《高级计算机体系结构 课程实验报告》

授课教师：	沈海华
姓　　名：	刘炼
学　　号：	202128013229021
日　　期：	完成日期

高级计算机系统结构 课程实验报告

刘炼，202128013229021

中国科学院大学 计算机学院

1 Build iFlow & Run Example

```
git clone https://github.com/PCNL-EDA/iFlow.git
cd iFlow
./build_iflow.sh  # 在 build_iflow 中，存在很多 sudo 权限的内容，实际上应该删去改内容
```

整个项目目录如下所示：

```
root@00e9253897ee:/home/iFlow# tree -d -L 2
.
|-- foundry
|   |-- asap7
|   |-- nangate45
|   `-- sky130
|-- log
|-- report
|   |-- aes_cipher_top.synth.yosys_0.9.sky130.HS.TYP.default
|   `-- gcd.synth.yosys_0.9.sky130.HS.TYP.default
|-- result
|   |-- aes_cipher_top.synth.yosys_0.9.sky130.HS.TYP.default
|   `-- gcd.synth.yosys_0.9.sky130.HS.TYP.default
|-- rtl
|   |-- aes_cipher_top
|   |-- gcd
|   `-- uart
|-- scripts
|   |-- aes_cipher_top
|   |-- cfg
|   |-- common
|   |-- gcd
|   |-- shell
|   `-- uart
|-- tools
|   |-- OpenROAD9295a533
|   |-- OpenROAD_fixcts
|   |-- OpenROADae191807
|   `-- yosys4be891e8
`-- work
    |-- aes_cipher_top.synth.yosys_0.9.sky130.HS.TYP.default
    `-- gcd.synth.yosys_0.9.sky130.HS.TYP.default
```

1.1 顶层与配置脚本

1.1.1 run_flow.py

运行`./run_flow.py -h`，可以得到如下的说明：

```
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 tapcell 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
```

可以看到，通过选择不同的参数，可以设置不同的RTL 设计 （-d design) , 后端设计的不同步骤 （-s step) 等内容。

1.1.2 configuration

在`iFlow/scripts/cfg`文件夹中，包含了四个脚本文件：`data_def.py`，`flow_cfg.py`，`foundry_cfg.py`和`tools_cfg.py`，分别用来控制数据的定义， 流程配置，工艺库配置以及工具版本的配置。

```
  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           = ('synth','floorplan','tapcell','pdn','gplace','resize','dplace','cts','filler','groute','droute','layout')
      self.tool           = {
          'synth'     :   'yosys_0.9'         ,
          'v2def'     :   'openroad_1.2.0'    ,
          'floorplan' :   'openroad_1.2.0'    ,
           'floorplan' :   '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'    ,
           'groute'    :   'iGR'     ,
          'groute'    :   'openroad_1.2.0'    ,
           'droute'    :   'iDR'     ,
          'droute'    :   'openroad_1.2.0'    ,
          'layout'    :   'klayout_0.26.2'
      }
```

在`data_def.py`中，如上图所示，蓝框标识了后端的不同步骤，并且设置了每个步骤所使用的工具，可以更改这些工具内容。而在`flow_cfg.py`中，设置了默认的参数，因此可以不用设置工艺相关的参数而直接运行命令，同样能够得到结果。

例如运行命令：

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

实际上对aes_cipher_top 设置了默认的参数为：`sky130`，`HS` 和 `TYP`。

而对于文件`floundry_cfg.py`，其中定义了不同工艺节点的库文件路径，如果要加入新的工艺，则需要将其添加到这个文件中，截取sky130的工艺库路径的部分展示如下：

```
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* *221* *311* *32* *41* *clk* *dly* *nand4* *or4*
        'macro,TYP' : (
            'foundry/sky130/lib/sky130_dummy_io.lib',
            'foundry/sky130/lib/sky130_sram_1rw1r_128x256_8_TT_1p8V_25C.lib',
            'foundry/sky130/lib/sky130_sram_1rw1r_44x64_8_TT_1p8V_25C.lib',
            'foundry/sky130/lib/sky130_sram_1rw1r_64x256_8_TT_1p8V_25C.lib',
            'foundry/sky130/lib/sky130_sram_1rw1r_80x64_8_TT_1p8V_25C.lib'
        ),
    },
    lef = {
        'tech'       : (
            'foundry/sky130/lef/sky130_fd_sc_hs.tlef',
        ),
        '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',
```

而文件`tools_cfg.py`用于配置每一步要用哪种开源EDA 工具及其对应的版本号。

1.2 iFlow 流程介绍

1.2.1 综合

综合的目的是将RTL代码转化为网表，在对应的`.tcl`文件中，需要配置相应的参数来实现，需要配置的内容包括：

- 综合需要读入的库文件，例如blackbox 的 verilog 文件 和 map 文件
- 特定的cell，包括tie cell 和 buffer
- 综合所需要的RTL 代码路径

在配置好相应的内容之后，运行单步综合命令如下：

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

可以看到，在`report`文件夹下得到了`gcd`综合相关的文件夹，其中包含了`synth_check.txt` 和 `synth_stat.txt` 文件

```
root@00e9253897ee:/home/iFlow/report/gcd.synth.yosys_0.9.sky130.HS.TYP.default# ls
synth_check.txt  synth_stat.txt
```

1.2.2 布局

在iFlow中，布局包括了六个小步骤，分别是`floorplan`，`tapcell`，`PDN`，`gplace`，`resize`和 `dplace`。

1.2.2.1 floorplan

根据gcd中`floorplan`的脚本，配置其中的“DIE_AREA" 和 "CORE_AREA"参数，具体如下图所示：

```
#=========================================================
#   set tool related parameter
#=========================================================
#set DIE_AREA              "0 0 16.2 16.2"
#set CORE_AREA             "1.08 1.08 15.12 15.12"
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"
```

