CAD Design Project 5 – OpenROAD (Final Project)

我負責針對 Placement 的步驟來做細部的分析

Placement 之 Makefile 部分

```
#  ____  _                    _   ____ _____
# |  _ \| |      /\          / \ / ___| ____|
# | |_) | |     / _ \ _____ / _ \ |   |  _|
# |  __/| |___ / ___ \_____/ ___ \ |___| |___
# |_|   |_____/_/   \_\   /_/   \_\____|_____|
#
.PHONY: place
place: $(RESULTS_DIR)/3_place.odb \
       $(RESULTS_DIR)/3_place.sdc
# ==========================================================================
# STEP 1: Global placement without placed IOs, timing-driven, and routability-driven.
#--------------------------------------------------------------------------
$(eval $(call do-step,3_1_place_gp_skip_io,$(RESULTS_DIR)/2_floorplan.odb $(RESULTS_DIR)/2_floorplan.sdc $(LIB_FILES),global_place_skip_io))

# STEP 2: IO placement (non-random)
#--------------------------------------------------------------------------
ifndef IS_CHIP
$(eval $(call do-step,3_2_place_iop,$(RESULTS_DIR)/3_1_place_gp_skip_io.odb $(IO_CONSTRAINTS),io_placement))
else
$(eval $(call do-copy,3_2_place_iop,3_1_place_gp_skip_io.odb,$(IO_CONSTRAINTS)))
endif

# STEP 3: Global placement with placed IOs, timing-driven, and routability-driven.
#--------------------------------------------------------------------------
$(eval $(call do-step,3_3_place_gp,$(RESULTS_DIR)/3_2_place_iop.odb $(RESULTS_DIR)/2_floorplan.sdc $(LIB_FILES),global_place))

# STEP 4: Resizing & Buffering
#--------------------------------------------------------------------------
$(eval $(call do-step,3_4_place_resized,$(RESULTS_DIR)/3_3_place_gp.odb $(RESULTS_DIR)/2_floorplan.sdc,resize))

.PHONY: clean_resize
clean_resize:
	rm -f $(RESULTS_DIR)/3_4_place_resized.odb

# STEP 5: Detail placement
#--------------------------------------------------------------------------
$(eval $(call do-step,3_5_place_dp,$(RESULTS_DIR)/3_4_place_resized.odb,detail_place))

$(eval $(call do-copy,3_place,3_5_place_dp.odb,))

$(eval $(call do-copy,3_place,2_floorplan.sdc,,.sdc))

.PHONY: do-place
do-place:
	$(UNSET_AND_MAKE) do-3_1_place_gp_skip_io do-3_2_place_iop do-3_3_place_gp do-3_4_place_resized do-3_5_place_dp do-3_place do-3_place.sdc

# Clean Targets
#--------------------------------------------------------------------------
.PHONY: clean_place
clean_place:
	rm -f $(RESULTS_DIR)/3_*place*.odb
	rm -f $(RESULTS_DIR)/3_place.sdc
	rm -f $(RESULTS_DIR)/3_*.def $(RESULTS_DIR)/3_*.v
```

Step1 Global placement without placed IOs, timing-driven, and routability-driven:

Source code:

```
$(eval $(call do-step,3_1_place_gp_skip_io,$(RESULTS_DIR)/2_floorplan.odb $(RESULTS_DIR)/2_floorplan.sdc $(LIB_FILES),global_place_skip_io))
```

這一段程式碼主要執行名為 global_place_skip_io 的腳本,

Input 為 2_floorplan.odb, 2_floorplan.sdc 這兩個就是上一步驟 Floorplan 的文件

指定輸出為 3_1_place_gp_skip_io.odb 也就是運行後的結果

過程中的運行時間或是錯誤信息等會記錄在 3_1_place_gp_skip_io.log

而這個腳本為下，逐步拆解腳本

```
source $::env(SCRIPTS_DIR)/load.tcl
load_design 2_floorplan.odb 2_floorplan.sdc


if { [info exists ::env(FLOORPLAN_DEF)] } {
  puts "FLOORPLAN_DEF is set. Skipping global placement without IOs"
} else {
  source $::env(SCRIPTS_DIR)/set_place_density.tcl

  if { 0 != [llength [array get ::env GLOBAL_PLACEMENT_ARGS]] } {
  global_placement -skip_io -density $place_density \
      -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
      -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
      {*}$::env(GLOBAL_PLACEMENT_ARGS)
  } else {
  global_placement -skip_io -density $place_density \
      -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
      -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT)
  }
}

write_db $::env(RESULTS_DIR)/3_1_place_gp_skip_io.odb
```

腳本步驟為

1. 載入 floorplan 的結果也就是 2_floorplan.odb, 2_floorplan.sdc

source $::env(SCRIPTS_DIR)/load.tcl //設定執行所需的環境變數

load_design 2_floorplan.odb 2_floorplan.sdc

2. 檢查是否跳過 I/O 區域的全域佈局

if { [info exists ::env(FLOORPLAN_DEF)] } {

  puts "FLOORPLAN_DEF is set. Skipping global placement without IOs"

