```
“ git clone https://github.com/PCNL-EDA/iFlow.git ” //构建 iFlow 目录结构  
“ cd iFlow ”  
“ ./build_iflow.sh ” //运行脚本下载 EDA 工具  
或者使用代理下载，代理可自定义。例如代理为 hub.fastgit.xyz  
../build_iflow.sh -mirror hub.fastgit.xyz //运行脚本下载 EDA 工具
```

完成后即可使用 iFlow。

二、iFlow 目录结构

1、foundry/

存放工艺库，按不同工艺命名，包括 lef、lib、Verilog(instance 仿真用)和 gds 等库文件。

2、log/

存放 flow 每一步产生的 log，命名格式为 \$design.\$step.\$tools.\$track(eg. HD/uHD).\$corner(eg. MAX/MIN).\$version.log。

3、report/

存放每一步输出的报告，文件夹命名格式类似为 \$design.\$step.\$tools.\$track(eg. HD/uHD).\$corner(eg. MAX/MIN).\$version.

4、result/

存放每一步输出的结果，如 \$design.v, \$design.def, \$design.gds，文件夹命名格式为 \$design.\$step.\$tools.\$track(eg. HD/uHD).\$corner(eg. MAX/MIN).\$version。

5、rtl/

存放 design 的 rtl 文件和 sdc 文件，按不同的 design 命名。

6、scripts/

```
|—cfg           //库文件配置、工具配置、flow 配置脚本存放目录  
|—common        //对文件操作的通用脚本存放目录  
|—run_flow.py  //整个 flow 的运行脚本  
|—$design       //对应 design 每一步的脚本存放目录
```

7、tools/

存放各个工具的文件。

8、work/

存放跑 flow 过程中生成的临时文件，沿用商业工具的习惯。

9、build_iflow.sh

下载安装 EDA 工具。

10、README.md

iFlow 使用说明。

三、 iFlow command

iFlow 可以跑单步流程，也可以同时跑多步流程，同时跑综合和 Floorplan 的命令如下：

Eg. run_flow.py -d aes_cipher_top -s synth,floorplan -f sky130 -t HS -c TYP

命令参数：

-d (design):

design name;

-s (step):

flow 可选 step: synth、v2def、floorplan、tapcell、pdn、gplace、resize、dplace、cts、filler、groute、droute、layout;

-p (previous step):

用于调用前一步的结果；

-f (foundry):

工艺选择，可选：sky130、nangate45、asap7；

-t (track):

标准单元 track 选择，可选：sky130[HS HD]、nangate[HD]、asap7[HS]；

-c (corner):

工艺角，可选：sky130 [TYP]、nangate[TYP]、asap7[MAX TYP MIN]；

-v (version):

追加到 log/result rpt 的版本号；

-l :

前一步的版本号。

step command:

synth: run_flow.py -d \$design -s synth -f \$foundry -t \$track -c \$corner

v2def: run_flow.py -d \$design -s v2def -f \$foundry -t \$track -c \$corner

floorplan: run_flow.py -d \$design -s floorplan -f \$foundry -t \$track -c \$corner

tapcell: run_flow.py -d \$design -s tapcell -f \$foundry -t \$track -c \$corner

```
pdn: run_flow.py -d $design -s pdn -f $foundry -t $track -c $corner
gplace: run_flow.py -d $design -s gplace -f $foundry -t $track -c $corner
resize: run_flow.py -d $design -s resize -f $foundry -t $track -c $corner
dplace: run_flow.py -d $design -s dplace -f $foundry -t $track -c $corner
cts: run_flow.py -d $design -s cts -f $foundry -t $track -c $corner
filler: run_flow.py -d $design -s filler -f $foundry -t $track -c $corner
groute: run_flow.py -d $design -s groute -f $foundry -t $track -c $corner
droute: run_flow.py -d $design -s droute -f $foundry -t $track -c $corner
layout: run_flow.py -d $design -s layout -f $foundry -t $track -c $corner
```

四、iFlow 顶层脚本介绍

1、顶层脚本

iFlow 的顶层脚本为 iFlow/scripts/run_flow.py，可以通过选择不同的参数，包括“design”、“step”、“prestep”、“foundry”、“track”、“corner”、“version”、“preversion”，来运行不同设计、不同步骤或不同工艺等等的流程。进入“iFlow/scripts”目录下，运行命令：

```
./run_flow.py -h
```

即可查看可设置的参数及其参数介绍，如图 1 所示，顶层脚本的参数设置后，通过查找配置脚本中对应的参数进行匹配，再反馈回顶层脚本，从而运行相对应的流程。

```
^0^ ./run_flow.py -h
usage: run_flow.py [-h] --design DESIGN --step STEP [--prestep PRESTEP] [--foundry FOUNDRY] [--track TRACK]
                   [--corner CORNER] [--version VERSION] [--preversion PREVERSION]

For Iflow running

optional arguments:
  -h, --help            show this help message and exit
  --design DESIGN, -d DESIGN
                        Design name
  --step STEP, -s STEP  Flow step, such as synth floorplan pdn gplace resize dplace cts filler groute
                        droute layout
  --prestep PRESTEP, -p PRESTEP
                        Previous step
  --foundry FOUNDRY, -f FOUNDRY
                        Foundry selection, such as nangate45 asap7 smic110 smic55 sky130
  --track TRACK, -t TRACK
                        Standard cell track selection, nangate45 [HD]; asap7 [HS]; smic110 [HD]; smic55 [HD];
                        sky130 [HS HD]
  --corner CORNER, -c CORNER
                        Corner selection, nangate45 [TYP]; asap7 [MAX TYP MIN]; smic110 [MAX MIN]; smic55 [MAX
                        MIN]; sky130 [TYP]
  --version VERSION, -v VERSION
                        Append version name to the end of log/result/rpt
  --preversion PREVERSION, -l PREVERSION
                        Version of prestep
```

图 1 顶层脚本参数

2、配置脚本

iFlow 的配置脚本目录为 “iFlow/scripts/cfg” , 目录下有四个脚本包括 “data_def.py” 、“flow_cfg.py” 、“foundry_cfg.py” 、“tools_cfg.py” , 他们分别控制数据的定义、流程配置、工艺库配置以及工具版本的配置。

(1) data_def.py

这个脚本主要定义了 “Foundry” 、“Tools” 、“Flow” 三个主要参数及其属性，其中，在 “Flow” 参数中定义了流程所含有的步骤，如图 2 所示。

```
class Flow(object):
    all_object = ()

    def __init__(self,design,default_foundry,default_track,default_corner):
        self.design          = design
        self.default_foundry = default_foundry
        self.default_track   = default_track
        self.default_corner  = default_corner
        self.step             = []
        self.tool             = []

        'synth'      : 'yosys_0.9',
        'v2def'      : 'openroad_1.2.0',
        'floorplan'  : 'openroad_1.2.0',
        '#           : 'iFP',
        'tapcell'    : 'openroad_1.2.0',
        'pdn'        : 'openroad_1.2.0',
        'gplace'     : 'openroad_1.2.0',
        'resize'     : 'openroad_1.2.0',
        'dplace'     : 'openroad_1.2.0',
        'cts'        : 'openroad_1.2.0',
        'filler'     : 'openroad_1.2.0',
        '#           : 'tGR',
        'groute'     : 'openroad_1.2.0',
        'droute'     : 'tDR',
        'droute'     : 'openroad_1.2.0',
        'layout'     : 'klayout_0.26.2'

    Flow.all_object += (self,)

Flow.all_object += (self,)
```

图 2 类参数 Flow 的定义

图 2 中绿框中定义了 Flow 中具有哪些步骤，iFlow 中默认的步骤分得比较细，一共有 12 步，用户也可以根据自己的需求进行添加步骤或者合并步骤，并到顶层脚本中做相应的修改。图 2 中蓝框中配置了每一步对应使用的工具及其版本（这里的版本是我们定义的版本号，并非 github 中的版本号，对应的工具可在 “tools_cfg.py” 脚本中配置），iFlow 除了综合 synth 及版图输出 layout 步骤外，其余后端物理设计步骤均使用 OpenRoad 工具实现，其中 v2def 步骤不是必须的，该步骤用于将网表转为 def，OpenROAD 工具也可以直接读入 .v 文件（网表）。

(2) flow_cfg.py

这个脚本中定义了一些 Flow 的默认参数，如图 3 所示，定义了 4 个 Flow 的默认设置，例如，在运行 aes_cipher_top 这个设计的综合时，可以不设置工艺相关的参数，直接运行命令：

```
./run_flow.py -d aes_cipher_top -s synth
```

这时会使用默认的“foundry”、“track”、“corner”运行流程，分别为“sky130”、“HS”、“TYP”。用户可以根据需求进行自定义，或者在运行“run_flow.py”脚本时设定相应的参数。在运行不同设计的流程时，顶层脚本会根据设计名在“iFlow/scripts”目录下查找以设计顶层 module 名一致的目录，读取相应步骤的脚本，如图 4 所示，在“iFlow/scripts/aes_cipher_top”目录下有不同步骤及不同工具版本的 tcl 脚本。

```
#!/usr/bin/python3
#-----
# Copyright 2021 PENG CHENG LABORATORY
#-----
# Author      : liuboju
# Email       : liubj@pcl.ac.cn
# Date        : 2021-04-06
# Project     :
# Language   : Python
# Description :
#-----
import sys
import os
import subprocess
import re
from data_def import *

aes          = Flow('aes_cipher_top','sky130','HS','TYP')
gcd          = Flow('gcd','sky130','HS','TYP')
uart         = Flow('uart','asap7','HS','TYP')
ibex         = Flow('ibex_core','sky130','HS','TYP')
```