配置的track 对应参数为：

```
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
```

最终执行单步`floorplan`的命令如下：

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

得到 其 check 如下：

```
  Delay    Time   Description
---------------------------------------------------------
   0.00    0.00   clock core_clock (rise edge)
   0.00    0.00   clock network delay (ideal)
   0.00    0.00 ^ dpath/b_reg/_133_/CLK (sky130_fd_sc_hs__dfxtp_1)
   0.22    0.22 ^ dpath/b_reg/_133_/Q (sky130_fd_sc_hs__dfxtp_1)
   0.15    0.37 ^ dpath/sub/_214_/X (sky130_fd_sc_hs__buf_1)
   0.15    0.51 ^ dpath/sub/_124_/Y (sky130_fd_sc_hs__nand2b_4)
   0.08    0.59 v dpath/sub/_126_/Y (sky130_fd_sc_hs__nand3_2)
   0.07    0.66 ^ dpath/sub/_127_/Y (sky130_fd_sc_hs__nand2_2)
   0.07    0.73 v dpath/sub/_135_/Y (sky130_fd_sc_hs__nand3_4)
   0.11    0.84 ^ dpath/sub/_162_/Y (sky130_fd_sc_hs__a21boi_4)
   0.06    0.90 v dpath/sub/_166_/Y (sky130_fd_sc_hs__nor2b_2)
   0.11    1.01 v dpath/sub/_186_/Y (sky130_fd_sc_hs__nand2b_2)
   0.18    1.19 ^ dpath/sub/_203_/Y (sky130_fd_sc_hs__a211oi_2)
   0.06    1.25 v dpath/sub/_205_/Y (sky130_fd_sc_hs__o21ai_2)
   0.05    1.30 ^ dpath/sub/_208_/Y (sky130_fd_sc_hs__nand2_1)
   0.14    1.45 v dpath/sub/_209_/X (sky130_fd_sc_hs__xnor3_1)
   0.08    1.53 v dpath/sub/_257_/X (sky130_fd_sc_hs__buf_1)
   0.00    1.53 v resp_msg[15] (out)
           1.53   data arrival time

 420.00  420.00   clock core_clock (rise edge)
   0.00  420.00   clock network delay (ideal)
   0.00  420.00   clock reconvergence pessimism
 -84.00  336.00   output external delay
         336.00   data required time
---------------------------------------------------------
         336.00   data required time
          -1.53   data arrival time
---------------------------------------------------------
         334.47   slack (MET)
```

1.2.2.2 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 阱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。其基本配置可以参考配置文件：

```
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"
}
```

执行对应的`tapcell`命令如下：

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

得到了最终的tapcell 结果

1.2.2.3 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。其脚本是比较简单的，具体为：

```
pdngen $PDN_CFG_FILE --verbose
```

执行对应的`PDN`命令如下：

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

其具体对应的配置实例如下图所示：

```
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}}
}
```

1.2.2.4 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，需要配置线RC参数的抽取层以评估延时，而另一个参数用于设置摆放标准单元时的密度。 具体的运行gplace的命令如下图所示：

```
global_placement -density $PLACE_DENSITY

#global_placement -incremental -overflow 0.1 -density $PLACE_DENSITY
```

其中默认overflow 为 0.1 ，执行单步`gplace`命令为：

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

可以观察到其最终的overflow 会小于0.1 ,迭代超过了 360 轮

```
[NesterovSolve] Iter: 1 overflow: 0.886762 HPWL: 10352089
[NesterovSolve] Iter: 10 overflow: 0.623103 HPWL: 13773677
[NesterovSolve] Iter: 20 overflow: 0.569885 HPWL: 13634061
[NesterovSolve] Iter: 30 overflow: 0.568019 HPWL: 13531788
[NesterovSolve] Iter: 40 overflow: 0.570133 HPWL: 13539674
[NesterovSolve] Iter: 50 overflow: 0.570887 HPWL: 13560444
[NesterovSolve] Iter: 60 overflow: 0.572603 HPWL: 13566460
[NesterovSolve] Iter: 70 overflow: 0.572458 HPWL: 13563518
[NesterovSolve] Iter: 80 overflow: 0.571832 HPWL: 13558287
[NesterovSolve] Iter: 90 overflow: 0.571173 HPWL: 13555546
[NesterovSolve] Iter: 100 overflow: 0.570576 HPWL: 13555987
[NesterovSolve] Iter: 110 overflow: 0.570741 HPWL: 13555874
[NesterovSolve] Iter: 120 overflow: 0.570872 HPWL: 13558379
[NesterovSolve] Iter: 130 overflow: 0.570502 HPWL: 13563954
[NesterovSolve] Iter: 140 overflow: 0.569819 HPWL: 13573371
[NesterovSolve] Iter: 150 overflow: 0.568827 HPWL: 13587350
[NesterovSolve] Iter: 160 overflow: 0.567058 HPWL: 13608804
[NesterovSolve] Iter: 170 overflow: 0.564073 HPWL: 13634437
[NesterovSolve] Iter: 180 overflow: 0.558837 HPWL: 13651451
[NesterovSolve] Iter: 190 overflow: 0.550515 HPWL: 13634385
[NesterovSolve] Iter: 200 overflow: 0.540661 HPWL: 13575273
[NesterovSolve] Iter: 210 overflow: 0.528475 HPWL: 13551740
[NesterovSolve] Iter: 220 overflow: 0.512137 HPWL: 13603953
[NesterovSolve] Iter: 230 overflow: 0.482064 HPWL: 13497089
[NesterovSolve] Iter: 240 overflow: 0.454317 HPWL: 13460888
[NesterovSolve] Iter: 250 overflow: 0.418363 HPWL: 13339128
[NesterovSolve] Iter: 260 overflow: 0.386861 HPWL: 13273176
[NesterovSolve] Iter: 270 overflow: 0.34048 HPWL: 13243275
[NesterovSolve] Iter: 280 overflow: 0.313243 HPWL: 13198526
[NesterovSolve] Iter: 290 overflow: 0.281849 HPWL: 13203447
[NesterovSolve] Iter: 300 overflow: 0.248375 HPWL: 13185690
[NesterovSolve] Iter: 310 overflow: 0.218497 HPWL: 13111927
[NesterovSolve] Iter: 320 overflow: 0.192366 HPWL: 13137642
[NesterovSolve] Iter: 330 overflow: 0.165354 HPWL: 13148862
[NesterovSolve] Iter: 340 overflow: 0.145061 HPWL: 13146461
[NesterovSolve] Iter: 350 overflow: 0.126217 HPWL: 13162066
[NesterovSolve] Iter: 360 overflow: 0.110003 HPWL: 13181950
[NesterovSolve] Finished with Overflow: 0.099568
```