}

如果環境變數 FLOORPLAN_DEF 已被設置，則跳過這個步驟，並輸出訊息

表示跳過。

3. 設置元件的佈局密度

source $::env(SCRIPTS_DIR)/set_place_density.tcl

用於計算和設置 place_density。

4. 執行全域佈局（跳過 I/O 區域）

如果存在 GLOBAL_PLACEMENT_ARGS

```
global_placement -skip_io -density $place_density \
    -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
    -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
    {*}$::env(GLOBAL_PLACEMENT_ARGS)
```

如果沒有額外參數

```
global_placement -skip_io -density $place_density \
    -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
    -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT)
```

- ✓ -skip_io：跳過對 I/O 區域的佈局。
- ✓ -density：設定元件密度。
- ✓ -pad_left 和 -pad_right：設定佈局時元件與邊界的間距。
- ✓ GLOBAL_PLACEMENT_ARGS：可以提供額外的參數來自訂行為。

5. 輸出結果

write_db $::env(RESULTS_DIR)/3_1_place_gp_skip_io.odb

也就是將執行的結果存入 3_1_place_gp_skip_io.odb

Step2 IO placement (non-random):

Source code:

```
ifndef IS_CHIP
$(eval $(call do-step,3_2_place_iop,$(RESULTS_DIR)/3_1_place_gp_skip_io.odb $(IO_CONSTRAINTS),io_placement))
else
$(eval $(call do-copy,3_2_place_iop,3_1_place_gp_skip_io.odb,$(IO_CONSTRAINTS)))
endif
```

這裡是在處理 IO pins 的放置階段，主要有兩種情況：

1. 當 IS_CHIP 未定義時 (ifndef IS_CHIP):

$(eval $(call do-step, 3_2_place_iop,

$(RESULTS_DIR)/3_1_place_gp_skip_io.odb

$(IO_CONSTRAINTS),io_placement))

執行 io_placement.tcl 腳本進行 IO pins 的放置

Input 是 3_1_place_gp_skip_io.odb

需要考慮 IO_CONSTRAINTS (IO 限制條件)產生輸出檔 3_2_place_iop.odb

2. 當 IS_CHIP 已定義時 (else):

$(eval $(call do-copy, 3_2_place_iop,3_1_place_gp_skip_io.odb,

$(IO_CONSTRAINTS)))

只是複製 file，不執行實際的 IO 放置

將 3_1_place_gp_skip_io.odb 複製為 3_2_place_iop.odb

主要區別是：

✓ 非晶片設計時需要執行 IO pin 的實際放置

✓ 晶片設計時 IO pins 位置可能已經預先定義好，所以只需複製檔案

這是 IS_CHIP 定義

```
ifneq ($(FOOTPRINT),)
IS_CHIP = 1
else ifneq ($(FOOTPRINT_TCL),)
IS_CHIP = 1
endif
```

如果 FOOTPRINT 變數有被定義(不為空),則設定 IS_CHIP = 1

或者如果 FOOTPRINT_TCL 變數有被定義,也設定 IS_CHIP = 1

這表示當我們有預定義的晶片 footprint 或相關的 TCL 腳本時,就會將其視為

完整晶片設計(IS_CHIP = 1)

這個設定會影響後續的 IO pin 放置策略

如果是晶片設計(IS_CHIP = 1): IO pins 位置已預先定義,只需複製檔案

如果不是則需要執行 IO pins 的實際放置流程

io_placement.tcl 的部分

```
1  source $::env(SCRIPTS_DIR)/load.tcl
2  load_design 3_1_place_gp_skip_io.odb 2_floorplan.sdc
3
4  source $::env(SCRIPTS_DIR)/io_placement_util.tcl
5
6  write_db $::env(RESULTS_DIR)/3_2_place_iop.odb
```

source $::env(SCRIPTS_DIR)/load.tcl //載入基本的設定和工具腳本

load_design 3_1_place_gp_skip_io.odb 2_floorplan.sdc

//載入前一個步驟產生的設計檔案(3_1_place_gp_skip_io.odb)

同時載入時序限制檔案(2_floorplan.sdc)

source $::env(SCRIPTS_DIR)/io_placement_util.tcl

載入專門用來處理 IO pin 放置的工具腳本

以下是 io_placement_util.tcl 的內容與解釋

1. 檢查是否存在預定義的 floorplan

if {[info exists ::env(FLOORPLAN_DEF)]} {

puts "Skipping IO placement as DEF file was used to initialize

floorplan."