图 3 Flow 默认参数的配置

```
zhuangchunan@18:12 ~/zzs/iFlow/scripts/aes_cipher_top
^0^ ls
cts.openroad_0.9.0.tcl    floorplan.openroad_1.2.0.tcl  IP_global.cfg      pdn_sky130.cfg
cts.openroad_1.2.0.tcl    generateParam.tcl           pdn_asap7.cfg      resize.openroad_1.2.0.tcl
dplace.openroad_1.2.0.tcl  gplace.openroad_1.2.0.tcl  pdn.cfg            synth.yosys_0.9.tcl
droute.openroad_1.2.0.tcl  groute.iGR.tcl           pdn.iFP.tcl        tapcell.iFP.tcl
filler.openroad_1.2.0.tcl  groute.openroad_1.2.0.tcl  pdn_nangate45.cfg  tapcell.openroad_1.2.0.tcl
floorplan.iFP.tcl         iFP_script                pdn.openroad_1.2.0.tcl
```

图 4 Flow 设计对应脚本

(3) foundry_cfg.py

这个脚本中定义了不同工艺节点的库文件路径，运行流程时会根据所选择的工艺节点“foundry”参数，到这个脚本里找到对应工艺节点的库文件路径进行读取。如图 5 所示，为 sky130 工艺的库文件配置，包含了“name”、“lib”、“lef”、“gds”四个属性，运行流程时，会根据不同的 track 和 corner 选择读入哪些库文件。sky130 中默认只配置了 TYP 一种 corner，也只有一种 corner，对于其他含有多个 corner 的工艺库，用户可以根据需要，添加其它 corner 的 lib 库，以及添加需要的 sram 的 lib 库、lef 库和 gds 库。

```

#-
# sky130
#-
sky130 = Foundry(
    name='sky130',
    lib = {
        'std,HS,TYP' : (
            '../foundry/sky130/lib/sky130_fd_sc_hs_tt_025C_1v80.lib',
        ),
        'std,HD,TYP' : (
            '../foundry/sky130/lib/sky130_fd_sc_hd_tt_025C_1v80.lib',
        ),
        'dontuse' : 'sky130_fd_sc_hs_xor3_1 *2111* *2211* *3111* *32* *41* *clk* *dly* *nand4* *or4*',
        'macro,TYP' : (
            '../foundry/sky130/lib/sky130_dummy_io.lib',
            '../foundry/sky130/lib/sky130_sram_1rwir_128x256_8_TT_1p8V_25C.lib',
            '../foundry/sky130/lib/sky130_sram_1rwir_44x64_8_TT_1p8V_25C.lib',
            '../foundry/sky130/lib/sky130_sram_1rwir_64x256_8_TT_1p8V_25C.lib',
            '../foundry/sky130/lib/sky130_sram_1rwir_80x64_8_TT_1p8V_25C.lib'
        ),
    },
    lef = {
        'tech' : (
            '../foundry/sky130/lef/sky130_fd_sc_hs.lef',
        ),
        'std,HS' : (
            '../foundry/sky130/lef/sky130_fd_sc_hs_merged.lef',
        ),
        'std,HD' : (
            '../foundry/sky130/lef/sky130_fd_sc_hd_merged.lef',
        ),
        'macro' : (
            '../foundry/sky130/lef/sky130_ef_io_com_bus_slice_10um.lef',
            '../foundry/sky130/lef/sky130_ef_io_com_bus_slice_1um.lef',
            '../foundry/sky130/lef/sky130_ef_io_com_bus_slice_20um.lef',
            '../foundry/sky130/lef/sky130_ef_io_com_bus_slice_5um.lef',
            '../foundry/sky130/lef/sky130_ef_io_connect_vccb_and_vswitch_vddio_slice_20um.lef',
            '../foundry/sky130/lef/sky130_ef_io_corner_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_disconnect_vccd_slice_5um.lef',
            '../foundry/sky130/lef/sky130_ef_io_disconnect_vdda_slice_5um.lef',
            '../foundry/sky130/lef/sky130_ef_io_gpiov2_pad_wrapped.lef',
            '../foundry/sky130/lef/sky130_ef_io_vccd_hvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vccd_lvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vdda_hvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vdda_lvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vddio_hvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vssa_hvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vssa_lvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vssd_hvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vssd_lvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vssi_hvc_pad.lef',
            '../foundry/sky130/lef/sky130_ef_io_vssi_lvc_pad.lef',
            '../foundry/sky130/lef/sky130_fd_io_top_xres4v2.lef',
            '../foundry/sky130/lef/sky130io_fill.lef',
            '../foundry/sky130/lef/sky130_sram_1rwir_128x256_8.lef',
            '../foundry/sky130/lef/sky130_sram_1rwir_44x64_8.lef',
            '../foundry/sky130/lef/sky130_sram_1rwir_64x256_8.lef',
            '../foundry/sky130/lef/sky130_sram_1rwir_80x64_8.lef'
        ),
    },
    gds = {
        'std,HS' : (
            '../foundry/sky130/gds/sky130_fd_sc_hs.gds',
        ),
        'std,HD' : (
            '../foundry/sky130/gds/sky130_fd_sc_hd.gds',
        ),
        'macro' : (
            '../foundry/sky130/gds/sky130_sram_1rwir_128x256_8.gds',
            '../foundry/sky130/gds/sky130_sram_1rwir_44x64_8.gds',
            '../foundry/sky130/gds/sky130_sram_1rwir_64x256_8.gds',
            '../foundry/sky130/gds/sky130_sram_1rwir_80x64_8.gds'
        )
    }
)

```

图 5 工艺库文件路径的配置

(4) tools_cfg.py

这个脚本用于配置每一步使用哪种开源 EDA 工具及其对应的版本号，如图 6 所示，这里配置了三种不同版本的 OpenRoad 工具，OpenROAD 的 1.2.0 版本更新之后相关的命令和 1.1.0 版本有一定的差别，iFlow 默认使用 1.2.0 版本的 OpenROAD，只提供了 1.2.0 版本的脚本，想尝试别的版本可到 OpenROAD 的 git hub 地址 (<https://github.com/The-OpenROAD-Project/OpenROAD>) 去了解相关

命令。方便在“data_def.py”脚本中定义每一步使用的默认工具，也可以用命令指定使用工具的版本，例如：

```
./run_flow.py -d aes_cipher_top -s floorplan=openroad_1.2.0
```

运行此命令指定使用 1.2.0 版本的 OpenRoad 工具进行 floorplan。此外，iFlow 还配置了 iEDA 点工具，也可以尝试使用 iEDA 的工具实现后端物理设计，相应的使用示例我们也会在后续更新。

```
#!/usr/bin/python3
#-----
# Copyright 2021 PENG CHENG LABORATORY
#-----
# Author : liuboju
# Email : liubj@pcl.ac.cn
# Date : 2021-04-06
# Project :
# Language : Python
# Description :
#-----
import sys
import os
import subprocess
import re
from data_def import *

tool1 = Tools(
    ('synth',),
    'yosys_0.9',
    '../..../iFlow/tools/yosys4be891e8/bin/yosys')
tool2 = Tools(
    ('floorplan', 'tapcell', 'pdn', 'gplace', 'resize', 'dplace', 'cts', 'filler', 'groute'),
    'openroad_1.1.0',
    '../..../iFlow/tools/OpenROAD9295a533/build/src/openroad')
tool3 = Tools(
    ('v2def', 'floorplan', 'tapcell', 'pdn', 'gplace', 'resize', 'dplace', 'cts', 'filler', 'groute', 'droute'),
    'openroad_1.2.0',
    '../..../iFlow/tools/OpenROADae191807/build/src/openroad')
tool4 = Tools(
    ('droute',),
    'TritonRoute_1.0',
    '../..../iFlow/tools/TritonRoute758cdac/build/TritonRoute')
tool5 = Tools(
    ('layout',),
    'klayout_0.26.2',
    '/usr/bin/klayout')
tool6 = Tools(
    ('floorplan', 'tapcell', 'pdn', 'gplace', 'resize', 'dplace', 'cts', 'filler', 'groute'),
    'openroad_0.9.0',
    '../..../iFlow/tools/OpenROAD_fixcts/openroad')
tool7 = Tools(
    ('groute',),
    'iGR',
    '../..../iFlow/tools/iGR/run_gr')
tool8 = Tools(
    ('droute',),
    'iDR',
    '../..../iFlow/tools/iDR/run_dr')
tool9 = Tools(
    ('floorplan', 'tapcell', 'pdn'),
    'iFP',
    '../..../iFlow/tools/iEDA/iFP/run_iFP')
```

图 6 工艺库文件路径的配置

五、iFlow 流程介绍

3、综合

iFlow 使用的综合工具是 yosys，版本号为 4be891e8。综合的目的是将 RTL 代码转化为网表，在 iFlow 中，RTL 代码放在“iFlow/rtl”中，RTL 代码的目录

