实验三　简单处理器（CPU）的版图设计实验

一、实验要求

1. 利用实验二产生的门级网表和 SDC 文件，参考示例设计，采用 EDA 版图设计工具进行版图设计，输出版图后门级电路。

2. 在生成的版图上进行时序参数提取；

3. 利用 PT 把 SPEF 文件变换成 SDF 格式；

4. 利用 VCS 和已有的 test 文件，反标 SDF，对版图后门级网表进行后仿真，并解决仿真过程中遇到的问题。

5. 请使用 AI 工具生成本实验的脚本，要求写出使用 AI 生成脚本的问题清单、优化方法和经验总结。

注：在 cpu 工作目录下使用 make 命令对 CPU 进行验证，并根据提示执行相应的命令运行 test 文件对 CPU 进行测试。

输入“ucli%>call test(1);run”运行 CPUtest1.dat 测试文件。

运行以上文件，应显示如下结果：

```
ucli% call test(1);run
RUNNING THE BASIC DIAGOSTIC TEST
THIS TEST SHOULD HALT WITH PC = 17
PC INSTR OP DATA ADR
-- ----- -- ---- ---
*Verdi* Loading libsscore_vcs202206.so
FSDB Dumper for VCS, Release Verdi_T-2022.06, Linux x86_64/64bit, 05/29/2022
(C) 1996 - 2022 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
*Verdi* : Create FSDB file 'cpu_test.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : Enable +all dumping.
*Verdi* : End of traversing.
xx HLT   0  zz  zz  00
00 JMP   7  zz  fe  00
1e JMP   7  zz  e3  1e
03 LDA   5  zz  ba  03
04 SKZ   1  zz  20  04
06 LDA   5  zz  bb  06
07 SKZ   1  zz  20  07
08 JMP   7  zz  ea  08
0a STO   6  zz  dc  0a
0b LDA   5  zz  ba  0b
0c STO   6  zz  dc  0c
0d LDA   5  zz  bc  0d
0e SKZ   1  zz  20  0e
10 XOR   4  zz  9b  10
11 SKZ   1  zz  20  11
12 JMP   7  zz  f4  12
14 XOR   4  zz  9b  14
15 SKZ   1  zz  20  15
17 HLT   0  zz  00  17
HALTED AT PC = 17

*************************************************************
*       THE FOLLOWING DEBUG TASKS ARE AVAILABLE:            *
* Enter "call test(1);run" to run the 1st diagnostic program.   *
* Enter "call test(2);run" to run the 2nd diagnostic program.   *
* Enter "call test(3);run" to run the Fibonacci program.        *
* Enter "call test(4);run" to run the COUNTER program.        *
* Enter "call test(5);run" to run the 2^n program.          *
*************************************************************

cpu_test.v, 69 :       $stop ;
```

输入“ucli%>call test(2);run”运行 CPUtest2.dat 测试文件。

运行以上文件，应显示如下结果：

```
ucli% call test(2);run
RUNNING THE ADVANCED DIAGOSTIC TEST
THIS TEST SHOULD HALT WITH PC = 10
PC INSTR OP DATA ADR
-- ----- -- ---- ---
00 LDA   5  zz  bb  00
01 AND   3  zz  7c  01
02 XOR   4  zz  9b  02
03 SKZ   1  zz  20  03
05 ADD   2  zz  5a  05
06 SKZ   1  zz  20  06
07 JMP   7  zz  e9  07
09 XOR   4  zz  9c  09
0a ADD   2  zz  5a  0a
0b STO   6  zz  dd  0b
0c LDA   5  zz  ba  0c
0d ADD   2  zz  5d  0d
0e SKZ   1  zz  20  0e
10 HLT   0  zz  00  10
HALTED AT PC = 10

************************************************************
*        THE FOLLOWING DEBUG TASKS ARE AVAILABLE:          *
* Enter "call test(1);run" to run the 1st diagnostic program.   *
* Enter "call test(2);run" to run the 2nd diagnostic program.   *
* Enter "call test(3);run" to run the Fibonacci program.        *
* Enter "call test(4);run" to run the COUNTER program.        *
* Enter "call test(5);run" to run the 2^n program.         *
************************************************************

cpu_test.v, 69 :       $stop ;
```

输入“ucli%>call test(3);run”运行 CPUtest3.dat 测试文件。

运行以上文件，应显示如下结果：

```
ucli% call test(3);run
RUNNING THE FIBONACCI CALCULATOR
THIS PROGRAM SHOULD CALCULATE TO 144
FIBONACCI NUMBER
 ---------------
  0
  1
  1
  2
  3
  5
  8
 13
 21
 34
 55
 89
144
HALTED AT PC = 0c

************************************************************
*        THE FOLLOWING DEBUG TASKS ARE AVAILABLE:          *
* Enter "call test(1);run" to run the 1st diagnostic program.   *
* Enter "call test(2);run" to run the 2nd diagnostic program.   *
* Enter "call test(3);run" to run the Fibonacci program.        *
* Enter "call test(4);run" to run the COUNTER program.        *
* Enter "call test(5);run" to run the 2^n program.         *
************************************************************

cpu_test.v, 69 :       $stop ;
```

二、注意事项

1. 版图设计采用的工艺库所在的路径和库名：

/home/eda/lib/smic/SP013D3_V1p4/syn

/home/eda/lib/smic/SP013D3_V1p4/syn/SP013D3_V1p2_typ.db

/home/eda/lib/smic/SP013D3_V1p4/apollo/SP013D3_V1p2_8MT

/home/eda/lib/smic/SmicSPM4PR8R_starRCXT013_mixrf_p1mtx_1233_V1/SmicSPM4PR8R_starRCXT013_mixrf_p1mtx_1233_V1.6/mapping/TM9k_MIM4f

/home/eda/lib/smic/SmicSPM4PR8R_starRCXT013_mixrf_p1mtx_1233_V1/SmicSPM4PR8R_starRCXT013_mixrf_p1mtx_1233_V1.6/ITF/TM9k_MIM4f/TLUPLUS