1.2.2.5 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。这一步骤需要配置的参数如下图所示：

```
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"
    }
```

包括了最大扇出和固定的tie cell 和 buffer 类型。在这一流程中，主要进行fanout 的 修复，降低fanout 以增加各级的驱动能力。执行单步`resize`命令如下：

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

会在report 中保存 resize 前后的差别

```
|-- gcd.resize.openroad_1.2.0.sky130.HS.TYP.default
|   |-- post_resize.rpt
|   `-- pre_resize.rpt
```

1.2.2.6 dplance

dplace 是布局中的最后一步，主要是对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 范围的Row 上，从而确保电源网络能为标准单元供电。

执行单步`dplace`命令如下：

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

可以得到最终的布局阶段的分析：

```
Placement Analysis
---------------------------------
total displacement       7769.9 u
average displacement        6.1 u
max displacement           67.6 u
original HPWL           13312.4 u
legalized HPWL          23440.5 u
delta HPWL                    76 %
```

1.2.3 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。

首先需要设置用于构建时钟树的buffer 的 cell 类型，其在配置文件中表示为：

```
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"
}
```

执行单步`CTS`命令如下：

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

得到其分析为：

```
Placement Analysis
--------------------------------
total displacement         57.5 u
average displacement        0.0 u
max displacement           15.9 u
original HPWL           23815.3 u
legalized HPWL          23818.5 u
delta HPWL                     0 %
```

可以看到，与 CTS 步骤之前相比，整体的分析有了很大的不同。

1.2.4 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 fillercell，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。需要设置filler cell 的类型，具体如图所示：

```
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"
}
```

执行单步`filler`命令如下：

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

生成得到了.def 文件。

1.2.5 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。

1.2.5.1 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。

在开始groute前，如果有用到SRAM 等marco，需要给marco 加上routing package，具体如下所示（由于这里使用的是gcd，暂时用不上，所以注释掉，在后续内容中，可能会使用到routing package）：

```
#[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
}
```

执行单步`groute`命令如下：

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

可以生成相应的.guide 文件

1.2.5.2 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。其主要是依赖于生成的route.guide 文件，没有额外的参数需要设置：

```
# generate droute.param file
source $GEN_PARAM

detailed_route -param $PARAM_FILE
```

执行单步`droute`命令如下：

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

1.2.6 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”。其中的layout 是将所得到的def 文件和标准单元中的黑盒进行合并，得到最终的gds 文件。

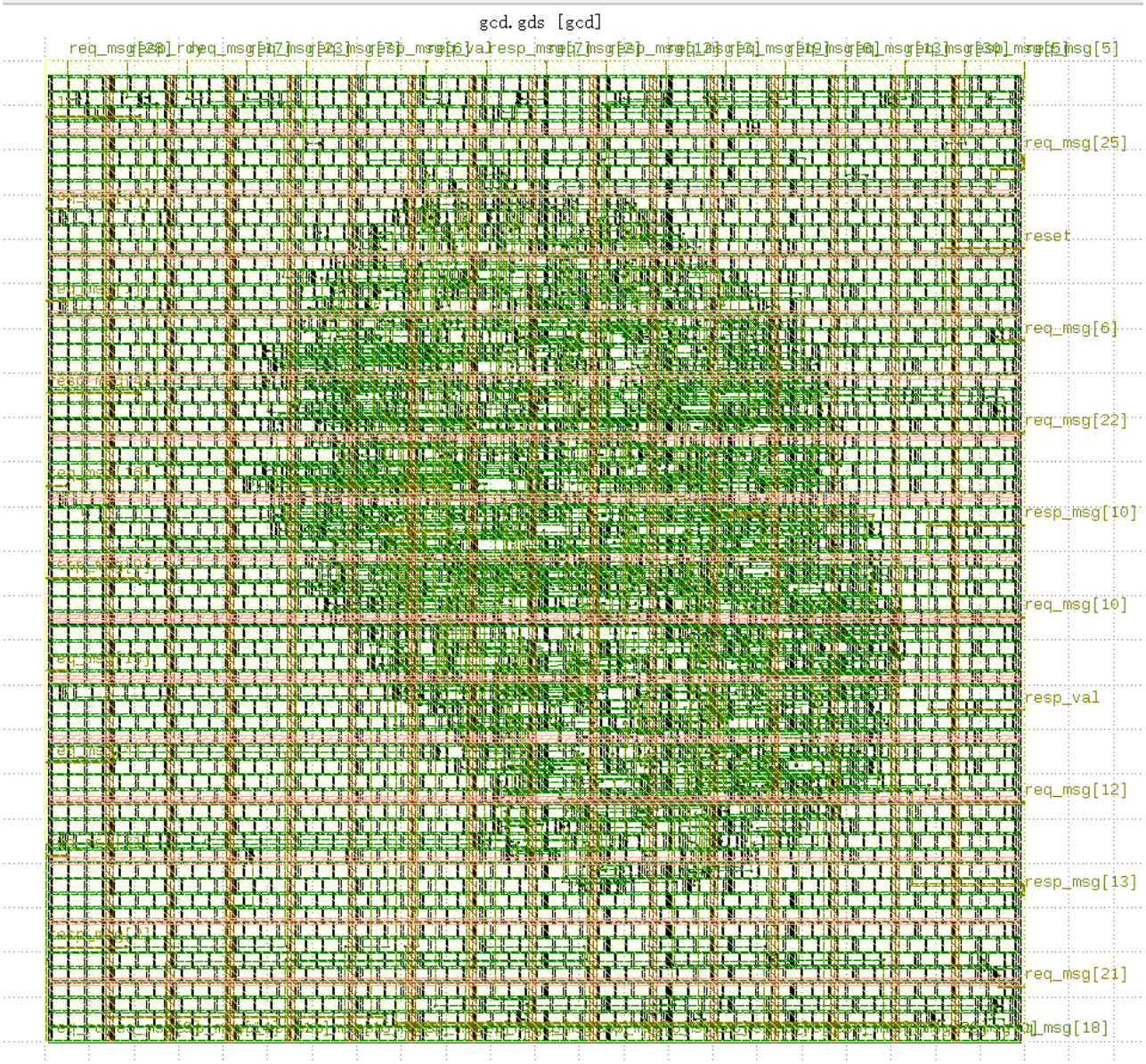
merge 过程的具体命令在顶层脚本“run_flow.py”中实现，如图 29 所示，merge 需要读入的文件包括 droute 输出的 def，还有标准单元、IO cell、marco 的gds 文件，以及工艺的 layer map 文件 klayout.lyt 和 klayout.lyp，最终输出 gds 版图。

```
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 &')
```