用顶层 module 名称来命名。在运行综合流程之前，首先要确认综合脚本中的配置是否正确。以 gcd 设计为例，进入“iFlow/scripts/gcd”目录，打开综合流程相关的 tcl 脚本，用户需要重点关注的部分参数配置在脚本的前面，如图 7 所示。

```
#=====
# set tool related parameter
#=====
set MERGED_LIB_FILE      "$PROJ_PATH/foundry/$FOUNDRY/lib/merged.lib"
set BLACKBOX_V_FILE      "$PROJ_PATH/foundry/$FOUNDRY/verilog/blackbox.v"
#set VERILOG_TOP_PARAMS   ""
set CLKGATE_MAP_FILE     "$PROJ_PATH/foundry/$FOUNDRY/verilog/cells_clkgate.v"
set LATCH_MAP_FILE       "$PROJ_PATH/foundry/$FOUNDRY/verilog/cells_latch.v"
set BLACKBOX_MAP_TCL     "$PROJ_PATH/foundry/$FOUNDRY/blackbox_map.tcl"
set CLOCK_PERIOD          "20.0"

if { $FOUNDRY == "sky130" } {
    if { $TRACK == "HS" } {
        set TIEHI_CELL_AND_PORT      "sky130_fd_sc_hs_conb_1 HI"
        set TIELO_CELL_AND_PORT      "sky130_fd_sc_hs_conb_1 LO"
        set MIN_BUF_CELL_AND_PORTS  "sky130_fd_sc_hs_buf_1 A X"
    } elseif { $TRACK == "HD" } {
        set TIEHI_CELL_AND_PORT      "sky130_fd_sc_hd_conb_1 HI"
        set TIELO_CELL_AND_PORT      "sky130_fd_sc_hd_conb_1 LO"
        set MIN_BUF_CELL_AND_PORTS  "sky130_fd_sc_hd_buf_1 A X"
    }
} elseif { $FOUNDRY == "nangate45" } {
    set TIEHI_CELL_AND_PORT      "LOGIC1_X1 Z"
    set TIELO_CELL_AND_PORT      "LOGIC0_X1 Z"
    set MIN_BUF_CELL_AND_PORTS  "BUF_X1 A Z"
} elseif { $FOUNDRY == "asap7" } {
    set TIEHI_CELL_AND_PORT      "TIEHIx1_ASAP7_75t_R H"
    set TIELO_CELL_AND_PORT      "TIELOx1_ASAP7_75t_R L"
    set MIN_BUF_CELL_AND_PORTS  "BUFx2_ASAP7_75t_R A Y"
}

set VERILOG_INCLUDE_DIRS  \
"
set VERILOG_FILES  \
$RTL_PATH/gcd.v \
"
```

图 7 综合相关配置

首先，要配置好综合需要读入的库文件，例如 blackbox 的 verilog 文件和 map 文件等等，然后，还需要对一些综合时要用到的特定的 cell 也要在这里进行配置，包括 tie cell 和 buffer。最后，还需要配置 RTL 代码所在的路径。

```
# generic synthesis
synth -top $DESIGN

# Optimize the design
opt -purge

# technology mapping of latches
if {[info exist LATCH_MAP_FILE]} {
    techmap -map $LATCH_MAP_FILE
}

# technology mapping of flip-flops
dfflibmap -liberty $MERGED_LIB_FILE
opt -undriven

# Technology mapping for cells
abc -D [expr $CLOCK_PERIOD * 1000] \
    -constr "$SDC_FILE" \
    -liberty $MERGED_LIB_FILE \
    -script $abc_script \
    -showtmp
```

图 8 综合主要命令

综合相关的命令如图 8 所示，包括综合、优化以及 mapping 三个主要步骤，其中 abc 在优化时需要读入时序约束 sdc 文件，这个文件需要放在“iFlow/rtl”目录中对应的设计目录下，用于综合时进行时序优化。跑单步综合命令如下：

```
./run_flow.py -d gcd -s synth
```

4、布局

iFlow 中布局包括六个小步骤，分别为 floorplan、tapcell、PDN、gplace、resize、dplace，在默认情况下，必须按照上述步骤的顺序进行流程，用户也可以根据需求通过修改顶层脚本的“-s”和“-p”参数来修改当前步骤及前一步骤。在布局规划中，目的是为了规划芯片的面积及形状，并将综合后输出的网表中所包含的 instance 摆放到芯片上。接下来将一一讲述布局中的各个步骤：

(1) floorplan

在 floorplan 这一步中，主要是进行芯片的面积以及形状的规划，配置参数“DIE_AREA”和“CORE_AREA”，如图 9 所示。

```
#=====
#   set tool related parameter
#=====

#set DIE_AREA          "0 0 1120 1020.8"
#set CORE_AREA         "10 12 1110 1011.2"
set DIE_AREA          "0 0 220.2 220.2"
set CORE_AREA         "1.08 1.08 219.12 219.12"

set TRACKS_INFO_FILE      "$PROJ_PATH/foundry/$FOUNDRY/tracks_1.2.0.info"

if { $FOUNDRY == "sky130" } {
    set PLACE_SITE    "unit"
    set IO_H_LAYER   "met3"
    set IO_V_LAYER   "met2"
} elseif { $FOUNDRY == "nangate45" } {
    set PLACE_SITE    "FreePDK45_38x28_10R_NP_162NW_340"
    set IO_H_LAYER   "metal3"
    set IO_V_LAYER   "metal2"
} elseif { $FOUNDRY == "asap7" } {
    set PLACE_SITE    "asap7sc7p5t"
    set IO_H_LAYER   "M4"
    set IO_V_LAYER   "M5"
}
```

图 9 floorplan 脚本配置

在 floorplan 阶段，会根据工艺相关的 techfile 文件生成 Row 和 Site，这里选择的 Site 类型为“unit”，如图 10 所示。此外，floorplan 阶段还会生成用于走线的 track，因此还需要在“iFlow/foundry/\$FOUNDRY”目录下配置 track 对应的参

数，如图 11 所示，sky130 工艺一共有 6 层金属，这里对它们的走线 track 进行了定义。

```
# Standard density, single height
SITE unit
    SYMMETRY Y ;
    CLASS CORE ;
    SIZE 0.48 BY 3.33 ;
END unit

# Standard density, double height
SITE unitdbl
    SYMMETRY Y ;
    CLASS CORE ;
    SIZE 0.48 BY 6.66 ;
END unitdbl
```

图 10 techfile 中 Site 的选择

```
make_tracks li1 -x_offset 0.24 -x_pitch 0.48 -y_offset 0.185 -y_pitch 0.37
make_tracks met1 -x_offset 0.185 -x_pitch 0.37 -y_offset 0.185 -y_pitch 0.37
make_tracks met2 -x_offset 0.24 -x_pitch 0.48 -y_offset 0.24 -y_pitch 0.48
make_tracks met3 -x_offset 0.37 -x_pitch 0.74 -y_offset 0.37 -y_pitch 0.74
make_tracks met4 -x_offset 0.48 -x_pitch 0.96 -y_offset 0.48 -y_pitch 0.96
make_tracks met5 -x_offset 1.85 -x_pitch 3.33 -y_offset 1.85 -y_pitch 3.33
```

图 11 track 的参数配置

完成了参数的配置后，需要进行 floorplan 的初始化，生成相应的 Die、Core 及 Row 等等，OpenRoad 的 floorplan 初始化有三种，可以根据设置好的策略进行初始化，可以根据设定的利用率进行初始化，还可以根据设定的 “DIE_AREA” 和 “CORE_AREA” 进行初始化，iFlow 的 floorplan 脚本默认情况下，采用第三种，如图 12 红框中所示。

```
# Initialize floorplan using ICeWall FOOTPRINT
# -----
if {[info exists strategy_file]} {
    ICeWall load_footprint $strategy_file
    initialize_floorplan \
        -die_area [ICeWall get_die_area] \
        -core_area [ICeWall get_core_area] \
        -site $PLACE_SITE
    source $TRACKS_INFO_FILE
    ICeWall init_footprint $sigmap_file
}

# Initialize floorplan using CORE_UTILIZATION
# -----
} elseif {[info exists CORE_UTILIZATION] && $CORE_UTILIZATION != "" } {
    initialize_floorplan -utilization $CORE_UTILIZATION \
        -aspect_ratio $CORE_ASPECT_RATIO \
        -core_space $CORE_MARGIN \
        -site $PLACE_SITE
    source $TRACKS_INFO_FILE
}

# Initialize floorplan using DIE_AREA/CORE_AREA
# -----
} else {
    initialize_floorplan -die_area $DIE_AREA \
        -core_area $CORE_AREA \
        -site $PLACE_SITE
    source $TRACKS_INFO_FILE
}
```

图 12 floorplan 初始化

跑单步 floorplan 命令如下：

```
./run_flow.py -d gcd -s floorplan -p synth
```

(2) tapcell

在 floorplan 初始化之后，需要在 core area 范围内插入 tapcell，tapcell 的作用是为所有标准单元的 N 阵和衬底提供偏置电源，在 core area 范围内每间隔一段距离则需要摆放一个 tapcell，在 tapcell 这一步还需要插入 endcap，主要是为了插在边界处或 sram 及 ip 周围消除不对称性，在脚本中对应的配置如图 13 所示，这里需要配置摆放 tapcell 的间距，以及 tapcell 和 endcap 选用的标准单元的类型，这些参数可以在脚本前面设置，如图 14 所示。

```
tapcell \
    -distance $DISTANCE \
    -tapcell_master $TAPCELL_MASTER \
    -endcap_master $ENDCAP_MASTER
```

图 13 tapcell 脚本配置

```
=====
#   set tool related parameter
=====
if { $FOUNDRY == "sky130" } {
    set DISTANCE 14
    if { $TRACK == "HS" } {
        set TAPCELL_MASTER "sky130_fd_sc_hs_tap_1"
        set ENDCAP_MASTER "sky130_fd_sc_hs_fill_1"
    } elseif { $TRACK == "HD" } {
        set TAPCELL_MASTER "sky130_fd_sc_hd_tap_1"
        set ENDCAP_MASTER "sky130_fd_sc_hd_fill_1"
    }
} elseif { $FOUNDRY == "nangate45" } {
    set DISTANCE 120
    set TAPCELL_MASTER "TAPCELL_X1"
    set ENDCAP_MASTER "TAPCELL_X1"
} elseif { $FOUNDRY == "asap7" } {
    set DISTANCE 25
    set TAPCELL_MASTER "TAPCELL_ASAP7_75t_R"
    set ENDCAP_MASTER "TAPCELL_ASAP7_75t_R"
}
```