/home/eda/lib/smic/aci/sc-x/synopsys

/home/eda/lib/smic/aci/sc-x/synopsys/typical_1v2c25.db

/home/eda/lib/smic/aci/sc-x/apollo/smic13g

/home/eda/lib/smic/aci/sc-x/apollo/tf/smic13lvt_8lm.tf

2. 版图设计用的脚本文件的目录：/home/icctools

三、示例设计

本设计目标：对电路综合结果 control_pad 进行布局布线。

1. 初始化工作环境

（1）创建工作目录

在用户根目录下创建 ICC 目录 mkdir icc。

在 icc 目录下创建 design_data、run、logs、rm_setups、scripts、output 等目录。

各目录说明如下：

design_data 保存门级网表 control_pad.v 和 control_pad.sdc。

run:工作目录，在此目录下运行脚本，也在这个目录下创建库和设计 cell。

logs:用来存放脚本运行的日志文件。

rm_setup:工具和库的设置目录。

scripts:用于创建并保存脚本文件。

output:存放输出结果。

（2）初始化工作环境

从/home1/student/icc 目录中复制工具配置和基本变量设置脚本文件到相应工作目录。(所有脚本文件已放到桌面）

2. 数据准备

（1）在 scripts 目录下用 vi 编辑器创建数据准备的脚本文件 design_setup.tcl。

```
 # Design_setup.tcl
source ../rm_setup/lcrm_setup.tcl
source -echo ../rm_setup/icc_setup.tcl
##########################################################################
# Design Library Creation
##########################################################################
 create_mw_lib control_pad.mw -open -tech $TECH_FILE -mw_reference_library $MW_REFERENCE_LIB_DIRS

echo ###############end of creating of the mw lib ########################
##########################################################################
# Read the Netlist and Create a Design CEL
##########################################################################
if {$ICC_INIT_DESIGN_INPUT == "VERILOG" } {
 read_verilog -top $DESIGN_NAME  $ICC_INPUTS_PATH/$ICC_IN_VERILOG_NETLIST_FILE
 current_design $DESIGN_NAME
 uniquify
 save_mw_cel -as $DESIGN_NAME
}
set_tlu_plus_files                                                    -max_tluplus SmicSPM4PR8R_starRCXT013_log_mixRF_p1mt8_cell_max_1233_9k_1f.tluplus  -tech2itf_map SmicSPM4PR8R_013_log_mixRF_p1mt8_cell_max_1233_9k_1f.map
```

```
#########################################################################
#connect to supply nets
#########################################################################
derive_pg_connection -create_net; # first create P/G nets defined in UPF FILE
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS -ground_pin VSS ;
  # Connect P/G Pins to supply nets,
derive_pg_connection -power_net VDD -ground_net VSS -create_ports top -tie;
check_mv_design
#########################################################################
#Apply and Check Timing Constraints
#########################################################################
read_sdc $ICC_INPUTS_PATH/control_pad.sdc
check_timing
report_timing_requirements
report_disable_timing
report_case_analysis
#########################################################################
#Check the clocks
#########################################################################
report_clock
report_clock -skew
#########################################################################
#Timing and optimization controls
#########################################################################
source ../scripts/common_optimization_settings_icc.tcl
set_zero_interconnect_delay_mode true
report_constraint -all
report_timing
set_zero_interconnect_delay_mode false
remove_ideal_network [get_ports "rst_ clk"]
```

```
#####################################################################
#Save the cel after data setup
#####################################################################
save_mw_cel -as data_setup
```

（2）在 run 目录下创建数据准备的脚本文件 design_setup.tcl。

```
echo ok
if{[file exists [which design_setup.log]]}
rm ../logs/design_setup.log
icc_shell -64 -f ../scripts/design_setup.tcl | tee -i ../logs/design_setup.log
```

（3）数据准备执行，在 run 目录下启动 icc_shell,并运行数据准备相关步骤。

```
source design_setup.tcl
```

（4）查看 logs 目录下的 design_setup.log 文件，查看运行结果是否有运行错误?如果有，请问是什么错误，把错误写在下面并给出修改方法。

（5）修改错误后，删除 run 目录下已经创建的 control_pad.mw(注意每次重新运行 design_setup.tcl 之前均需要删除 control_pad.mw library，因为执行 design_setup.tcl 会再次创建该 library)，再次运行。直到错误被完全修改。查看 run 目录，应该生成了 control_pad.mw 库，库内包含 design_setup cell。在以下的实验中可直接读入 design_setup cell,在此基础上完成版图设计的下一任务——布图规划。