}

先檢查是否已有 FLOORPLAN_DEF 檔案，如果有就會跳過 IO 放置過程，

因為 floorplan 已經被初始化了。

2. 載入 IO 限制條件

else {

if {[info exists ::env(IO_CONSTRAINTS)]} {

source $::env(IO_CONSTRAINTS)

}

檢查是否有 IO_CONSTRAINTS 設定，如果有就載入這些限制條件。

3. 設定 pin 放置的參數

```
set args [list -hor_layer $::env(IO_PLACER_H) \
               -ver_layer $::env(IO_PLACER_V) \
               {*}$::env(PLACE_PINS_ARGS)]
```

- hor_layer: 指定水平方向的金屬層
- ver_layer: 指定垂直方向的金屬層
- PLACE_PINS_ARGS: 其他額外的 pin 放置參數

4. 執行 pin 放置命令並記錄

```
log_cmd place_pins {*}$args
```

使用 log_cmd 執行並記錄 place_pins 命令

write_db $::env(RESULTS_DIR)/3_2_place_iop.odb

將處理完的結果存檔，產生新的設計檔案(3_2_place_iop.odb)

Step3 Global placement with placed IOs, timing-driven, and routability-driven:

Source code:

```
$(eval $(call do-step,3_3_place_gp,$(RESULTS_DIR)/3_2_place_iop.odb $(RESULTS_DIR)/2_floorplan.sdc $(LIB_FILES),global_place))
```

呼叫一個定義好的 do-step 函數，執行 global_place.tcl 這個腳本

$(RESULTS_DIR)/3_2_place_iop.odb：前一步驟(IO placement)的輸出檔案

$(RESULTS_DIR)/2_floorplan.sdc：時序限制檔案

以下是其腳本內容

```
utl::set_metrics_stage "globalplace__{}"
source $::env(SCRIPTS_DIR)/load.tcl
load_design 3_2_place_iop.odb 2_floorplan.sdc

set_dont_use $::env(DONT_USE_CELLS)

# set fastroute layer reduction
if {[info exist env(FASTROUTE_TCL)]} {
  source $env(FASTROUTE_TCL)
} else {
  set_global_routing_layer_adjustment $env(MIN_ROUTING_LAYER)-$env(MAX_ROUTING_LAYER) 0.5
  set_routing_layers -signal $env(MIN_ROUTING_LAYER)-$env(MAX_ROUTING_LAYER)
  if {[info exist env(MACRO_EXTENSION)]} {
    set_macro_extension $env(MACRO_EXTENSION)
  }
}

source $::env(SCRIPTS_DIR)/set_place_density.tcl

set global_placement_args {}

# Parameters for routability mode in global placement
if {$::env(GPL_ROUTABILITY_DRIVEN)} {
  lappend global_placement_args {-routability_driven}
}

# Parameters for timing driven mode in global placement
if {$::env(GPL_TIMING_DRIVEN)} {
  lappend global_placement_args {-timing_driven}
}

proc do_placement {place_density global_placement_args} {
  set all_args [concat [list -density $place_density \
    -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
    -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT)] \
    $global_placement_args]

  if { 0 != [llength [array get ::env GLOBAL_PLACEMENT_ARGS]] } {
    lappend all_args {*}$::env(GLOBAL_PLACEMENT_ARGS)
  }

  log_cmd global_placement {*}$all_args
}

set result [catch {do_placement $place_density $global_placement_args} errMsg]
if {$result != 0} {
  write_db $::env(RESULTS_DIR)/3_3_place_gp-failed.odb
  error $errMsg
}

estimate_parasitics -placement

if {[info exist ::env(CLUSTER_FLOPS)]} {
  cluster_flops
  estimate_parasitics -placement
}

report_metrics 5 "global place" false false

write_db $::env(RESULTS_DIR)/3_3_place_gp.odb
```

utl::set_metrics_stage "globalplace__{}"

source $::env(SCRIPTS_DIR)/load.tcl

load_design 3_2_place_iop.odb 2_floorplan.sdc

set_dont_use $::env(DONT_USE_CELLS)

✓ 設定 metrics 階段名稱

- ✓ 載入前一步驟的設計檔和時序限制
- ✓ 設定不使用的單元列表

```
# set fastroute layer reduction
if {[info exist env(FASTROUTE_TCL)]} {
   source $env(FASTROUTE_TCL)
} else {
   set_global_routing_layer_adjustment $env(MIN_ROUTING_LAYER)-
$env(MAX_ROUTING_LAYER) 0.5
   set_routing_layers -signal $env(MIN_ROUTING_LAYER)-
$env(MAX_ROUTING_LAYER)
   if {[info exist env(MACRO_EXTENSION)]} {
      set_macro_extension $env(MACRO_EXTENSION)
   }
}
```

如果有特定的 FastRoute 設定檔就載入，否則使用預設的佈線層級設定

而 FastRoute 文件可以去 Openroad 的網站看

FastRoute

Min Pan, Yue Xu, Yanheng Zhang, Chris Chu

Contacts: yuexu@iastate.edu

Introduction

FastRoute is a global routing tool for VLSI back-end design. It is based on sequential rip-up and re-route (RRR) and a lot of novel techniques. FastRoute 1.0 first uses FLUTE to construct congestion-driven Steiner trees, which will later undergo the edge shifting process to optimize tree structure to reduce congestion. It then uses pattern routing and maze routing with logistic function based cost function to solve the congestion problem. FastRoute 2.0 proposed monotonic routing and multi-source multi-sink maze routing techniques to enhance the capability to reduce congestion. FastRoute 3.0 introduced the virtual capacity technique to adaptively change the capacity associated with each global edge to divert wire usage from highly congested regions to less congested regions. FastRoute 4.0 proposed via-aware Steiner tree, 3-bend routing and a delicate layer assignment algorithm to effectively reduce via count while maintaining outstanding congestion reduction capability. FastRoute 4.1 simplifies the way the virtual capacities are updated and applies a single set of tuning parameters to all benchmark circuits.