执行单步`layout (merge)`命令如下：

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

使用klayout 工具，查看gds 版图如下（所有中间生成结果和最终的gds文件均在 result 文件夹下。



2 更换设计

在前面的执行示例中，我们使用了gcd 来作为基本设计单元，其实际上是一个非常小的设计，只有百门级别的芯片。下面将介绍几个别的设计，进行更换。

2.1 uart 设计

2.1.1 拷贝rtl 代码并修改flow 定义

首先需要将uart 的 rtl 拷贝到 `iFlow/rtl` 中，而后需要修改默认flow 参数，这里的默认foundry 为 "asap7"，可以修改为"sky130"，具体如图所示：

2.1.2 设定rtl 代码脚本

首先将 gcd 设计脚本作为uart设计的脚本，并修改相应的参数，具体而言，首先修改综合脚本，将输入的verilog 代码改为 uart 设计的rtl 代码，如下图所示：

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

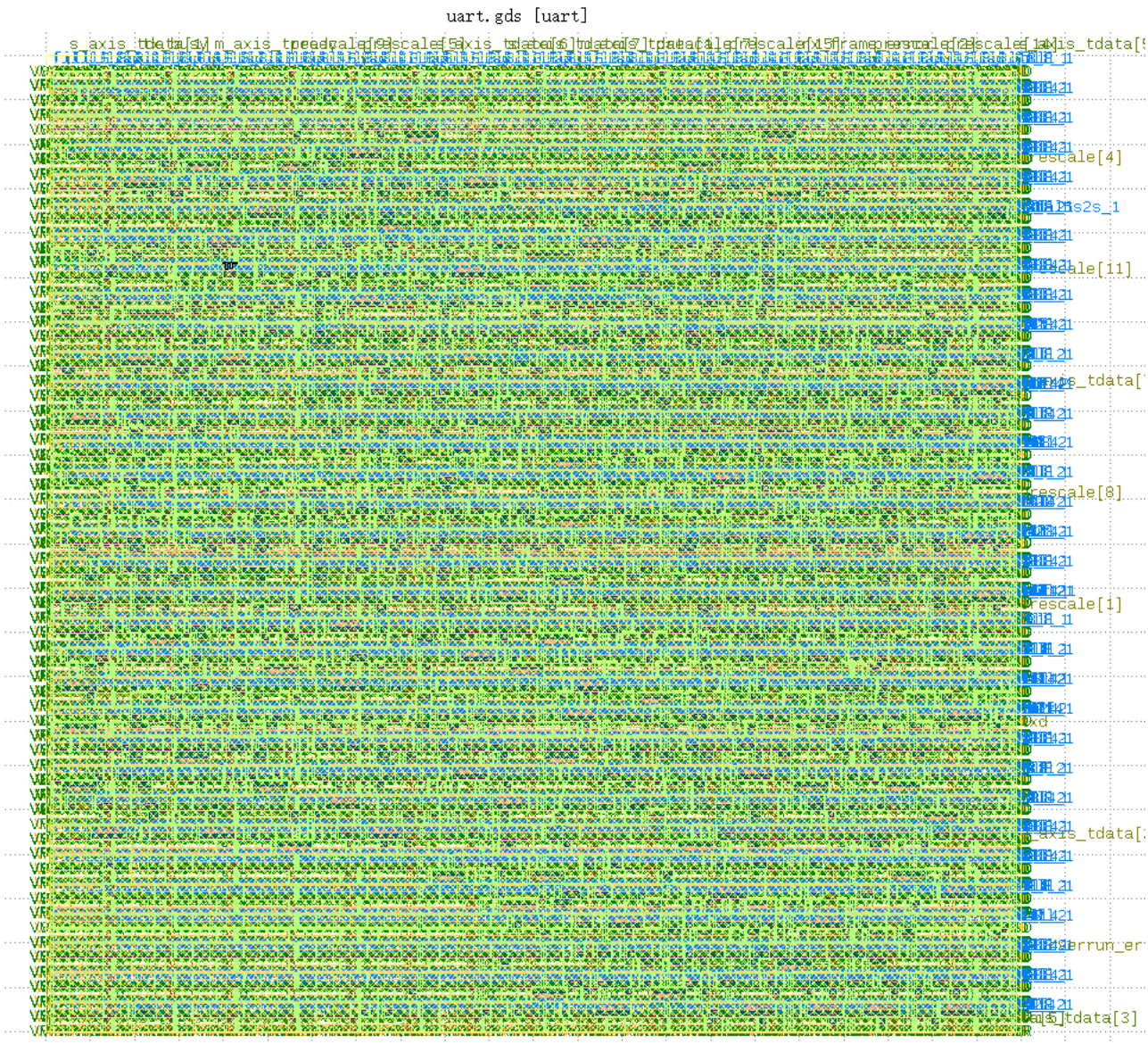
这里由于uart 设计的规模和gcd 设计非常接近，在使用sky130 工艺的情况下，我们可以沿用gcd 设计的floorplan 设置，也可以适当调节为如下：

```
#set DIE_AREA              "0 0 1120 1020.8"
#set CORE_AREA             "10 12 1110 1011.2"
set DIE_AREA              "0 0 220 220"
set CORE_AREA             "1.08 1.08 219 219"
```