（6）简述 ICC 在数据准备阶段的主要工作。

3. 布图规划

（1）在 scripts 目录下创建布图规划的脚本文件 floorplan.tcl。

```
source ../rm_setup/lcrm_setup.tcl
source -echo ../rm_setup/icc_setup.tcl
open_mw_lib control_pad.mw
copy_mw_cel -from data_setup -to floorplan
open_mw_cel floorplan


########################################
#Create a Rectangular Block
```

```
########################################


create_cell {cornerll cornerlr cornerul cornerur}    PCORNER
create_cell {vss1left vss1right } PVSS1 ; # core ground
create_cell {vdd1left vdd1right } PVDD1;    # core supply
create_cell {vss2top vss2bottom} PVSS2;      # pad ground
create_cell {vdd2top vdd2bottom} PVDD2;      #pad_supply

# additional create_cell commands not shown
set_pad_physical_constraints -pad_name "cornerul" -side 1
set_pad_physical_constraints -pad_name "cornerur" -side 2
set_pad_physical_constraints -pad_name "cornerlr" -side 3
set_pad_physical_constraints -pad_name "cornerll" -side 4


#Place the pad

#Uper side
set_pad_physical_constraints -pad_name "i_opcode_2" -side 2 -order 1
set_pad_physical_constraints -pad_name "i_zero" -side 2 -order 2
set_pad_physical_constraints -pad_name "vdd2top" -side 2 -order 3
set_pad_physical_constraints -pad_name "vss2top" -side 2 -order 4
set_pad_physical_constraints -pad_name "i_ld_pc" -side 2 -order 5
#set_pad_physical_constraints -pad_name "i_inc_pc" -side 2 -order 6

#Right side rd, wr, ld_ir, ld_ac
set_pad_physical_constraints -pad_name "i_rd" -side 3 -order 1
set_pad_physical_constraints -pad_name "i_wr" -side 3 -order 2
set_pad_physical_constraints -pad_name "vdd1right" -side 3 -order 4
```

```
set_pad_physical_constraints -pad_name "vss1right" -side 3 -order 3
set_pad_physical_constraints -pad_name "i_ld_ir" -side 3 -order 5
set_pad_physical_constraints -pad_name "i_ld_ac" -side 3 -order 6
#Bottom side
set_pad_physical_constraints -pad_name "i_halt" -side 4 -order 1
set_pad_physical_constraints -pad_name "i_data_e" -side 4 -order 2
set_pad_physical_constraints -pad_name "vdd2bottom" -side 4 -order 3
set_pad_physical_constraints -pad_name "vss2bottom" -side 4 -order 4
set_pad_physical_constraints -pad_name "i_sel" -side 4 -order 5
set_pad_physical_constraints -pad_name "i_inc_pc" -side 4 -order 6


#Left side
set_pad_physical_constraints -pad_name "i_rst" -side 1 -order 1
set_pad_physical_constraints -pad_name "i_clk" -side 1 -order 2
set_pad_physical_constraints -pad_name "vdd1left" -side 1 -order 3
set_pad_physical_constraints -pad_name "vss1left" -side 1 -order 4
set_pad_physical_constraints -pad_name "i_opcode_0" -side 1 -order 5
set_pad_physical_constraints -pad_name "i_opcode_1" -side 1 -order 6


create_floorplan -control_type aspect_ratio -core_aspect_ratio 1 -core_utilization 0.7
-left_io2core 30 -bottom_io2core 30 -right_io2core 30 -top_io2core 30 -start_first_row
save_mw_cel -as floorplaned

##########################################
#Place Pins
##########################################
#create_fp_placement -timing
#route_zrt_global -congestion_map_only true \
```

```
                        -exploration true

save_mw_cel -as floorplanprepn

##########################################
# Insert Pad Fillers
##########################################
insert_pad_filler -cell " PFILL001 PFIIL01 PFILL1 PFILL10 PFILL2 PFILL20 PFILL5 PFILL50"

##########################################
#Specify Unrouting Layers
##########################################
set_ignored_layers -max_routing_layer METAL6
report_ignored_layers

##########################################
#Create the Power Network
##########################################

#Performing Power Planning
save_mw_cel -as floorplan_prepns
#Defining Logical Power and Ground Connections

#Applying Power Rail Constraints
#set_fp_rail_constraints -remove_all_layers
#set_fp_rail_constraints -add_layer -layer METAL7 -direction vertical -max_strap 20 -min_strap 10
-min_width 0.4 -spacing minimum
#set_fp_rail_constraints -add_layer -layer METAL8 -direction horizontal -max_strap 20
-min_strap 10 -min_width 0.4 -spacing minimum
```

```
#Applying Power Ring Constraints
create_power_plan_regions    core -core
#set_power_ring_strategy core -core -nets    {VDD VSS}

set_power_ring_strategy core -core -nets {VDD VSS}
-template ../scripts/basic_ring.tpl:basic_ring

#set_fp_block_ring_constraints -add -horizontal_layer METAL5 -vertical_layer METAL6
-horizontal_width 3 -vertical_width 3 -horizontal_offset 0.600 -vertical_offset 0.600 -block_type
plan_group -block core -nets {VDD VSS}
#compile_power_plan -ring
#compile_power_plan
#Synthesizing the Power Network
#synthesize_fp_rail -power_budget 800 -voltage_supply 1.32 -output_directory powerplan.dir
-nets {VDD VSS} -synthesize_power_plan
#synthesize_fp_rail -voltage_supply 1.32 -output_directory powerplan.dir -nets {VDD VSS}
-synthesize_power_plan



remove_power_plan_strategy -all
set_power_plan_strategy s_basic_no_va -nets {VDD VSS} -core -extension
{{{ nets:VDD}{stop:outermost_ring}} {{nets: VSS}{stop:outermost_ring}}}
-template ../scripts/pg_mesh.tpl:pg_mesh_top
compile_power_plan -ring
compile_power_plan

derive_pg_connection -create_net; # first create P/G nets defined in UPF FILE
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS -ground_pin VSS ; #
```

Connect P/G Pins to supply nets,

save_mw_cel -as floorplanafterpn

#vdd1left vdd1right vdd1top vdd1bottom

derive_pg_connection -power_net VDD -ground_net VSS -tie ; # Connect P/G Pins to supply nets,

##

#Route Standard Cell Rails

##

set_preroute_drc_strategy -min_layer METAL3 -max_layer METAL8

preroute_standard_cells -nets "VDD VSS" -remove_floating_pieces -do_not_route_over_macros ;

create_fp_placement -congestion -timing -no_hierarchy_gravity

route_zrt_global -congestion_map_only true -exploration true

preroute_instances

save_mw_cel -as floorplaned

create_fp_placement -congestion -timing -no_hierarchy_gravity

route_zrt_global -congestion_map_only true -exploration true

preroute_instances

（2）在 scripts 目录下创建 ring，生成约束模板 basic_ring.tpl。

```
template: basic_ring{
    side: horizontal {
            layer: METAL8
            width: 6 6
            spacing: 3
            offset : 0
            }
    side : vertical{

             layer: METAL7
            width: 6 6
            spacing: 3
            offset : 0
}
}
```