source $::env(SCRIPTS_DIR)/set_place_density.tcl

載入佈局密度設定腳本

```
# check the lower boundary of the PLACE_DENSITY and add PLACE_DENSITY_LB_ADDON if it exists
if {[info exist ::env(PLACE_DENSITY_LB_ADDON)]} {
  set place_density_lb [gpl::get_global_placement_uniform_density \
  -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
  -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT)]
  set place_density [expr $place_density_lb + ((1.0 - $place_density_lb) * $::env(PLACE_DENSITY_LB_ADDON)) + 0.01]
  if {$place_density > 1.0} {
    utl::error FLW 24 "Place density exceeds 1.0 (current PLACE_DENSITY_LB_ADDON = $::env(PLACE_DENSITY_LB_ADDON)). Please check if the value of PLACE_DENSITY_LB_ADDON is between 0 and 0.99."
  }
} else {
  set place_density $::env(PLACE_DENSITY)
}
```

和 global_place_skip_io 一樣的 place_density 設定

set global_placement_args {} //加入佈線驅動模式

Parameters for routability mode in global placement

if {$::env(GPL_ROUTABILITY_DRIVEN)} {

  lappend global_placement_args {-routability_driven}

} //加入時序驅動模式

Parameters for timing driven mode in global placement

if {$::env(GPL_TIMING_DRIVEN)} {

```
    lappend global_placement_args {-timing_driven}
}
//定義布局函數
proc do_placement {place_density global_placement_args} {
    set all_args [concat [list -density $place_density \
        -pad_left $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT) \
        -pad_right $::env(CELL_PAD_IN_SITES_GLOBAL_PLACEMENT)] \
        $global_placement_args]
//檢查是否有額外的全域佈局參數
    if { 0 != [llength [array get ::env GLOBAL_PLACEMENT_ARGS]] } {
        lappend all_args {*}$::env(GLOBAL_PLACEMENT_ARGS)
    }
    log_cmd global_placement {*}$all_args
}
如果失敗，儲存目前狀態並報錯
set result [catch {do_placement $place_density $global_placement_args}
errMsg]
if {$result != 0} {
    write_db $::env(RESULTS_DIR)/3_3_place_gp-failed.odb
```

```
    error $errMsg
}
```

後處理，估算寄生效應，儲存 result

```
estimate_parasitics -placement
if {[info exist ::env(CLUSTER_FLOPS)]} {
    cluster_flops
    estimate_parasitics -placement
}
report_metrics 5 "global place" false false
write_db $::env(RESULTS_DIR)/3_3_place_gp.odb
```

Step4 Resizing & Buffering:

```
$(eval $(call do-step,3_4_place_resized,$(RESULTS_DIR)/3_3_place_gp.odb $(RESULTS_DIR)/2_floorplan.sdc,resize))

.PHONY: clean_resize
clean_resize:
        rm -f $(RESULTS_DIR)/3_4_place_resized.odb
```