图 14 tapcell 相关参数

跑单步 tapcell 命令如下：

```
./run_flow.py -d gcd -s tapcell -p floorplan
```

(3) PDN

在布局中，除了面积规划及标准单元的摆放之外，还有相当重要的一步为 power plan，又称为 PDN，这一步主要是构建为整个芯片供电的电源网络，一个

芯片的电源网络质量直接影响整个芯片的性能。PDN 这一步的脚本比较简单，只有一条简单的命令，如图 15 所示，与电源网络相关的配置在“pdn_\$FOUNDRY.cfg”配置文件中，对于使用不同的工艺库，电源网络的构建不同。

```
#=====  
#   set tool related parameter  
#=====  
set PDN_CFG_FILE      "$PROJ_PATH/scripts/$DESIGN/pdn_$FOUNDRY.cfg"  
#=====  
#   main running  
#=====  
# Read lef  
foreach lef $LEF_FILES {  
    read_lef $lef  
}  
  
# Read liberty files  
foreach libFile $LIB_FILES {  
    read_liberty $libFile  
}  
  
# Read def files  
read_def $PRE_RESULT_PATH/$DESIGN.def  
  
pdngen $PDN_CFG_FILE -verbose
```

图 15 pdn 脚本命令及配置

Sky130 工艺的配置文件中具体的内容如图 16 所示。在“pdn_sky130.cfg”配置文件中，首先要创建电源相关的 net，包括“VDD”和“VSS”，如图 16 中红框所示，然后需要把与电源相关的 pin 从逻辑上连接到“VDD”和“VSS”两个 net 上，如图 16 中绿框所示，最后是构建电源网络的 power stripe，在这个工艺下，构建了用于标准单元供电的 met1 power rail 和 met4、met5 的 power stripe，如图 16 中橙框所示。此外，在含有 macro 的设计中，我们还需要将 macro 中的电源连接到芯片的电源网络上，从而为 macro 供电。

```

# POWER or GROUND #Std. cell rails starting with power or ground rails at the bottom of the core area
set ::rails_start_with "POWER" ;
# POWER or GROUND #Upper metal stripes starting with power or ground rails at the left/bottom of the core area
set ::stripes_start_with "POWER" ;
# Power nets
set ::power_nets "VDD"
set ::ground_nets "VSS"

set pdngen::global_connections {
    VDD {
        {inst_name .* pin_name VPWR}
        {inst_name .* pin_name VPB}
        {inst_name .* pin_name vdd}
    }
    VSS {
        {inst_name .* pin_name VGND}
        {inst_name .* pin_name VNB}
        {inst_name .* pin_name gnd}
    }
}
##=> Power grid strategy
# Ensure pitches and offsets will make the stripes fall on track
pdngen::specify_grid stdcell {
    name grid
    rails {
        met1 {width 0.48 offset 0}
    }
    straps {
        met4 {width 1.600 pitch 27.140 offset 13.570}
        met5 {width 1.600 pitch 27.200 offset 13.600}
    }
    connect {{met1 met4} {met4 met5}}
}

pdngen::specify_grid macro {
    orient {R0 R180 MX MY}
    power_pins "vdd"
    ground_pins "gnd"
    blockages "li1 met1 met2 met3 "
    connect {{met3_PIN_ver met5}}
}

```

图 16 电源网络相关配置

跑单步 pdn 命令如下：

```
./run_flow.py -d gcd -s pdn -p tapcell
```

(4) gplace

在完成电源网络的构建后，接下来需要将标准单元摆放到 core area 范围中，这一步即为 gplace，又称为 global place。在 gplace 阶段，需要配置的主要参数有两个，如图 17 所示，一个为线 RC 参数的抽取层，主要是为了在 gplace 阶段抽取线 RC 参数进行延时的评估，从而更好地优化标准单元的摆放位置；另一个为“PLACE_DENSITY”，这一参数是用于设置摆放标准单元时的密度，即标准单元摆放的紧密程度。

```

=====
#   set tool related parameter
=====
if { $FOUNDRY == "sky130" } {
    set WIRE_RC_LAYER "met3"
} elseif { $FOUNDRY == "nangate45" } {
    set WIRE_RC_LAYER "metal3"
} elseif { $FOUNDRY == "asap7" } {
    set WIRE_RC_LAYER "M3"
}

set PLACE_DENSITY    "0.3"

```

图 17 gplace 脚本配置

运行 gplace 的命令如图 18 所示，overflow 参数默认为 0.1，用户也可以自行定义。在 gplace 阶段，是不会去修复所有的单元重叠，标准单元的合法化需要到 dplace 阶段才会实现，在 gplace 阶段，会不断对标准单元的位置进行优化迭代，直到 overflow 达到所设定的值，如图 19 所示，经过 420 次迭代后满足设定的 overflow 值 0.01。

```
global_placement -density $PLACE_DENSITY  
#global_placement -incremental -overflow 0.05 -density $PLACE_DENSITY
```

图 18 gplace 运行命令

```
[NesterovSolve] Iter: 280 overflow: 0.662674 HPWL: 664557588  
[NesterovSolve] Iter: 290 overflow: 0.635004 HPWL: 687324316  
[NesterovSolve] Iter: 300 overflow: 0.60319 HPWL: 708135616  
[NesterovSolve] Iter: 310 overflow: 0.568292 HPWL: 739411362  
[NesterovSolve] Iter: 320 overflow: 0.53149 HPWL: 768832909  
[NesterovSolve] Iter: 330 overflow: 0.49191 HPWL: 797001605  
[NesterovSolve] Iter: 340 overflow: 0.445586 HPWL: 828156689  
[NesterovSolve] Iter: 350 overflow: 0.392748 HPWL: 860864247  
[NesterovSolve] Iter: 360 overflow: 0.325198 HPWL: 899922378  
[NesterovSolve] Iter: 370 overflow: 0.268116 HPWL: 932820005  
[NesterovSolve] Iter: 380 overflow: 0.217422 HPWL: 962285533  
[NesterovSolve] Iter: 390 overflow: 0.180499 HPWL: 984585691  
[NesterovSolve] Iter: 400 overflow: 0.147763 HPWL: 1001375600  
[NesterovSolve] Iter: 410 overflow: 0.123375 HPWL: 1013666525  
[NesterovSolve] Iter: 420 overflow: 0.103283 HPWL: 1022425892  
[NesterovSolve] Finished with Overflow: 0.099596
```

图 19 gplace 的 overflow 参数

跑单步 gplace 命令如下：

```
./run_flow.py -d gcd -s gplace -p pdn
```

(5) resize

resize 这一步骤主要是在 dplace 前，进行一部分标准单元的更换及插入，其中包括将逻辑 0 和逻辑 1 的驱动端加上 Tie cell 和在需要 fix fanout 的驱动端加上 buffer。resize 阶段需要配置的参数主要有“MAX_FANOUT”以及 fix fanout 时需要用到的 Tie cell 和 buffer 类型，如图 20 所示。

```

#=====
#   set tool related parameter
#=====
set MAX_FANOUT      "30"
if { $FOUNDRY == "sky130" } {
    set WIRE_RC_LAYER      "met3"
    if { $TRACK == "HS" } {
        set TIEHI_CELL_AND_PORT "sky130_fd_sc_hs_conb_1 HI"
        set TIELO_CELL_AND_PORT "sky130_fd_sc_hs_conb_1 LO"
        set FIX_DRC_BUF         "sky130_fd_sc_hs_buf_8"
    } elseif { $TRACK == "HD" } {
        set TIEHI_CELL_AND_PORT "sky130_fd_sc_hd_conb_1 HI"
        set TIELO_CELL_AND_PORT "sky130_fd_sc_hd_conb_1 LO"
        set FIX_DRC_BUF         "sky130_fd_sc_hd_buf_8"
    }
} elseif { $FOUNDRY == "nangate45" } {
    set WIRE_RC_LAYER      "metal3"
    set TIEHI_CELL_AND_PORT "LOGIC1_X1 Z"
    set TIELO_CELL_AND_PORT "LOGIC0_X1 Z"
    set FIX_DRC_BUF         "BUF_X4"
} elseif { $FOUNDRY == "asap7" } {
    set WIRE_RC_LAYER      "M3"
    set TIEHI_CELL_AND_PORT "TIEHIx1_ASAP7_75t_R H"
    set TIELO_CELL_AND_PORT "TIELOx1_ASAP7_75t_R L"
    set FIX_DRC_BUF         "BUFx4_ASAP7_75t_R"
}

```

图 20 resize 脚本的参数配置

在 iFlow 的 resize 流程中，主要是进行 fanout 的修复，降低 fanout 以增加各级的驱动能力，具体的命令如图 21 所示。此外，还可以通过命令指定修复 cap 和 slew 所用的 buffer 类型，分别为“repair_max_cap -buffer_cell \$buffer_cell”、 “repair_max_slew -buffer_cell \$buffer_cell”。

```

#####
# Repair max slew/cap/fanout violations and normalize slews
puts "Repair max slew/cap/fanout violations and normalize slews..."
estimate_parasitics -placement

repair_design

# Repair tie lo fanout
puts "Repair tie lo fanout..."
puts "*****"
set tieolo_cell_name [lindex $TIELO_CELL_AND_PORT 0]
puts "*****"
set tieolo_lib_name [get_name [get_property [get_lib_cell */$tieolo_cell_name] library]]
set tieolo_pin $tieolo_lib_name/$tieolo_cell_name/[lindex $TIELO_CELL_AND_PORT 1]
repair_tie_fanout -max_fanout 1 $tieolo_pin

# Repair tie hi fanout
puts "Repair tie hi fanout..."
set tiehi_cell_name [lindex $TIEHI_CELL_AND_PORT 0]
set tiehi_lib_name [get_name [get_property [get_lib_cell */$tiehi_cell_name] library]]
set tiehi_pin $tiehi_lib_name/$tiehi_cell_name/[lindex $TIEHI_CELL_AND_PORT 1]
repair_tie_fanout -max_fanout 1 $tiehi_pin

# Repair max fanout
puts "Repair max fanout..."
repair_max_fanout -max_fanout $MAX_FANOUT -buffer_cell $buffer_cell