（3）在 scripts 目录下创建 mesh，生成约束模板 pg_mesh.tpl。

```
template : pg_mesh_top {
layer : METAL8 {
      direction : horizontal
      width : 5
      spacing : 30
      number :
      pitch : 42
      offset_start : boundary   # user can also specify coordinate as {x y}
      offset_type : edge   # user can also specify centerline
      offset :
      trim_strap : true
}
```

```
layer : METAL7 {
        direction : vertical
        width : 5
        spacing : 30
        number :
        pitch : 42
        offset_start : boundary   # user can also specify coordinate as {x y}
        offset_type : edge   # user can also specify centerline
        offset :
        trim_strap : true
}



advanced_rule : off {
}


}


template : pg_mesh_m2(p1, p2, p3, p4, p5) {
layer : M2 {
        direction : vertical
        width : {@p1 @p2 @p3}
        spacing : {@p4 @p5}
        number :
        pitch : 16
        offset_start : boundary   # user can also specify coordinate as {x y}
        offset_type : edge   # user can also specify centerline
        offset :
        trim_strap : true
}
```

```
  # Advanced rules for power plan creation
    advanced_rule : on { #all the advanced rules are turned off
    stack_vias : all #all | adjacent | specified
    optimize_routing_tracks : off {
        layer : all             #layers to perform optimization
        alignment : true    #align straps to track or half-track
        sizing : true         #further size up straps w/o blocking more tracks
    }
    insert_channel_straps: off {
        layer :                   #channel strap layer
        width : minimum      #channel strap width
        spacing : minimum #channel strap spacing
        channel_threshold: #ignore channels narrower than the threshold
        check_one_layer : false #only check straps in the specified layer
        boundary_strap : false #insert straps between region boundary
        honor_placement_blockage : true #honor channels between placement blocakges and
macros
        honor_voltage_area : false #honor channels between voltage areas and macros
        honor_keepout_margins : true #honor channels between hard keepout margins
    }
    honor_max_stdcell_strap_distance : off {
        max_distance :       #maximum distance from stdcell to strap
        layer :                  #additional strap layer
        offset :                 #offset to the nearest straps
    }
    align_straps_with_power_switch : off {
        power_switch :       #specify power switch name to align
        layer :                  #strap layer
        width :                   #strap width
```

```
        direction :             #strap direction
        offset :                #offset from the center of pin
    }
    align_straps_with_stdcell_rail: off {
        layer :                 #strap layer
        align_with_rail : false         #align strap with the rail
        put_strap_in_row : false        #put straps inside cell row
    }
    honor_advanced_via_rules : on
    align_straps_with_terminal_pin : off
    align_straps_with_physical_cell: off {
        layer :                 #strap layer
        cell :                  #lib cell to align
        pin :                   #pin to align
        direction :             #strap direction
        width :                 #strap width
        offset :                #offset from the center of pin
    }
    }
}
```

（4）在 run 目录下创建部图规划的运行脚本 floorplan.tcl。

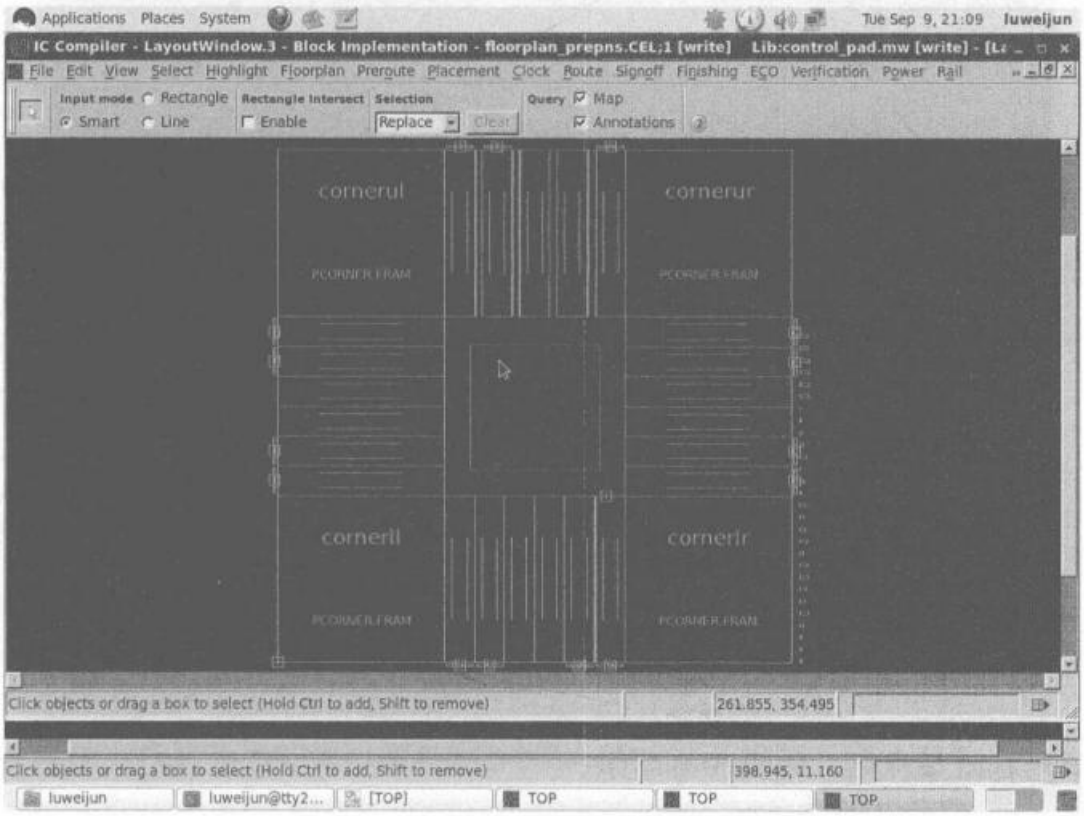
```
echo ok
if{ [file exists [which floorplan.log]]}
  rm ../logs/floorplan.log
icc_shell -64 -f ../scripts/floorplan.tcl | tee -i ../logs/floorplan.log
```