呼叫 do-step 來執行 resize 步驟

$(RESULTS_DIR)/3_3_place_gp.odb：前一步驟的全域佈局結果

$(RESULTS_DIR)/2_floorplan.sdc：時序限制檔案

resize 是腳本名稱以下是其腳本

```
utl::set_metrics_stage "placeopt__{}"
source $::env(SCRIPTS_DIR)/load.tcl
load_design 3_3_place_gp.odb 2_floorplan.sdc

estimate_parasitics -placement

set instance_count_before [sta::network_leaf_instance_count]
set pin_count_before [sta::network_leaf_pin_count]

set_dont_use $::env(DONT_USE_CELLS)

# Do not buffer chip-level designs
# by default, IO ports will be buffered
# to not buffer IO ports, set environment variable
# DONT_BUFFER_PORT = 1
if { ![info exists ::env(FOOTPRINT)] } {
  if { ![info exists ::env(DONT_BUFFER_PORTS)] || $::env(DONT_BUFFER_PORTS) == 0 } {
    puts "Perform port buffering..."
    buffer_ports
  }
}

puts "Perform buffer insertion..."
set additional_args ""
if { [info exists ::env(CAP_MARGIN)] && $::env(CAP_MARGIN) > 0.0} {
  puts "Cap margin $::env(CAP_MARGIN)"
  append additional_args " -cap_margin $::env(CAP_MARGIN)"
}
if { [info exists ::env(SLEW_MARGIN)] && $::env(SLEW_MARGIN) > 0.0} {
  puts "Slew margin $::env(SLEW_MARGIN)"
  append additional_args " -slew_margin $::env(SLEW_MARGIN)"
}

repair_design {*}$additional_args

if { [info exists env(TIE_SEPARATION)] } {
  set tie_separation $env(TIE_SEPARATION)
} else {
  set tie_separation 0
}

# Repair tie lo fanout
puts "Repair tie lo fanout..."
set tielo_cell_name [lindex $env(TIELO_CELL_AND_PORT) 0]
set tielo_lib_name [get_name [get_property [lindex [get_lib_cell $tielo_cell_name] 0] library]]
set tielo_pin $tielo_lib_name/$tielo_cell_name/[lindex $env(TIELO_CELL_AND_PORT) 1]
repair_tie_fanout -separation $tie_separation $tielo_pin

# Repair tie hi fanout
puts "Repair tie hi fanout..."
set tiehi_cell_name [lindex $env(TIEHI_CELL_AND_PORT) 0]
set tiehi_lib_name [get_name [get_property [lindex [get_lib_cell $tiehi_cell_name] 0] library]]
set tiehi_pin $tiehi_lib_name/$tiehi_cell_name/[lindex $env(TIEHI_CELL_AND_PORT) 1]
repair_tie_fanout -separation $tie_separation $tiehi_pin

# hold violations are not repaired until after CTS

# post report

puts "Floating nets: "
report_floating_nets

report_metrics 3 "resizer" true false

puts "Instance count before $instance_count_before, after [sta::network_leaf_instance_count]"
puts "Pin count before $pin_count_before, after [sta::network_leaf_pin_count]"

write_db $::env(RESULTS_DIR)/3_4_place_resized.odb
```

```
utl::set_metrics_stage "placeopt__{}"

source $::env(SCRIPTS_DIR)/load.tcl

load_design 3_3_place_gp.odb 2_floorplan.sdc Step5 Detail placement:

設置指標階段名稱與載入腳本和設計檔案

estimate_parasitics -placement //估算電路的寄生電容

set instance_count_before [sta::network_leaf_instance_count]

set pin_count_before [sta::network_leaf_pin_count]

//記錄優化前的實例數量和接腳數量

set_dont_use $::env(DONT_USE_CELLS)

設置在優化過程中不應該使用的標準單元

# Do not buffer chip-level designs

# by default, IO ports will be buffered

# to not buffer IO ports, set environment variable

# DONT_BUFFER_PORT = 1

if { ![info exists ::env(FOOTPRINT)] } {

   if { ![info exists ::env(DONT_BUFFER_PORTS)] ||

$::env(DONT_BUFFER_PORTS) == 0 } {

      puts "Perform port buffering..."

      buffer_ports
```

```
    }
}
```

只在非 chip level 設計時執行

如果沒有設置 DONT_BUFFER_PORTS 或其值為 0，則執行接口緩衝

以下是 buffer_ports 的用法

Buffer Ports

The `buffer_ports -inputs` command adds a buffer between the input and its loads. The `buffer_ports -outputs` adds a buffer between the port driver and the output port. Inserting buffers on input and output ports makes the block input capacitances and output drives independent of the block internals.

```
buffer_ports
    [-inputs]
    [-outputs]
    [-max_utilization util]
    [-buffer_cell buf_cell]
```

```
puts "Perform buffer insertion..."

set additional_args ""

if { [info exists ::env(CAP_MARGIN)] && $::env(CAP_MARGIN) > 0.0} {
    puts "Cap margin $::env(CAP_MARGIN)"
    append additional_args " -cap_margin $::env(CAP_MARGIN)"
}

if { [info exists ::env(SLEW_MARGIN)] && $::env(SLEW_MARGIN) > 0.0} {
    puts "Slew margin $::env(SLEW_MARGIN)"
    append additional_args " -slew_margin $::env(SLEW_MARGIN)"
}
```

repair_design {*}$additional_args

可設置 CAP_MARGIN,SLEW_MARGIN

以下是 repair_design 用法

Repair Design

The `repair_design` command inserts buffers on nets to repair max slew, max capacitance and max fanout violations, and on long wires to reduce RC delay in the wire. It also resizes gates to normalize slews. Use `estimate_parasitics -placement` before `repair_design` to estimate parasitics considered during repair. Placement-based parasitics cannot accurately predict routed parasitics, so a margin can be used to "over-repair" the design to compensate.

```
repair_design
    [-max_wire_length max_length]
    [-slew_margin slew_margin]
    [-cap_margin cap_margin]
    [-max_utilization util]
    [-buffer_gain gain_ratio]
    [-match_cell_footprint]
    [-verbose]
```

if { [info exists env(TIE_SEPARATION)] } {

  set tie_separation $env(TIE_SEPARATION)

} else {

  set tie_separation 0

}

設置 tie cell 的間隔值，預設為 0

puts "Repair tie lo fanout..."

set tielo_cell_name [lindex $env(TIELO_CELL_AND_PORT) 0]

set tielo_lib_name [get_name [get_property [lindex [get_lib_cell

$tielo_cell_name] 0] library]]

```
set tielo_pin $tielo_lib_name/$tielo_cell_name/[lindex
$env(TIELO_CELL_AND_PORT) 1]
repair_tie_fanout -separation $tie_separation $tielo_pin
puts "Repair tie hi fanout..."
set tiehi_cell_name [lindex $env(TIEHI_CELL_AND_PORT) 0]
set tiehi_lib_name [get_name [get_property [lindex [get_lib_cell
$tiehi_cell_name] 0] library]]
set tiehi_pin $tiehi_lib_name/$tiehi_cell_name/[lindex
$env(TIEHI_CELL_AND_PORT) 1]
repair_tie_fanout -separation $tie_separation $tiehi_pin
```