```

图 21 resize 的主要实现

跑单步 resize 命令如下：

./run_flow.py -d gcd -s resize -p gplace

(6) dplace

iFlow 中 dplace 的主要作用是对 gplace 阶段已经摆放的标准单元进行合法化，消除标准单元之间的重叠，将标准单元对齐到 core area 范围内的 Row 上，从而确保电源网络能为标准单元供电，又称为 detail place。dplace 流程的主要命令为“detailed_placement”，dplace 这一步只是将标准单元位置进行合法化，因此不需要设置参数。

跑单步 dplace 命令如下：

```
./run_flow.py -d gcd -s dplace -p resize
```

5、CTS

CTS 的全称为 Clock Tree Synthesis，时钟树综合，这是后端物理设计的一个关键步骤，EDA 工具会根据时序约束文件，创建真实的时钟，并构建时钟树，目的是通过插入 buffer 或 inverter 的方法使得同一时钟域到各个寄存器时钟端的延迟尽可能保持一致，即时钟 skew 尽可能小。进行 CTS 流程前，需要设置用于构建时钟树的 buffer 的 cell 类型，如图 22 所示。

```
#=====
#   set tool related parameter
#=====
if { $FOUNDRY == "sky130" } {
    if { $TRACK == "HS" } {
        set ROOT_BUF      "sky130_fd_sc_hs_buf_1"
        set BUF_LIST      "sky130_fd_sc_hs_buf_1"
    } elseif { $TRACK == "HD" } {
        set ROOT_BUF      "sky130_fd_sc_hd_buf_1"
        set BUF_LIST      "sky130_fd_sc_hd_buf_1"
    }
    set WIRE_RC_LAYER "met3"
} elseif { $FOUNDRY == "nangate45" } {
    set ROOT_BUF      "CLKBUF_X2"
    set BUF_LIST      "CLKBUF_X2"
    set WIRE_RC_LAYER "metal3"
} elseif { $FOUNDRY == "asap7" } {
    set ROOT_BUF      "BUFx4_ASAP7_75t_R"
    set BUF_LIST      "BUFx4_ASAP7_75t_R"
    set WIRE_RC_LAYER "M3"
}
```

图 22 CTS buffer 设置

与 CTS 相关的主要命令如图 23 所示，构建时钟树之前需要先对原有的 buffer 和 inverter 进行 resize 操作，即通过更换 buffer 和 inverter 的尺寸已增强驱动能力，在时钟树综合时再根据所需插入 buffer 和 inverter，时钟树构建之后，这时已有实际的时钟，使用命令“repair_clock_nets”修 cap，slew 和 skew。此外，还需要重新进行一次 dplace，因为 CTS 时会插入 buffer 和 inverter，需要再次 dplace 来保证标准单元位置摆放的合法化。

```

set_wire_rc -layer $WIRE_RC_LAYER
repair_clock_inverters
clock_tree_synthesis \
    -buf_list $BUF_LIST \
    -root_buf $ROOT_BUF \
    -sink_clustering_enable \
    -out_path $RESULT_PATH/
repair_clock_nets
set_placement_padding -global -left 3 -right 3
detailed_placement
check_placement

```

图 23 CTS 相关命令

跑单步 CTS 命令如下：

```
./run_flow.py -d gcd -s cts -p dplace
```

6、filler

在构建时钟树，并完成 timing 的修复之后，所有的标准单元已经确认并固定，后续的操作不会改变网表，这时，我们需要在整个 core area 范围内填满 filler cell，主要作用是为了填充标准单元之间的空隙，将整个扩散层连接起来，以满足 DRC (Design Rule Check) 要求，以构成 power rail，使电源和地线保持连接。这一步骤也可以再布线之后进行，在 iFlow 中，默认是在 CTS 之后，布线之前进行 filler cell 的插入，用户可以根据需求进行调整。在进行 filler insert 操作前，需要设置 filler cell 的类型，如图 24 所示，这里一共使用了多种不同大小的 filler cell，最小的 filler cell 宽度与一个 Site 宽度一致。然后，使用命令“filler_placement \$FILL_CELLS”即可填充 filler。

```

=====
# set tool related parameter
=====
set FILL_CELLS "sky130_fd_sc_hs_fill_1 sky130_fd_sc_hs_fill_2 sky130_fd_sc_hs_fill_4 sky130_fd_sc_hs_fill_8 "
if { $FOUNDRY == "sky130" } {
    if { $TRACK == "HS" } {
        set FILL_CELLS "sky130_fd_sc_hs_fill_1 sky130_fd_sc_hs_fill_2 sky130_fd_sc_hs_fill_4 sky130_fd_sc_hs_fill_8 "
    } elseif { $TRACK == "HD" } {
        set FILL_CELLS "sky130_fd_sc_hd_fill_1 sky130_fd_sc_hd_fill_2 sky130_fd_sc_hd_fill_4 sky130_fd_sc_hd_fill_8 "
    }
} elseif { $FOUNDRY == "nangate45" } {
    set FILL_CELLS "FILLCELL_X1 FILLCELL_X2 FILLCELL_X4 FILLCELL_X8 FILLCELL_X16 FILLCELL_X32 "
} elseif { $FOUNDRY == "asap7" } {
    set FILL_CELLS "FILLERxp5_ASAP7_75t_R"
}

```

图 24 CTS 相关命令

跑单步 filler 命令如下：

```
./run_flow.py -d gcd -s filler -p cts
```

7、布线

在 iFlow 中，布线一共分为两步流程，分别是 groute 和 droute，groute 生成一个引导布线文件 guide，droute 读入 guide 完成实际的布线。

(1) groute

groute 又称为 global route，这一步骤会做好布线资源分配，生成布线引导文件“route.guide”。groute 主要设置的参数为各金属层在库里面的对应名字，用于确定布线所用层，对于不同的工艺有不同数量的金属层，每层金属的名称也不一样，sky130 工艺一共有六层金属层，如图 25 所示，同时，还需定义用于信号线网和时钟线网的走线层。

```
#=====
#   set tool related parameter
#=====
if { $FOUNDRY == "sky130" } {
    #   set MAX_ROUTING_LAYER    "6"
    set WIRE_RC_LAYER         "met3"
    set LAYER_M2               met2
    set LAYER_M3               met3
    set LAYER_M4               met4
    set LAYER_M5               met5
    set LAYER_M6               met6
    set LAYER_M7               met7
    set SIGNAL_LAYER           "met2-met5"
    set CLOCK_LAYER            "met2-met3"
} elseif { $FOUNDRY == "nangate45" } {
    #   set MAX_ROUTING_LAYER    "6"
    set WIRE_RC_LAYER         "metal3"
    set LAYER_M2               metal2
    set LAYER_M3               metal3
    set LAYER_M4               metal4
    set LAYER_M5               metal5
    set LAYER_M6               metal6
    set LAYER_M7               metal7
    set LAYER_M8               metal8
    set LAYER_M9               metal9
    set LAYER_M10              metal10
    set SIGNAL_LAYER           "metal2-metal8"
    set CLOCK_LAYER            "metal2-metal5"
} elseif { $FOUNDRY == "asap7" } {
    set MAX_ROUTING_LAYER    "6"
    set WIRE_RC_LAYER         "M3"
    set LAYER_M2               M2
    set LAYER_M3               M3
    set LAYER_M4               M4
    set LAYER_M5               M5
    set LAYER_M6               M6
    set LAYER_M7               M7
    set LAYER_M8               M8
    set LAYER_M9               M9
    set SIGNAL_LAYER           "M2-M8"
    set CLOCK_LAYER            "M2-M5"
}
```

图 25 groute 布线层设置

在开始 groute 前，如果设计中有用到 sram 等 marco，还需要给 marco 加上 routing blockage，避免在布线时外部的走线与 marco 内部走线重叠而产生短路，如图 26 所示。加上 routing blockage 后，便可以开始 groute，命令如图 27 所示，

这里设置 200 次迭代，groute 阶段必须保证完全消除 overflow，从而避免存在短路现象，否则会使芯片功能错误。

```

foreach inst $allInsts {
    set master [$inst getMaster]
    set name [$master getName]
    set loc_llx [lindex [$inst getLocation] 0]
    set loc_lly [lindex [$inst getLocation] 1]

    #[string match "S013PLL*" $name]|||
    if {[string match "sky130_sram_1rw1r*" $name]} {
        puts "create route block around $name"
        set w [$master getWidth]
        set h [$master getHeight]
        puts "$name loc : $loc_llx $loc_lly"

        set llx_Mx [expr $loc_llx - $distance]
        set lly_Mx [expr $loc_lly - $distance]
        set urx_Mx [expr $loc_llx + $w + $distance]
        set ury_Mx [expr $loc_lly + $h + $distance]

        set obs_M2 [odb::dbObstruction_create $block $layer_M2 $llx_Mx $lly_Mx $urx_Mx $ury_Mx]
        set obs_M3 [odb::dbObstruction_create $block $layer_M3 $llx_Mx $lly_Mx $urx_Mx $ury_Mx]
        set obs_M4 [odb::dbObstruction_create $block $layer_M4 $llx_Mx $lly_Mx $urx_Mx $ury_Mx]
        set obs_M5 [odb::dbObstruction_create $block $layer_M5 $llx_Mx $lly_Mx $urx_Mx $ury_Mx]
        set obs_M6 [odb::dbObstruction_create $block $layer_M6 $llx_Mx $lly_Mx $urx_Mx $ury_Mx]
        set obs_M7 [odb::dbObstruction_create $block $layer_M7 $llx_Mx $lly_Mx $urx_Mx $ury_Mx]

        incr cnt
    }
}