最后执行`uart`的命令：

```
./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
```

得到最终的版图为：

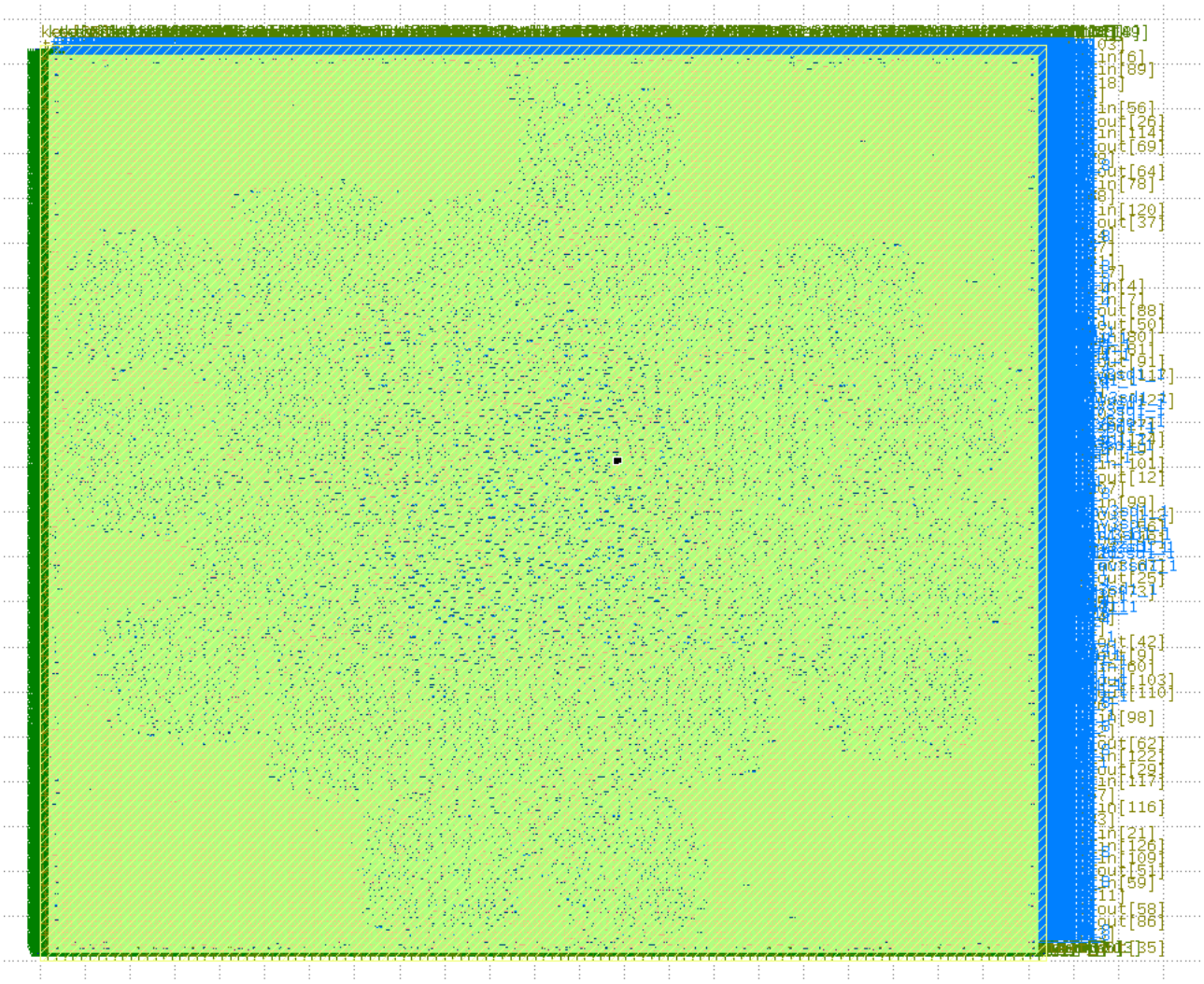


2.2 aes_cipher_top 设计

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，其面积设置为：

```
#============================================================
#   set tool related parameter
#============================================================
set DIE_AREA              "0 0 1120 1020.8"
set CORE_AREA             "10 12 1110 1011.2"
#set DIE_AREA              "0 0 100 100"
#set CORE_AREA             "1.08 1.08 99.12 99.12"
```

这里的操作是与之前uart 中的执行相同的，而后运行相应的命令生成`aes_cipher_top`的版图：



2.3 picorv32 设计更换

picorv32 是一个实现 RISC-V RV32IMC 指令集的 CPU 内核，其代码源地址为：https://github.com/YosysHQ/picorv32，下面将展示如何

2.3.1 下载rtl 代码 & 增加flow

```
git clone git@github.com:YosysHQ/picorv32.git # 拷贝数据到 rtl 文件夹下
#除了拷贝rtl 源代码之外，还需要加上.sdc 文件，否则会报错！！！
#拷贝.sdc 文件
cp -r aes_cipher_top/aes_cipher_top.sdc picorv32/
mv aes_cipher_top.sdc picorv32.sdc
```

查看`picorv32`的rtl 代码可知，其rtl 实现CPU 的代码为`picrov32.v`文件，因此，需要在后续综合脚本中导入该文件。 在cfg 中的`flow_cfg.py`文件中，添加对应的flow 为：

```
picorv32 = Flow('picorv32', 'sky130', 'HS', 'TYP')
```

2.3.2 脚本配置

首先将aes_cipher_top 的配置拷贝到picorv32中，指令如下：

```
cp -r aes_cipher_top/ picorv32
```

其中，对于综合脚本，需要修改rtl 源，如图所示

```
set VERILOG_FILES " \
$RTL_PATH/aes_cipher_top.v \
$RTL_PATH/aes_inv_cipher_top.v \
$RTL_PATH/aes_inv_sbox.v \
$RTL_PATH/aes_key_expand_128.v \
$RTL_PATH/aes_rcon.v \
$RTL_PATH/aes_sbox.v \
"
```

将`aes_cipher_top`中的源文件修改为：

```
set VERILOG_FILES " \
$RTL_PATH/picorv32.v \
"
```

观察综合之后的stas.txt 文件，可以发现，其结果如下图所示：

```
   Number of wires:                 13725
   Number of wire bits:             14107
   Number of public wires:           1569
   Number of public wire bits:       1951
   Number of memories:                  0
   Number of memory bits:               0
   Number of processes:                 0
   Number of cells:                 14005
```