- ✓ 獲取 tie-low, tie-high 單元和 pin 資訊
- ✓ 修復 tie-low, tie-high 的 fanout 連接

```
puts "Floating nets: "
report_floating_nets
report_metrics 3 "resizer" true false
puts "Instance count before $instance_count_before, after
[sta::network_leaf_instance_count]"
puts "Pin count before $pin_count_before, after
[sta::network_leaf_pin_count]"
```

write_db $::env(RESULTS_DIR)/3_4_place_resized.odb

✓ 報告浮接的網路

✓ 輸出 resizer 相關的指標

✓ 報告優化前後的實例數量和接腳數量變化

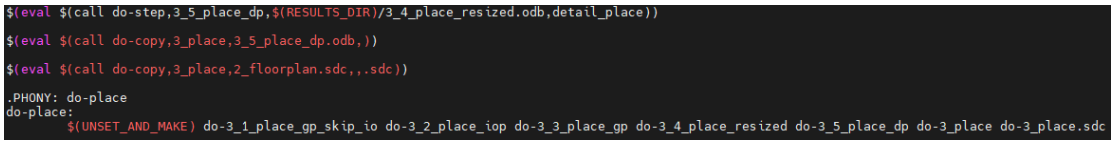
✓ 將優化後的設計寫入到指定的 ODB 檔案

所以第四步主要在處理:

插入緩衝器以改善驅動能力,修復時序問題

處理 tie-high 和 tie-low 的連接,確保電路滿足時序和電力的要求

Step5 Detail placement:



```
$(eval $(call do-step,3_5_place_dp,$(RESULTS_DIR)/3_4_place_resized.odb,detail_place))

$(eval $(call do-copy,3_place,3_5_place_dp.odb,))

$(eval $(call do-copy,3_place,2_floorplan.sdc,,.sdc))

.PHONY: do-place
do-place:
        $(UNSET_AND_MAKE) do-3_1_place_gp_skip_io do-3_2_place_iop do-3_3_place_gp do-3_4_place_resized do-3_5_place_dp do-3_place do-3_place.sdc
```

$(eval $(call do-step, 3_5_place_dp,

$(RESULTS_DIR)/3_4_place_resized.odb,detail_place))

輸入檔案是前一步驟的 resize 結果

執行 detail_place.tcl 腳本

$(eval $(call do-copy,3_place,3_5_place_dp.odb,))

將詳細佈局的結果複製為階段性的最終結果輸出為 3_place.odb

$(eval $(call do-copy,3_place,2_floorplan.sdc,,.sdc))

將 floorplan 階段的時序限制檔案複製到 place 階段輸出為 3_place.sdc

.PHONY: do-place

do-place:

$(UNSET_AND_MAKE) do-3_1_place_gp_skip_io do-3_2_place_iop do-3_3_place_gp do-3_4_place_resized do-3_5_place_dp do-3_place do-3_place.sdc

宣告 do-place 為虛擬目標順序執行所有 placement 相關的子步驟：

do-3_1_place_gp_skip_io: slip IO 的全域 placement

do-3_2_place_iop: IO placement

do-3_3_place_gp: 全域 placement

do-3_4_place_resized: 元件大小調整

do-3_5_place_dp: 詳細 placement

do-3_place: 複製最終 ODB

do-3_place.sdc: 複製時序限制檔案

接下來是關於 placement 與演算法的部分

整個 global placement 有以下參數可以調整

```
global_placement
    [-timing_driven]
    [-routability_driven]
    [-disable_timing_driven]
    [-disable_routability_driven]
    [-skip_initial_place]
    [-incremental]
    [-bin_grid_count grid_count]
    [-density target_density]
    [-init_density_penalty init_density_penalty]
    [-init_wirelength_coef init_wirelength_coef]
    [-min_phi_coef min_phi_conef]
    [-max_phi_coef max_phi_coef]
    [-reference_hpwl reference_hpwl]
    [-overflow overflow]
    [-initial_place_max_iter initial_place_max_iter]
    [-initial_place_max_fanout initial_place_max_fanout]
    [-pad_left pad_left]
    [-pad_right pad_right]
    [-skip_io]
    [-skip_nesterov_place]
    [-routability_use_grt]
    [-routability_target_rc_metric routability_target_rc_metric]
    [-routability_check_overflow routability_check_overflow]
    [-routability_max_density routability_max_density]
    [-routability_max_inflation_iter routability_max_inflation_iter]
    [-routability_inflation_ratio_coef routability_inflation_ratio_coef]
    [-routability_max_inflation_ratio routability_max_inflation_ratio]
    [-routability_rc_coefficients routability_rc_coefficients]
    [-timing_driven_net_reweight_overflow]
    [-timing_driven_net_weight_max]
    [-timing_driven_nets_percentage]
    [-keep_resize_below_overflow]
```