```

图 26 给 marco 加上 routing blockage

```

global_route -guide_file $RESULT_PATH/route.guide \
             -overflow_iterations 200 \
             -verbose 2 \

```

图 27 groute 相关命令

跑单步 groute 命令如下：

```
./run_flow.py -d gcd -s groute -p filler
```

(2) droute

droute 流程是将 groute 输出的 route.guide 文件读入，并根据 guide 文件的描述去形成实际布线的过程，又称为 detail place。这一步骤的实施主要是依赖 groute 输出的 route.guide 文件，因此，没有参数需要设置，读入相应的 lef 物理库和 guide 文件之后，便可以开始 droute。droute 步骤的命令在顶层脚本“run_flow.py”中实现，如图 28 所示。

```

# step run
if re.search('TritonRoute', Flow.get_tool(args.design)[astep]) :
    os.system(Tools.get_path(cur_tool_name) +
              '-lef '+proj_path+'/foundry/'+foundry_sel+'/'+lef/merged_spacing.lef' +
              '-def '+os.environ['IFLOW_PRE_RESULT_PATH']+args.design+'.def' +
              '-guide '+os.environ['IFLOW_PRE_RESULT_PATH']+route.guide' +
              '-output '+os.environ['IFLOW_RESULT_PATH']+args.design+'.def' +
              '-threads 8' +
              '-verbose 1'+ tee -ia '+log_file)

```

图 28 droute 相关命令

跑单步 droute 命令如下：

```
./run_flow.py -d gcd -s droute -p groute
```

8、版图

droute 完成后输出的是 def 文件，而不是 gds 文件，需要得到用于 foundry 生产的 gds 文件还需要一个 merge 的流程，在 iFlow 中，这一流程命名为“layout”。 droute 得到的 def 文件是一个基于金属层层面的描述文件，其中的标准单元、IO cell 以及 marco 等等都是一个黑盒子，只描述了其形状，没有具体的 layer 层描述，merge 流程是将这些黑盒子的 gds 和 def 文件进行合并，从而得到最终的 gds 文件的过程。

merge 过程的具体命令在顶层脚本“run_flow.py”中实现，如图 29 所示， merge 需要读入的文件包括 droute 输出的 def，还有标准单元、IO cell、marco 的 gds 文件，以及工艺的 layer map 文件 klayout.lyt 和 klayout.lyp，最终输出 gds 版图。

```
elif re.search('layout', Flow.get_tool(args.design)[astep]) :
    os.system(proj_path+'/scripts/common/klayoutInsertLef.py '+
              '-i '+proj_path+'/foundry/'+foundry_sel+'/klayout.lyt'+
              '-l "'+os.environ['IFLOW_LEF_FILES']+''+
              '-o './klayout.lyt')
    os.system(Tools.get_path(cur_tool_name)+' -zz'
              '-rd design_name='+args.design+
              '-rd in_def='+os.environ['IFLOW_PRE_RESULT_PATH']+ '/' + args.design+'.def'+
              '-rd in_gds="'+os.environ['IFLOW_GDS_FILES']+''+
              '-rd out_gds='+os.environ['IFLOW_RESULT_PATH']+ '/' + args.design+'.gds'+
              '-rd tech_file='./klayout.lyt'+
              '-rd config_file='+'''+
              '-rd seal_gds='+'''+
              '-rm '+proj_path+'/scripts/common/def2gds.py'+ ' | tee -ia '+log_file)
    os.system(Tools.get_path(cur_tool_name)+' '+
              '-l '+proj_path+'/foundry/'+foundry_sel+'/klayout.lyp'+
              '+os.environ['IFLOW_RESULT_PATH']+ '/' + args.design+'.gds &')
```

图 29 droute 相关命令

跑单步 layout (merge) 命令如下：

```
./run_flow.py -d gcd -s layout -p droute
```

得到 gds 版图后，可以使用 klayout 工具查看版图。此外，klayout 工具还可以查看 def 文件，打开 klayout 的 GUI 界面后，在菜单“File/Import”的子菜单中选择 DEF/LEF 导入 def 文件以及 lef 文件。如图 30 所示，在弹窗的“Import File”中选择 detail route 生成的 def 文件导入，在“With LEF files:”中添加 design 中标准单元、IO cell 及 marco 的 lef 文件，在“iFlow/foundry/sky130/lef”中可以找到，添加完毕后点“OK”即可导入。产生的结果如图 31 所示，这可以在 merge 前帮助我们检验布线结果的质量，也可以检查前面每一步之后的结果，包括