（5）布图规划执行，在 run 目录下启动 icc_shell，并运行 floorplan.tcl，操作如下：

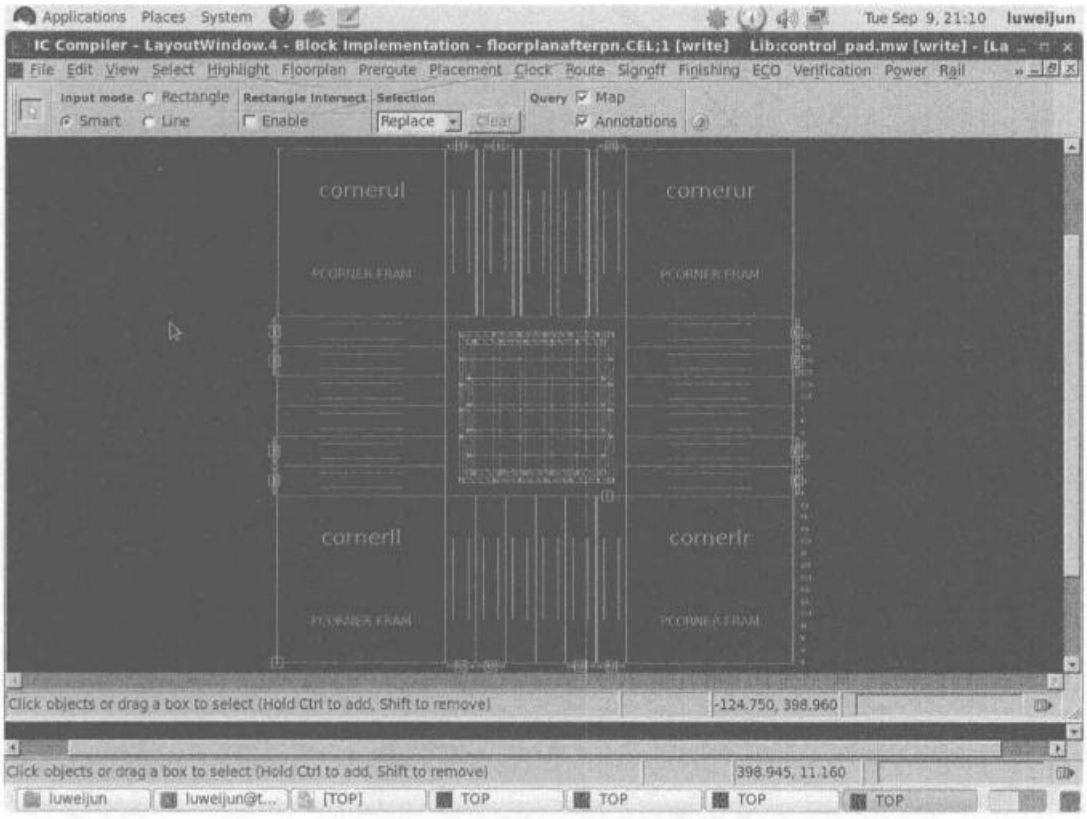
source floorplan.tcl

在图形界面下可打开设计的不同阶段

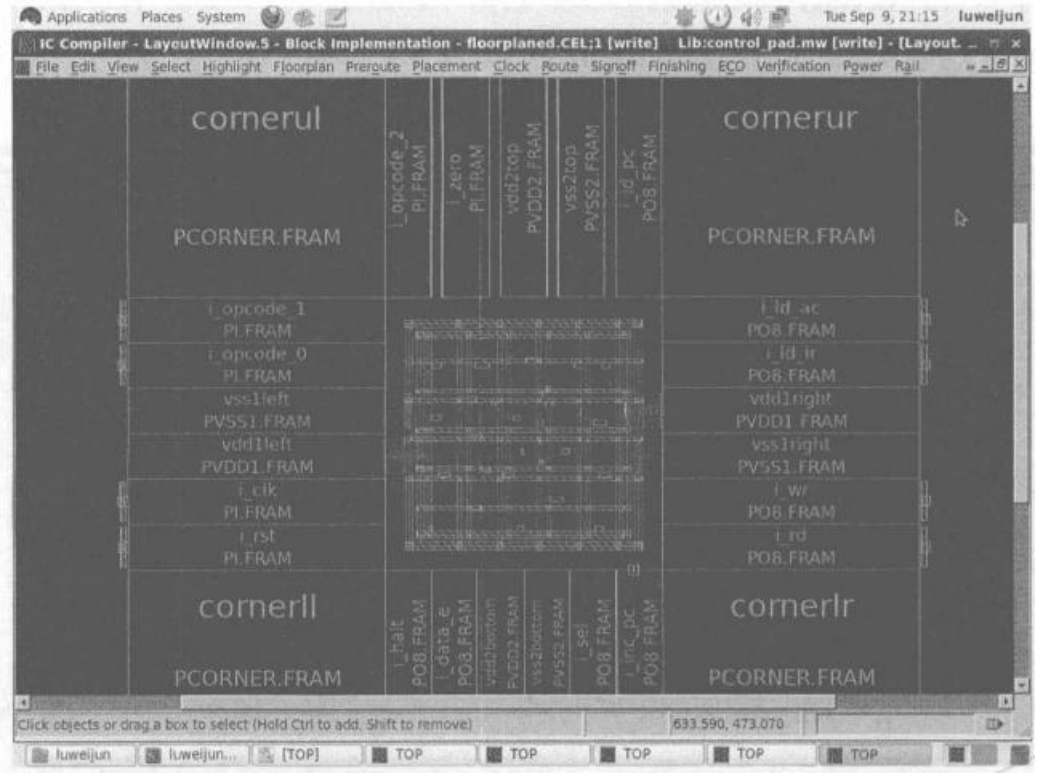
未产生 core ring 和 mesh 前的图形界面如图所示



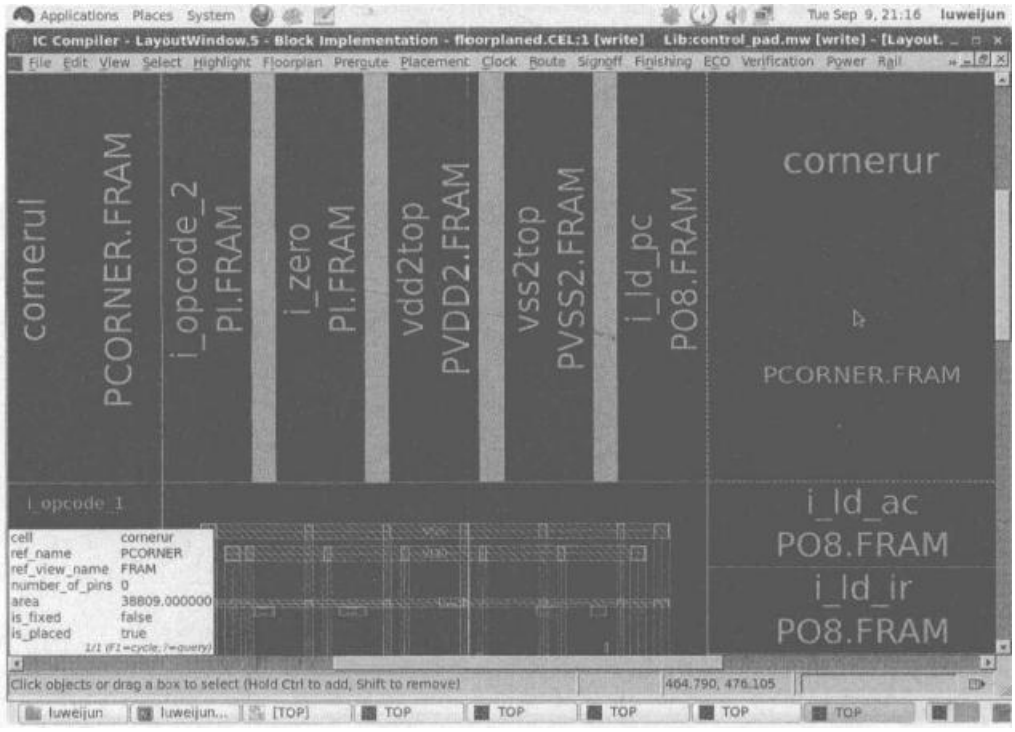
产生 core ring 和 mesh 后的图形界面如图所示



电源线和电源 pad 连接后的图形界面如图所示



标准填充单元 pad 填充后，如图所示



4. 布局

（1）在 scripts 目录下创建布局的脚本文件 place.tcl。

```
source ../rm_setup/lcrm_setup.tcl

source -echo ../rm_setup/icc_setup.tcl
```

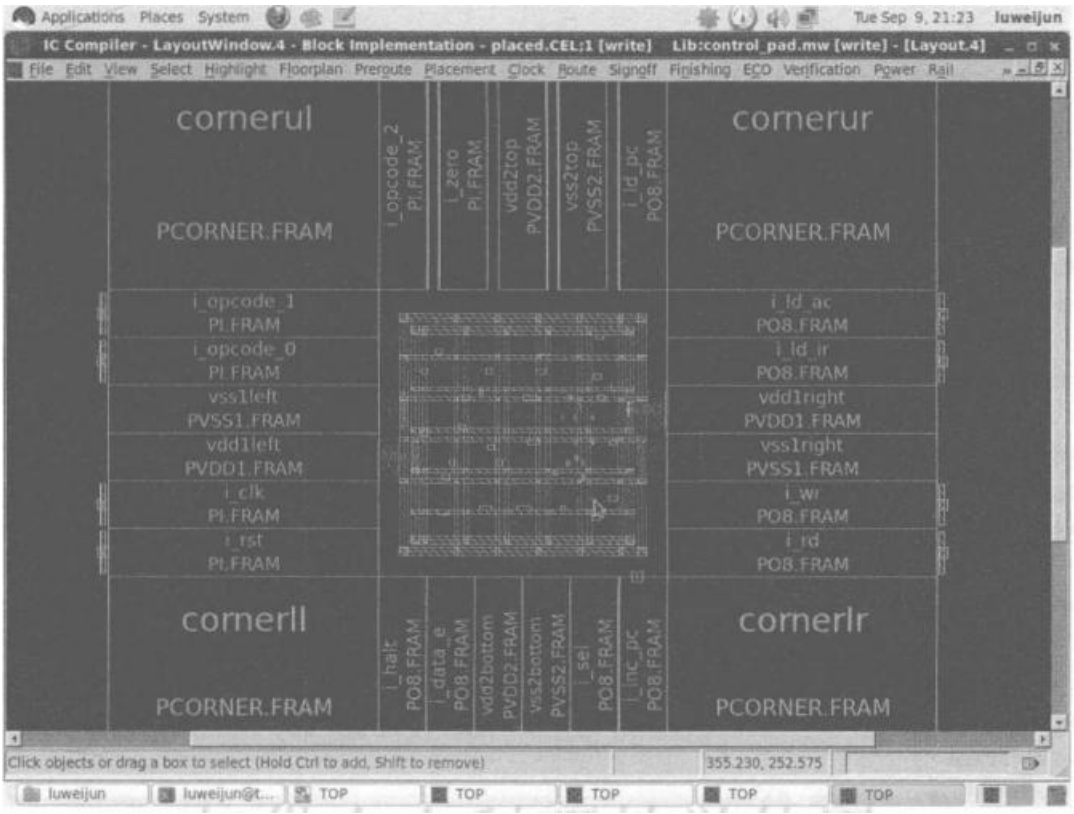
```
open_mw_lib control_pad.mw
copy_mw_cel -from floorplaned -to place
open_mw_cel place

#####################################################
source -echo ../scripts/common_optimization_settings_icc.tcl
source -echo ../scripts/common_placement_settings_icc.tcl
#####################################################
check_physical_design -stage pre_place_opt

set_ideal_network [all_fanout -flat -clock_tree]

place_opt    -area_recovery -congestion
psynopt -area_recovery    -congestion
refine_placement -congestion_effort high
psynopt -area_recovery    -congestion

create_qor_snapshot -name placed
query_qor_snapshot -name placed
save_mw_cel -as placed
```

（2）在 run 目录下创建布局的脚本文件 place.tcl。

```
echo ok
if{ [file exists [which place.log]]}
  rm ../logs/place.log
icc_shell -64 -f ../scripts/place.tcl | tee -i ../logs/place.log
```

（3）在 run 目录下运行 source place.tcl 并在图形界面查看布局的结果，查看 log 文件，确认没有执行错误。

布局后的图形界面如图所示



5. 时钟树综合

（1）在 scripts 目录下创建时钟树综合的脚本文件 cts.tcl。

```
source ../rm_setup/lcrm_setup.tcl

source -echo ../rm_setup/icc_setup.tcl

open_mw_lib control_pad.mw
copy_mw_cel -from place -to cts
open_mw_cel place

#####################################################
source -echo ../scripts/common_optimization_settings_icc.tcl
source -echo ../scripts/common_placement_settings_icc.tcl

## Source CTS Options
```

```
#source -echo common_cts_settings_icc.tcl

#########################################################
#Check The Design Before CTS
#########################################################
check_physical_design -stage pre_clock_opt
check_clock_tree

#########################################################
#Remove all ideal network settings on clocks
#########################################################
remove_ideal_network [get_ports clk]
remove_clock_uncertainty [all_clocks]
#set_delay_calculation_options -routed_clock arnoldi

#########################################################
#Defining CTS-Specific DRC Values
#########################################################

#The default values are to be used
#########################################################
#Specifying CTS Targets:Skew and Insertion Delay
#########################################################
set_clock_tree_option -target_early_delay 0.9
set_clock_tree_options -target_skew 0.2
report_clock_tree -settings

#########################################################
#Control Buffer/Inverter Selection
```