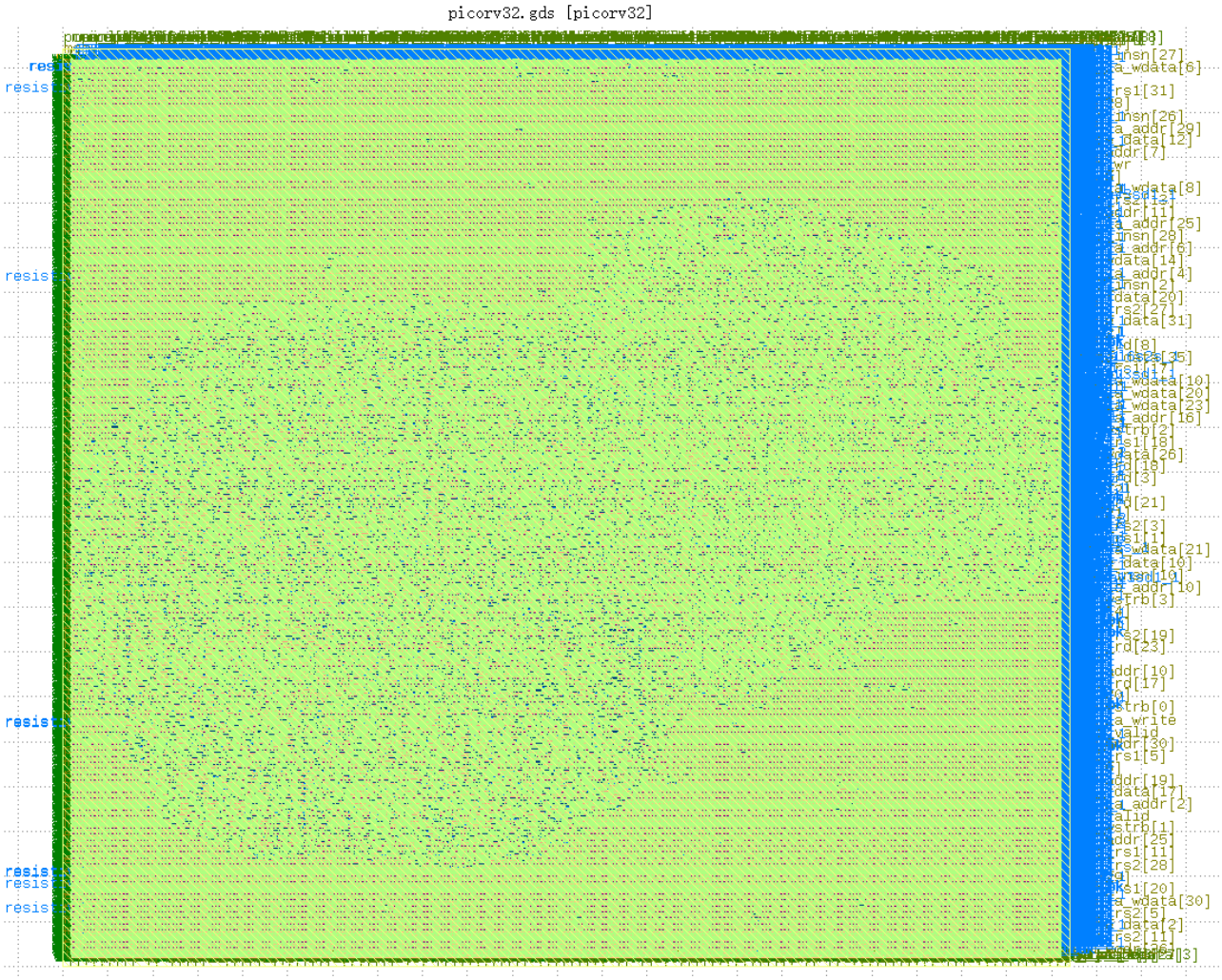
相较于`aes_cipher_top`而言，picorv 所拥有的cell 更少，所以使用`aes_cipher_top` 中 floorplan 中所设置的 DIE_AREA 和 CORE_AREA 是可行的。

2.3.3 运行后端流程

执行如下指令以运行`iFlow`后端流程：

```
./run_flow.py -d picorv32 -s \
synth,floorplan,tapcell,pdn,gplace,resize,dplace,cts,filler,groute,droute,layout
-f sky130 -t HS -c TYP -v V1 -l V
```

得到最终的版图文件为：



3 更换工艺库

3.1 nangate45

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，放在"iFlow/foundry" 目录下，然后进入"iFlow/scripts/cfg"目录，编辑"foundry_cfg.py"文件，配置好对应的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"。 具体如下图所示：