floorplan、filler 等等。

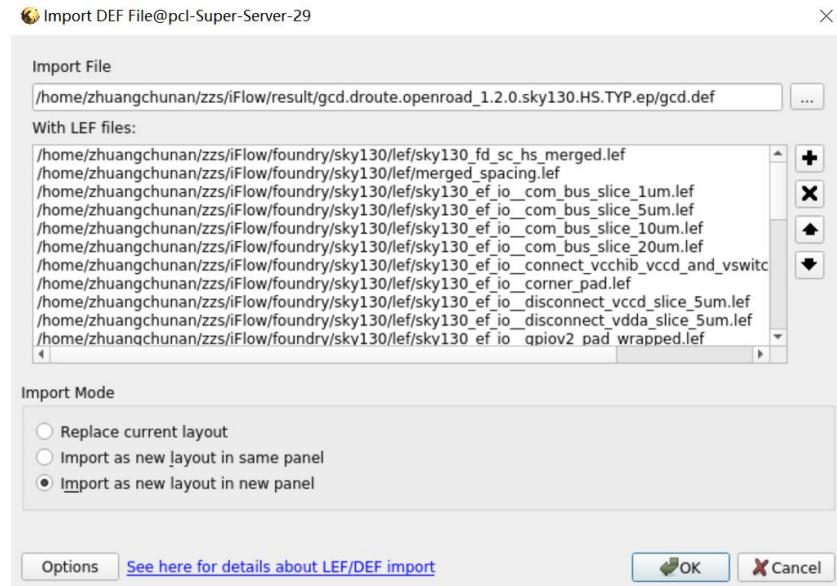


图 30 klayout 读入 def 和 lef

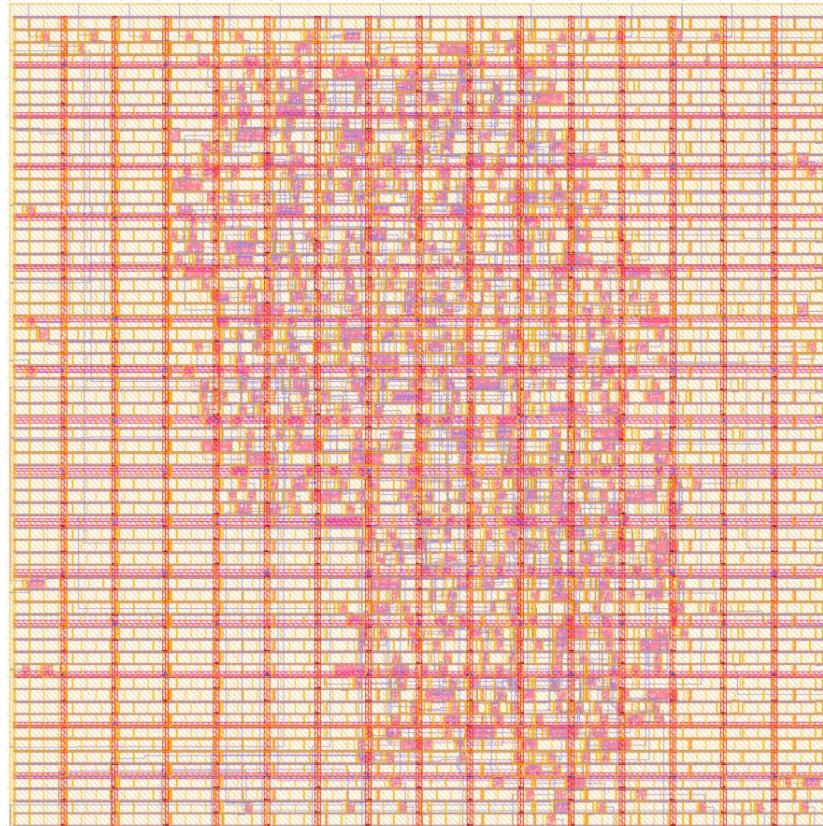


图 31 klayout 显示 def 的结果

klayout 也可以直接打开 GDS，使用 klayout 打开 GDS 的命令如下：

klayout xxxx.gds

用 klayout 打开 gcd 绕线后的 gds，如图 32 所示。

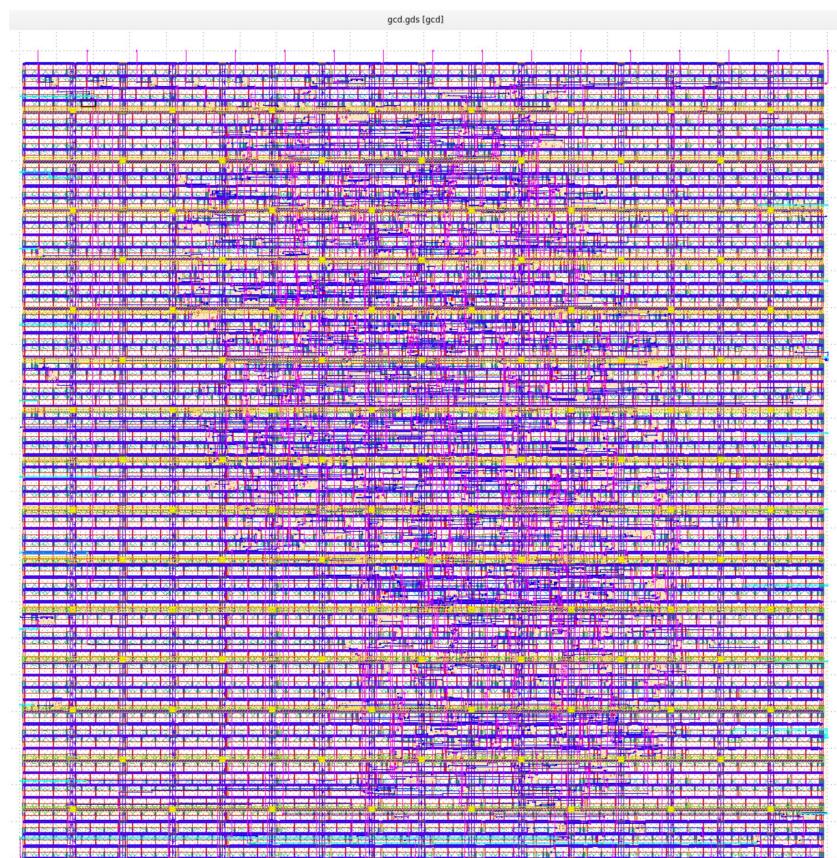


图 31 klayout 显示 gds 的结果

开源 EDA 流程 iFlow 使用示例——更换设计

一、gcd 设计

gcd(greatest common denominator)是一个计算最小公分母的设计，是一个非常小的设计，一个百门级别的芯片。使用 sky130 工艺库，去实现 gcd 设计的后端物理设计的流程及运行的命令可以参考上一部分的 iFlow 介绍。从综合流程的 log 显示，如图 33 所示，gcd 设计综合后一共有 639 个 cell，是一个非常小的设计。

| | |
|-----------------------------|------|
| Number of wires: | 693 |
| Number of wire bits: | 1044 |
| Number of public wires: | 147 |
| Number of public wire bits: | 498 |
| Number of memories: | 0 |
| Number of memory bits: | 0 |
| Number of processes: | 0 |
| Number of cells: | 639 |

图 33 gcd 综合网表部分特征参数

二、uart 设计

uart 设计是一种通用串行数据总线的设计，是一个百门级到千门级的设计，用于异步通信。下面将讲述一下如何将 iFlow 中的 design 更换为 uart 设计。

首先，需要把 uart 相关的 rtl 代码放在“iFlow/rtl”目录下，如图 34 所示，然后要定义 uart 设计的 flow，进入到“iFlow/scripts/cfg”目录下，编辑脚本“flow_cfg.py”，加入 uart 设计的默认 flow 参数，如图 35 所示，这里设置的默认 foundry 为“asap7”，改为“sky130”也是可以的。

```
zhuangchun'an@12:16 ~/zzs/iFlow/rtl
^0^ ls
aes_cipher_top coyote_tc gcd ibex_core uart
```

图 34 rtl 目录

```
uart = Flow('uart','asap7','HS','TYP')
```

图 35 flow 定义

定义好 flow 之后，我们需要准备 uart 设计的流程脚本，进入到“iFlow/scripts”目录，运行命令：

```
cp -r gcd uart
```

拷贝 gcd 设计的脚本作为 uart 设计的脚本，再进行相应的参数修改即可。首先，进入“iFlow/scripts/uart”目录，修改综合脚本“synth.yosys_0.9.tcl”，需要把输入的 Verilog 代码文件改为 uart 设计的 rtl 代码，如图 36 所示。

```
set VERILOG_FILES " \
$RTL_PATH/uart.v \
$RTL_PATH/uart_rx.v \
$RTL_PATH/uart_tx.v \
"
```

图 36 uart rtl 代码输入

由于 uart 设计的规模和 gcd 设计非常接近，在使用 sky130 工艺的情况下，我们可以沿用 gcd 设计的 floorplan 设置，也可以适当的调节 DIE_AREA 和 CORE_AREA，如图 37 所示，

```
set DIE_AREA      "0 0 220.2 220.2"
set CORE_AREA     "1.08 1.08 219.12 219.12"
```

图 37 floorplan 参数设置

其余的步骤可以不用修改，进入目录“iFlow/scripts”，运行命令：

```
./run_flow.py -d uart -s
synth,floorplan,tapcell,pdn,gplace,resize,dplace,cts,filler,groute,droute,layout -f sky130 -t HS
-c TYP -v V1 -l V1
```

即可基于 sky130 工艺跑 uart 设计的后端流程，结果如图 38 所示。

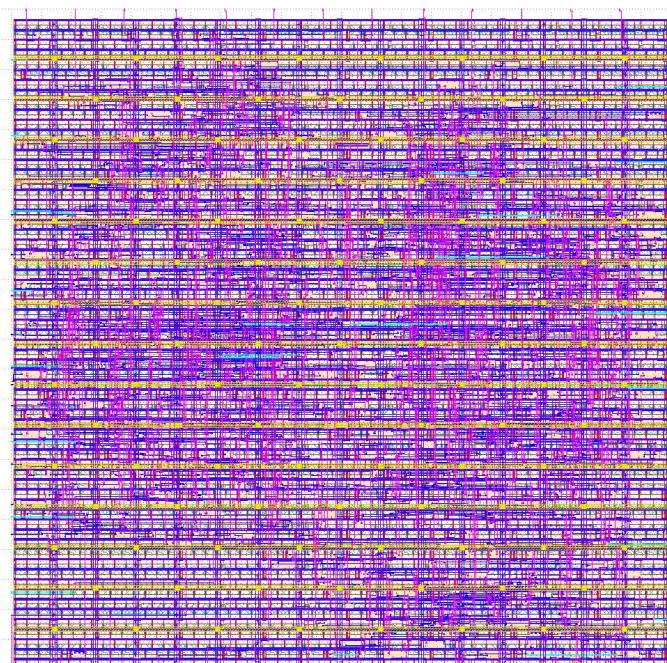


图 38 uart 版图

三、aes_cipher_top 设计

aes_cipher_top 是一个加密算法的小模块，相对于前面两个设计，aes_cipher_top 的规模要大很多，是一个万门级的设计。与 uart 设计一样，首先要修改综合脚本中的 Verilog 代码路径，然后调整 floorplan，增大芯片的面积，如图 39 所示。

```
set DIE_AREA      "0 0 1120 1020.8"
set CORE_AREA     "10 12 1110 1011.2"
```

图 39 aes_cipher_top 的 floorplan 参数设置

然后进入目录“iFlow/scripts”，运行命令

```
./run_flow.py -d aes_cipher_top -s
synth,floorplan,tapcell,pdn,gplace,resize,dplace,cts,filler,groute,droute,layout -f sky130 -t HS
-c TYP -v V1 -l V1
```

即可完成基于 sky130 工艺跑 aes_cipher_top 设计的后端流程，结果如图 40 所示，这个设计比较大，后端流程中 droute 步骤需要比较长的时间，大约需要一个小时。

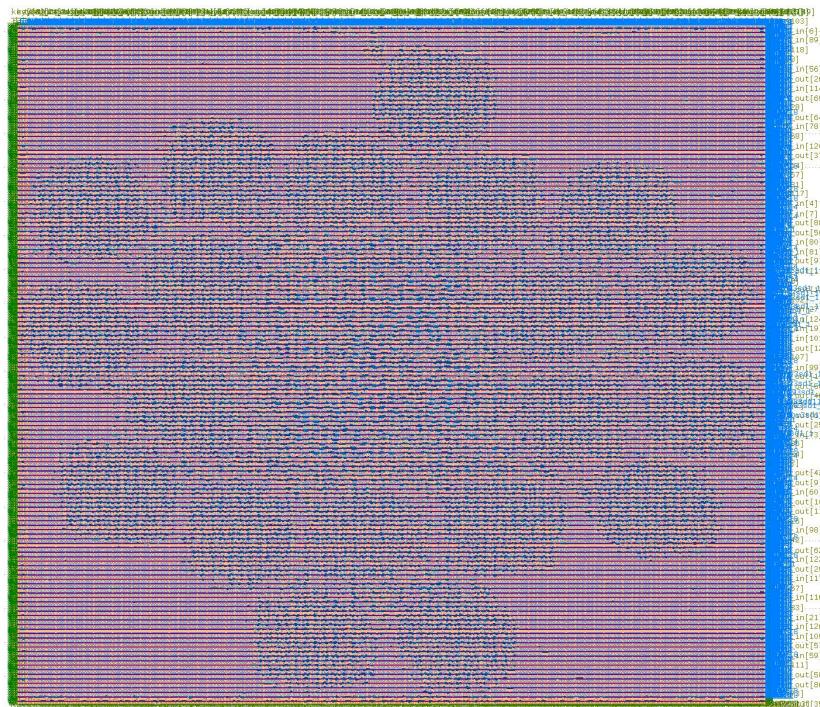


图 40 aes_cipher_top 的版图

练习：picorv32 设计更换并跑通后端流程

picorv32 是一个实现 RISC-V RV32IMC 指令集的 CPU 内核，大家可以尝试更换 picorv32 设计跑后端流程。

picorv32 代码源地址：<https://github.com/YosysHQ/picorv32>

开源 EDA 流程 iFlow 使用示例——更换工艺库

一、 nangate45

Nangate45 PDK 的源地址为：<https://eda.ncsu.edu/freepdk/>。在 iFlow 中的 nangate45 工艺库是经过整理的。