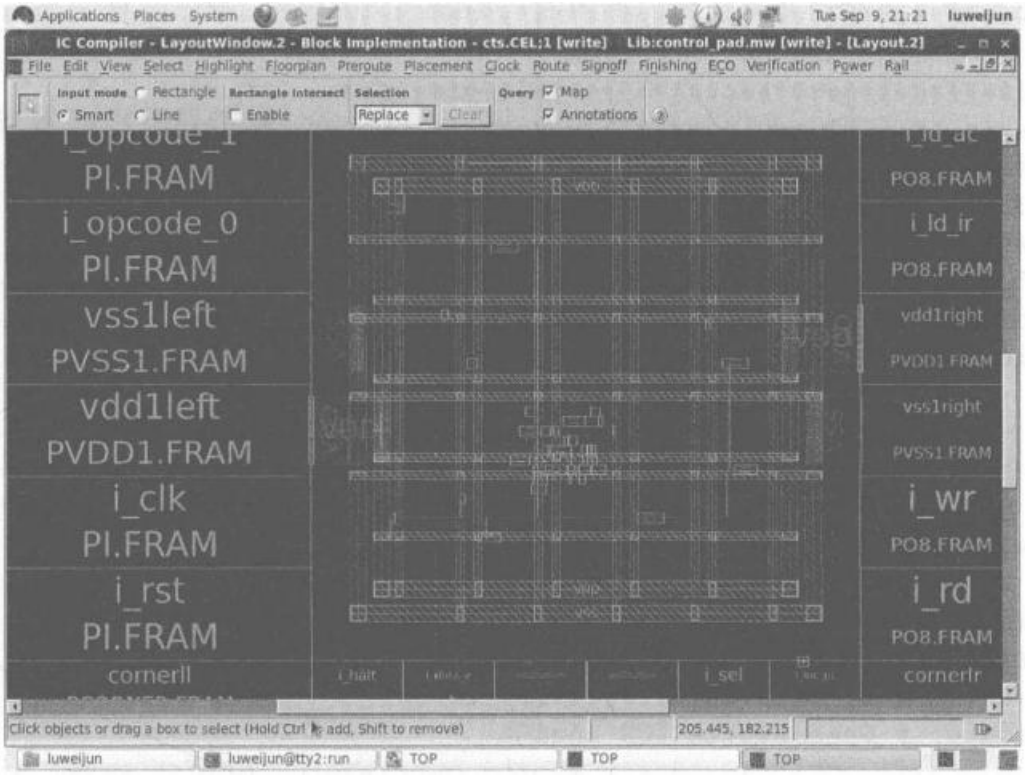
```
#####################################################

#If we dont define , all the Buffer/Inverter cells can be used

#####################################################
#CTS and Timing Optimization
#####################################################

#Clock tree synthesis
clock_opt -no_clock_route -only_cts

update_clock_latency

report_clock_tree
report_clock_timing -type skew


#Post CTS Logic Optimization
set_fix_hold [all_clocks]
clock_opt -no_clock_route -only_psyn
report_clock_tree
report_clock_timing -type skew

#Clock Tree Routing
set_fix_hold [all_clocks]
route_zrt_group -all_clock_nets -reuse_existing_global_route true
report_clock_tree
report_clock_timing -type skew

save_mw_cel -as cts
```

（2）在 run 目录下创建时钟树综合的运行脚本 cts.tcl，命令为 source cts.tcl。

```
echo ok
if{[file exists [which cts.log]]}
  rm ../logs/cts.log
icc_shell -64 -f ../scripts/cts.tcl | tee -i ../logs/cts.log
vcs -full64
```

（3）在图形界面下查看时钟树综合的结果，查看 log 文件，了解时钟树综合执行过程并确认没有运行错误。

时钟树综合后的图形界面如图所示，可以看到综合的时钟树。



6. 布线

（1）在 scripts 目录下创建布线的脚本文件 route.tcl。

```
source ../rm_setup/lcrm_setup.tcl
source -echo ../rm_setup/icc_setup.tcl

open_mw_lib control_pad.mw
copy_mw_cel -from cts -to route
open_mw_cel route
```

```
#####################################################
source -echo ../scripts/common_optimization_settings_icc.tcl
source -echo ../scripts/common_placement_settings_icc.tcl


#####################################################
#Pre-Routing Checks
#####################################################
check_physical_design -stage pre_route_opt
all_ideal_nets
all_high_fanout -nets -threshold 20
report_preferred_routing_direction

#####################################################
#Pre-Routing Setup
#####################################################

derive_pg_connection -power_net VDD -power_pin VDD \
                                -ground_net VSS -ground_pin VSS -tie; # Connect P/G Pins to supply
nets,



#####################################################
#Route Clock Nets Before Signal Nets
#####################################################
route_zrt_group -all_clock_nets -reuse_existing_global_route true
#####################################################
#Route the Signal Nets
```