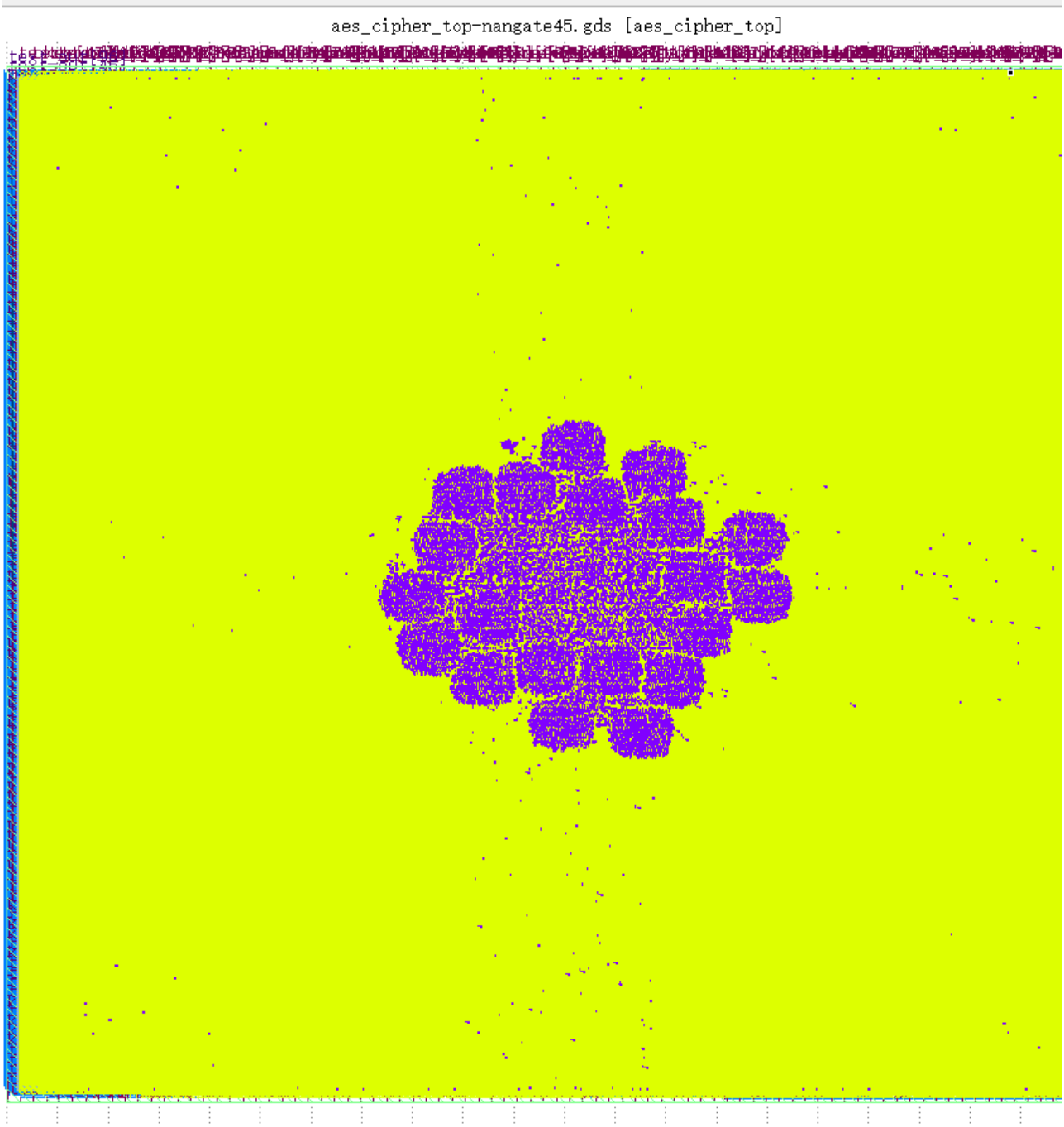
```
#-------------------------------------------------------------
# nangate45
#-------------------------------------------------------------
nangate45 = Foundry(
    name='nangate45',
    lib = {
        'std,HD,TYP': (
            'foundry/nangate45/lib/NangateOpenCellLibrary_typical.lib',
        ),
        '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'     : (
        )
    }
)
```

并且需要在综合脚本中添加其tie cell 和 buffer 名称以及内容，具体如下图所示：

```
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"
}
```

以`aes_cipher_top`为例，其基于nangate45工艺跑对应的设计，其生成的版图如下所示：

```
./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
```



3.2 asap7

3.2.1 gcd for asap7

在iFlow库中的asap7工艺库是经过整理的，asap7 是开源的7nm 工艺，因此我们需要吧floorplan 面积调得更小，以保证在一定的利用率下能够顺利布线，修改

floorplan 脚本，这里以gcd 为例，得到其结果为：

```
#========================================================
#   set tool related parameter
#========================================================
set DIE_AREA              "0 0 16.2 16.2"
set CORE_AREA             "1.08 1.08 15.12 15.12"
#set DIE_AREA              "0 0 220.2 220.2"
#set CORE_AREA             "1.08 1.08 219.12 219.12"
```

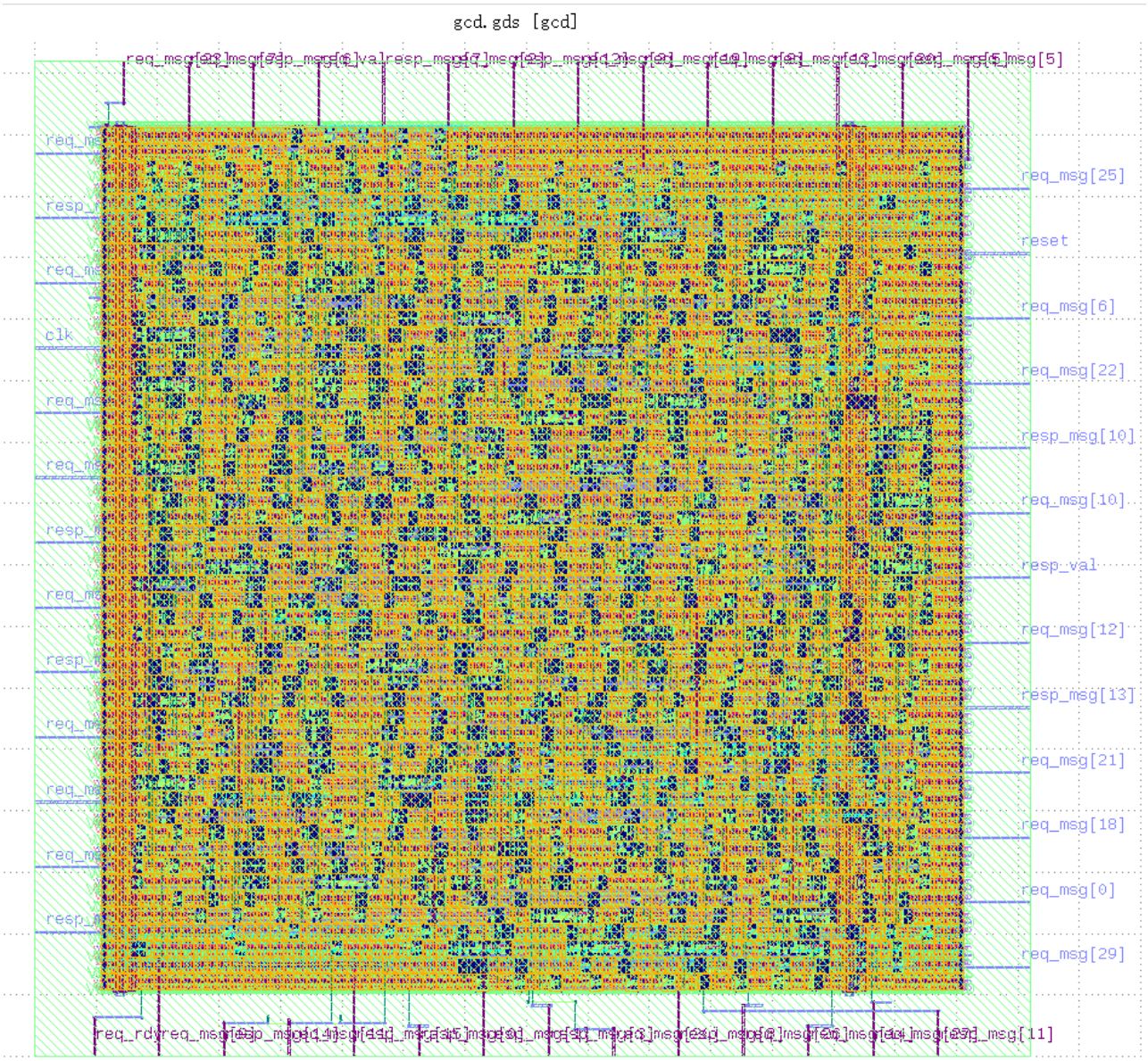
并且设置gplace 参数为：

```
set PLACE_DENSITY    "0.5"
```