要想用 nangate45 工艺库来设计后端，首先要将 nangate45 工艺库加到 iFlow 中，将 nangate45 工艺库整理后放在“iFlow/foundry”目录下。然后进入“iFlow/scripts/cfg”目录，编辑脚本“foundry_cfg.py”，配置好 lib、lef 和 gds 库的路径以及综合阶段需要禁掉的单元列表“don't use list”。

```
#-----#
# nangate45
#
nangate45 = Foundry(
    name='nangate45',
    lib = {
        'std,HD,TYP' : (
            'foundry/nangate45/lib/NangateOpenCellLibrary_typical.lib',
        ),
        'dontuse' : 'TAPCELL_X1 FILLCELL_X1 AOI211_X1 OAI211_X1',
        'macro,TYP' : (
            'foundry/nangate45/lib/fakeram45_32x64.lib',
            'foundry/nangate45/lib/fakeram45_64x7.lib',
            'foundry/nangate45/lib/fakeram45_64x15.lib',
            'foundry/nangate45/lib/fakeram45_64x21.lib',
            'foundry/nangate45/lib/fakeram45_64x32.lib',
            'foundry/nangate45/lib/fakeram45_64x96.lib',
            'foundry/nangate45/lib/fakeram45_256x34.lib',
            'foundry/nangate45/lib/fakeram45_256x95.lib',
            'foundry/nangate45/lib/fakeram45_256x96.lib',
            'foundry/nangate45/lib/fakeram45_512x64.lib',
            'foundry/nangate45/lib/fakeram45_1024x32.lib',
            'foundry/nangate45/lib/fakeram45_2048x39.lib'
        ),
    },
    lef = {
        'tech' : (
            'foundry/nangate45/lef/NangateOpenCellLibrary.tech.lef',
        ),
        'std,HD' : (
            'foundry/nangate45/lef/NangateOpenCellLibrary.macro.mod.lef',
        ),
        'macro' : (
            'foundry/nangate45/lef/fakeram45_32x64.lef',
            'foundry/nangate45/lef/fakeram45_64x7.lef',
            'foundry/nangate45/lef/fakeram45_64x15.lef',
            'foundry/nangate45/lef/fakeram45_64x21.lef',
            'foundry/nangate45/lef/fakeram45_64x32.lef',
            'foundry/nangate45/lef/fakeram45_64x96.lef',
            'foundry/nangate45/lef/fakeram45_256x34.lef',
            'foundry/nangate45/lef/fakeram45_256x95.lef',
            'foundry/nangate45/lef/fakeram45_256x96.lef',
            'foundry/nangate45/lef/fakeram45_512x64.lef',
            'foundry/nangate45/lef/fakeram45_1024x32.lef',
            'foundry/nangate45/lef/fakeram45_2048x39.lef'
        )
    },
    gds = {
        'std,HD' : (
            'foundry/nangate45/gds/NangateOpenCellLibrary.gds',
        ),
        'macro' : (
        )
    }
)
```

图 41 nangate45 工艺配置

然后需要在脚本中加入 nangate45 工艺库的参数设置，这里用 aes_cipher_top 设计的综合脚本举例，如图 42 所示，对于不同的工艺库，所用到的 TIE cell 和

buffer 名称是不一样的，需要根据工艺库进行修改，对于其他步骤的脚本也是如此。

```
if { $FOUNDRY == "sky130" } {
    if { $TRACK == "HS" } {
        set TIEHI_CELL_AND_PORT      "sky130_fd_sc_hs_conb_1 HI"
        set TIELO_CELL_AND_PORT      "sky130_fd_sc_hs_conb_1 LO"
        set MIN_BUF_CELL_AND_PORTS   "sky130_fd_sc_hs_buf_1 A X"
    } elseif { $TRACK == "HD" } {
        set TIEHI_CELL_AND_PORT      "sky130_fd_sc_hd_conb_1 HI"
        set TIELO_CELL_AND_PORT      "sky130_fd_sc_hd_conb_1 LO"
        set MIN_BUF_CELL_AND_PORTS   "sky130_fd_sc_hd_buf_1 A X"
    }
} elseif { $FOUNDRY == "nangate45" } {
    set TIEHI_CELL_AND_PORT      "LOGIC1_X1 Z"
    set TIELO_CELL_AND_PORT      "LOGIC0_X1 Z"
    set MIN_BUF_CELL_AND_PORTS   "BUF_X1 A Z"
} elseif { $FOUNDRY == "asap7" } {
    set TIEHI_CELL_AND_PORT      "TIEHIX1_ASAP7_75t_R H"
    set TIELO_CELL_AND_PORT      "TIELOx1_ASAP7_75t_R L"
    set MIN_BUF_CELL_AND_PORTS   "BUFx2_ASAP7_75t_R A Y"
}
```

图 42 nangate45 工艺综合参数配置

其中电源网络的脚本，需要引用 nangate45 工艺的电源网络配置“pdn_nangate45.cfg”，因为每个工艺电源网络的配置差别是比较大的。脚本参数修改完之后，运行命令：

```
./run_flow.py -d aes_cipher_top -s
synth,floorplan,tapcell,pdn,gplace,resize,dplace,cts,filler,groute,droute,layout -f nangate45 -t
HD -c TYP -v V1 -l V1
```

基于 nangate45 工艺跑 aes_cipher_top 设计的后端结果如图 43 所示。

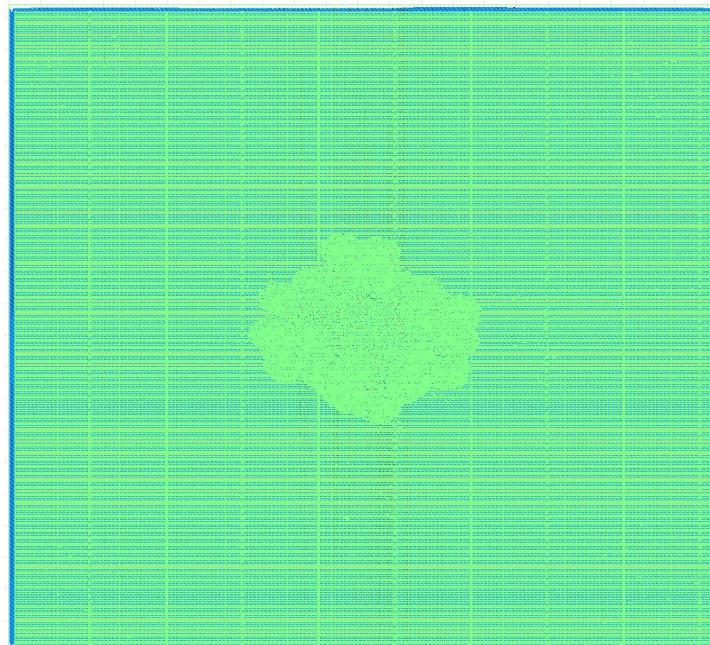


图 43 aes_cipher_top 的 nangate45 工艺版图

二、asap7

asap7 的源地址为: <https://asap.asu.edu/>。在 iFlow 中的 asap7 工艺库是经过整理的。

asap7 是开源的 7nm 工艺，因此我们需要把 floorplan 面积调得更小，以保证在一定的利用率下能够顺利布线，和 nangate45 工艺类似，修改相应的脚本参数之后，如图 44 和 45 所示，

```
set DIE_AREA      "0 0 16.2 16.2"  
set CORE_AREA     "1.08 1.08 15.12 15.12"
```

图 44 调节 floorplan 参数

```
set PLACE_DENSITY "0.5"
```

图 45 调节 gplace 参数

运行命令：

```
./run_flow.py -d gcd -s  
synth,floorplan,tapcell,pdn,gplace,resize,dplace,cts,filler,groute,droute,layout -f asap7 -t HS  
-c TYP -v V1 -l V1
```

即可完成基于 asap7 工艺跑 gcd 设计的后端流程，droute 步骤大约需要两小时，结果如图 46 所示。

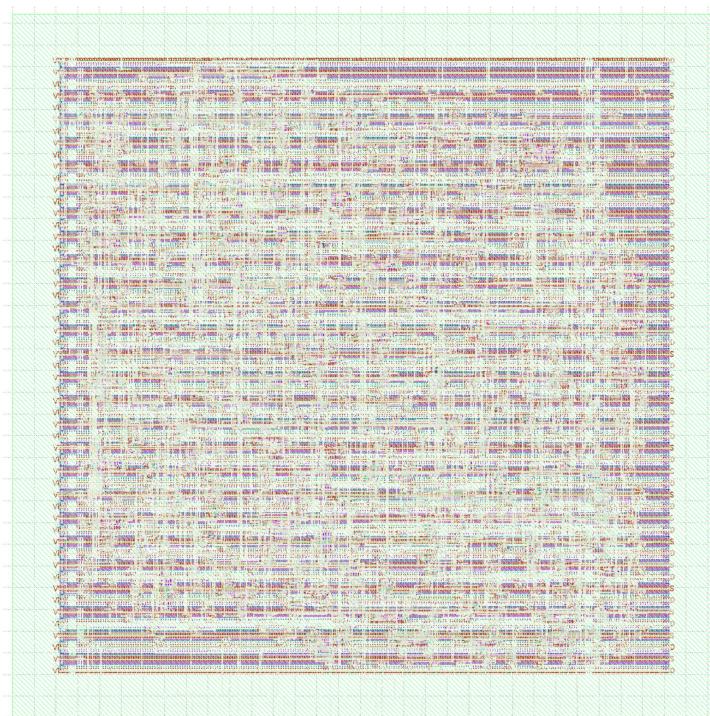


图 46 gcd 的 asap7 工艺版图

大家也可以尝试一下用 asap7 工艺去跑 uart 设计，但不建议用来尝试 aes_cipher_top 设计，因为 aes_cipher_top 设计比较大，droute 步骤会存在很多 DRC 违例，工具在解 DRC 时需要花费大量时间，可能还绕不通线。