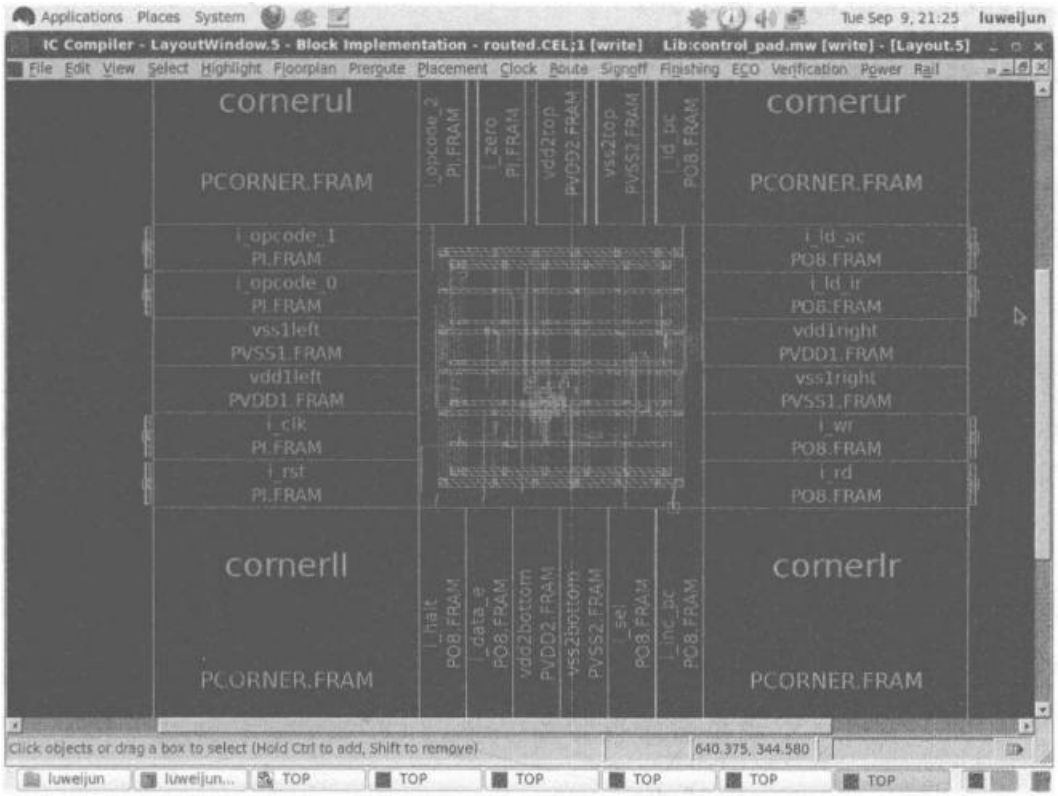
```
#####################################################
route_opt -initial_route_only

save_mw_cel -as signal_route
#####################################################
# Perform full post-route optimization
#####################################################
route_opt -skip_initial_route -xtalk_reduction -power

#####################################################
#Incremental Optimization
#####################################################
# Focus on hold time fixing only
set_fix_hold    clk
route_opt -incremental -only_hold_time

save_mw_cel -as routed
#Focus logical DRC violations
set_app_var routeopt_drc_over_timing true
route_opt -effort high -incremental -only_design_rule

#####################################################
#Check and Fix Physical DRC Violations
#####################################################
verify_zrt_route ; # Uses Zroute DRC engine
set_route_zrt_detail_options -repair_shorts_over_macros_effort_level high
route_zrt_detail -incremental true ; # Fix DRCs

#####################################################
#Debug Opens
```

```
#####################################################
#verify_lvs -ignore_short -ignore_min_area

#####################################################
#Sign off DRC Checking and Fixing
#####################################################



save_mw_cel -as route

#################################################
#Write netlist and parasitics
#################################################
change_names -hierarchy    -rules verilog
write_verilog -no_physical_only_cells \
                    -no_unconnected_cells \
                    -no_tap_cells              \
                    ../output/control_pad_final.v

extract_rc -coupling_cap
write_parasitics    -output ../output/control_pad.spef \
                            -format SPEF                \
                            -compress                   \
                            -no_name_mapping
```

（2）在 run 目录下创建布线的执行脚本 route.tcl，命令为 source route.tcl。

```
echo ok
if{[file exists [which route.log]]}
  rm ../logs/route.log
icc_shell -64 -f ../scripts/route.tcl | tee -i ../logs/route.log
vcs -full64
```

（3）执行 route.tcl，布线后的运行结果如图所示。



（4）当前设计是否存在 DRC 违规和时序错误？如果有，请对其进行修正，直到 DRC 违规和时序违规为 0。

7. 寄生参数的导出和后仿真

（1）利用以下命令导出布线的网表和寄生参数（SPEF 格式）。

```
open_mw_lib control_pad.mw
copy_mw_cel -from route -to spef
open_mw_cel spef
change_names -hierarchy -rules verilog
write_verilog -no_physical_only_cells -no_unconnected_cells
-no_tap_cells ../output/control_pad_final.v
extract_rc -coupling_cap
write_parasitics -output ./output/control_pad.spef -format SPEF -no_name_mapping
```

（2）SDF 延迟信息生成。

PT 读入 SPEF 网表，还有 control_pad.final.v 网表和 cell library，产生 SDF 文件。反标 SDF 文件后，对输出的网表和.v 库文件进行后仿真。编写如下脚本文件：

```
#sdf_gen.tcl
set link_library "/homel/lib/smic/SP013D3_Vlp4/syn/SP013D3_V1p2_typ.db
/homel/lib/smic/aci/sc-x/synopsys/typical_1v2c25.db"
read_verilog../output/control_pad_final.v
current_design control_pad
read_db /homel/lib/smic/aci/sc-x/synopsys/typical_1v2c25.db
read_db /homel/lib/smic/SP013D3_V1p4/syn/SP013D3_Vlp2_typ.db
read_parasitics -pin_cap_included ../output/control_pad.spef
write_sdf ../output/control_pad.sdf
```

在 run 目录下执行如下操作：

1) 启动 pt_shell；
2) 在 pt_shell 命令行下输入 source sdf_gen.tcl。

（3）布局布线后仿真和综合后仿真一样，也要反标 control_pad.sdf。

在仿真中加入 smic13g.v 以及 SP013D3_V1p2.v 库文件进行仿真。

具体命令为：

```
Vcs-full64-v /homel/lib/smic/aci/sc-x/verilog/smic13g.v /homel/lib/smic/SP013D3_V1p4/
verilog/SP013D3_Vlp2.vcontrol_test.v control_pad_final.v +max delays -R
```

其中，control_test.v 为之前使用控制器仿真使用的测试文件，control_pad.v 为之前保持的网表文件。如果之前的设计操作都是正确的，测试结果应当显示 TEST PASSED。如果有错误，则需要返回仔细查找。