运行如下指令：

```
./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
```

得到最终的版图如下所示：



3.2.2 uart for asap7

由于uart 中的单元和 gcd 是相似的，因此在设置floorplan 的时候，也需要考虑到具体的约束，因此DIE_AREA 和 CORE_AREA 分别设置为如下：

```
#=========================================================
#   set tool related parameter
#=========================================================
#set DIE_AREA              "0 0 1120 1020.8"
#set CORE_AREA             "10 12 1110 1011.2"
#set DIE_AREA              "0 0 220 220"
#set CORE_AREA             "1.08 1.08 219 219"
set DIE_AREA               "0 0 16.2 16.2"
set CORE_AREA              "1.08 1.08 15.12 15.12"
```

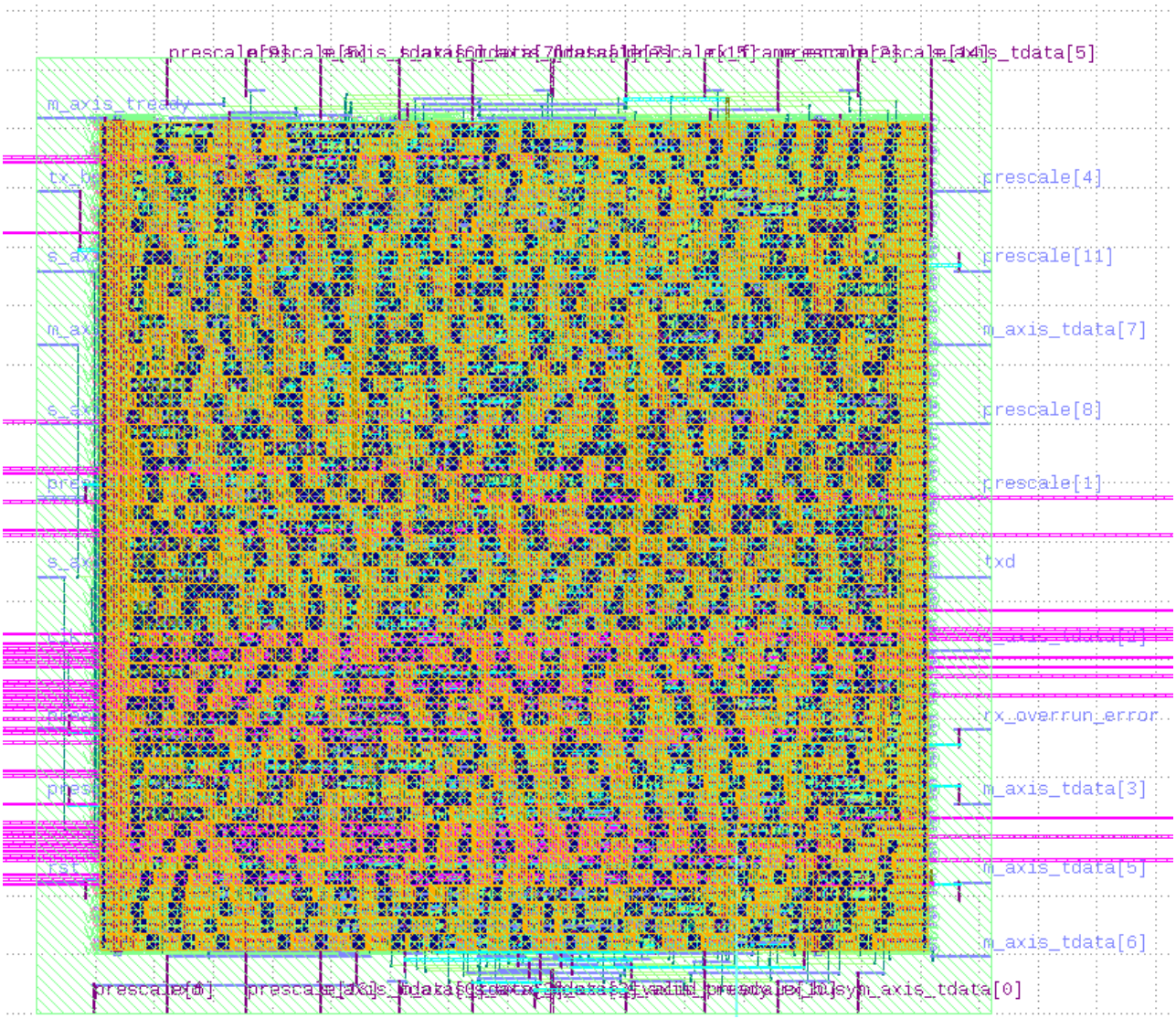
在进行gplace 时，也需要修改具体的`PLACE_DENSITY`，修改为0.8，如下图所示，否则会不满足约束：

```
set PLACE_DENSITY    "0.71"
```

运行如下指令对`uart`进行后端流程的处理：

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

得到`uart`在asap7 工艺库下的版图为：



4 打包Dockerfile

本实验在docker 环境下完成， 最终，我们将docker 容器打包成镜像，并上传到docker hub， 以便于不同环境之间的快速移植。 具体 镜像请参见 leliyliu/iflow

在整个提交的报告中，包含了中间生成的结果，在文件夹`result` 中，并且，将所生成的gsd 文件全部打包到`gds` 文件夹中。