每個 option 都有其意義

主要控制:

timing_driven: Enable timing-driven placement

routability_driven: Enable routability-driven placement

skip_initial_place: Skip initial placement (BiCGSTAB solving) before

Nesterov placement, improves HPWL by ~5% on large designs

incremental: Enable incremental global placement

skip_io: Ignore IO ports when computing wirelength during placement

skip_nesterov_place: Skip Nesterov-based placement algorithm

基本的參數:

bin_grid_count: 設定 bin grid's 數量

- ✓ Default: Internal heuristic
- ✓ Range: [64, 128, 256, 512, ...], integer

density: 設定目標 placement density

- ✓ Default: 0.7 (70%)
- ✓ Range: [0-1], float

pad_left/pad_right: 設定 left/right padding in sites

- ✓ Default: 0
- ✓ Range: [1-MAX_INT], integer

優化:

init_density_penalty: Initial density penalty value

- ✓ Default: 8e-5
- ✓ Range: [1e-6 - 1e6], float

init_wirelength_coef: Initial wirelength coefficient

- ✓ Default: 0.25
- ✓ Range: Unlimited float

min_phi_coef: Lower bound

- ✓ Default: 0.95
- ✓ Range: [0.95-1.05], float

max_phi_coef: Upper bound

- ✓ Default: 1.05
- ✓ Range: [1.00-1.20], float

overflow: Target overflow for termination

- ✓ Default: 0.1
- ✓ Range: [0-1], float

迭代:

initial_place_max_iter: Maximum iterations in initial place

- ✓ Default: 20
- ✓ Range: [0-MAX_INT], integer

initial_place_max_fanout: Net escape condition threshold

- ✓ Default: 200
- ✓ Range: [1-MAX_INT], integer

Routability Parameters

routability_use_grt: Use FastRoute for routing congestion

- ✓ More precise but higher runtime
- ✓ Default: False (uses RUDY)

routability_target_rc_metric: Target RC metric

- ✓ Default: 1.01
- ✓ Range: Float values

routability_check_overflow: Overflow threshold

- ✓ Default: 0.3
- ✓ Range: [0-1], float

routability_max_density: Density threshold

- ✓ Default: 0.99
- ✓ Range: [0-1], float

routability_max_inflation_iter: Maximum inflation iterations

- ✓ Default: 4
- ✓ Range: [1-MAX_INT], integer

routability_inflation_ratio_coef: Inflation ratio coefficient

- ✓ Default: 5
- ✓ Range: Float values

routability_rc_coefficients: RC coefficients for congestion

- ✓ Default: {1.0, 1.0, 0.0, 0.0}
- ✓ Format: Tcl List {k1, k2, k3, k4}

Timing Parameters:

timing_driven_net_reweight_overflow: Timing-driven reweighting thresholds

- ✓ Default: [79, 64, 49, 29, 21, 15]
- ✓ Range: [0-100], integer list

timing_driven_net_weight_max: Maximum timing-critical net multiplier

- ✓ Default: 1.9
- ✓ Range: Float values

timing_driven_nets_percentage: Percentage of reweighted nets

- ✓ Default: 10
- ✓ Range: [0-100], float

keep_resize_below_overflow: Retention threshold for timing changes

- ✓ Default: 0
- ✓ Range: [0-1], float

Global Placement 的部分是基於 **RePlAce** algo./tools，

來自於這一篇論文

Advancing Solution Quality and Routability Validation in Global Placement

基本原理:

- 使用電力學模型來處理元件擺放
- 通過模擬電場力來決定元件的最佳位置

Nesterov's Method:是一種數學方法求解電力場方程式

主要特點:

- Mixed-size placement:可以同時處理大小不同的元件
- 在不同編譯器和作業系統下都能驗證確定性解決方案
- 使用 OpenDB(7/14/16/28/45/55/65nm)上驗證

其流程為:

初始擺放

↓

建立電力場模型

↓

使用 Nesterov's method 求解

↓

迭代優化直到收斂

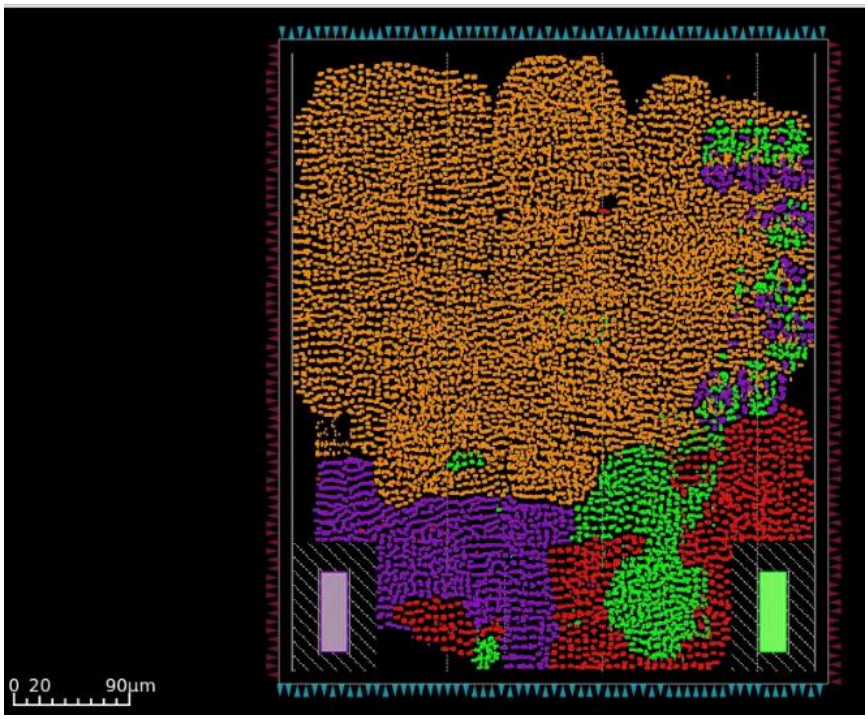
↓

輸出最終擺放結果

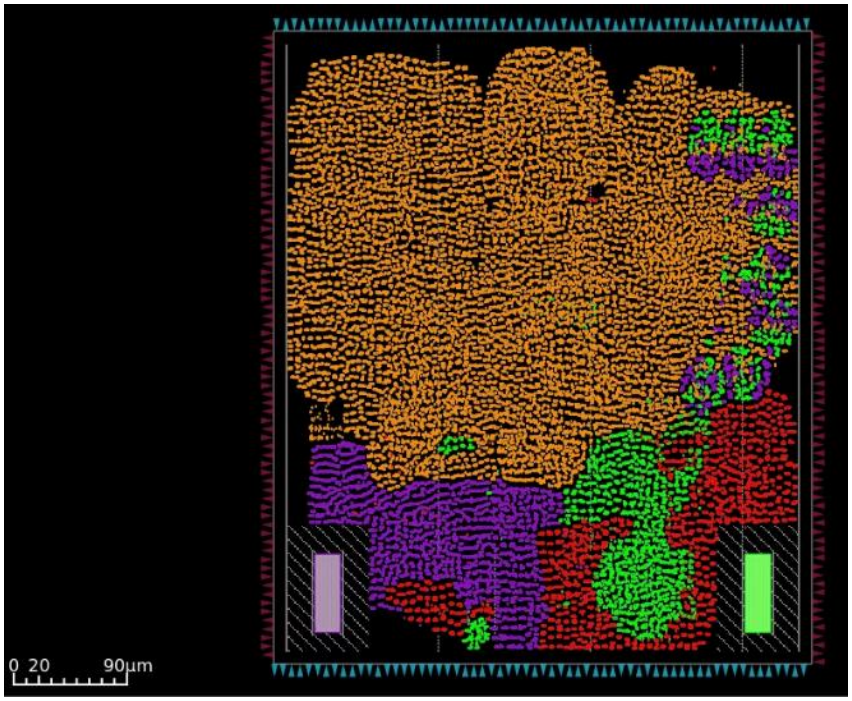
主要去調整的參數有 density, routability driven, timing driven 等

第一張圖片是原本還沒改動的結果(project1)

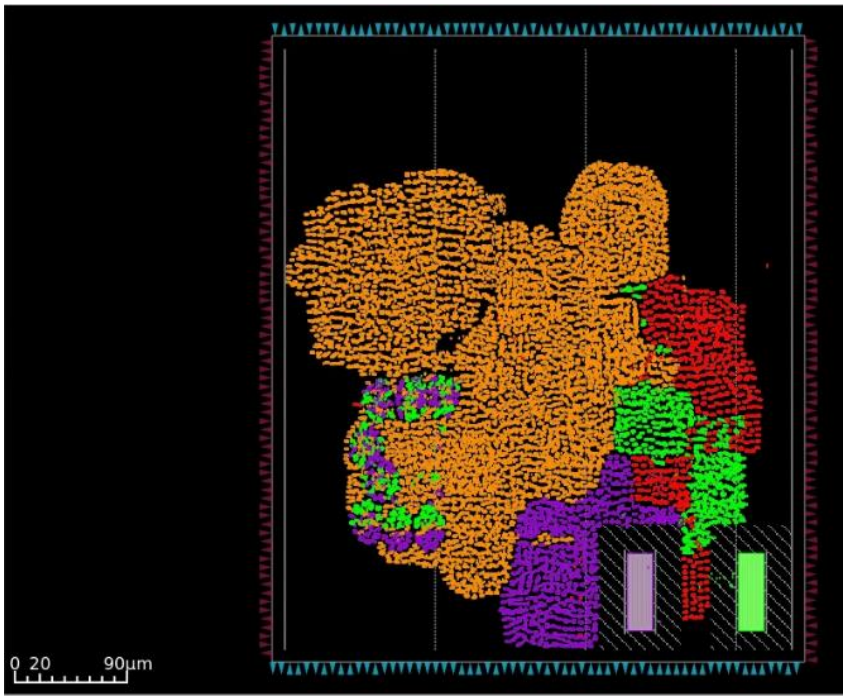
nangate45/tinyRocket (default)



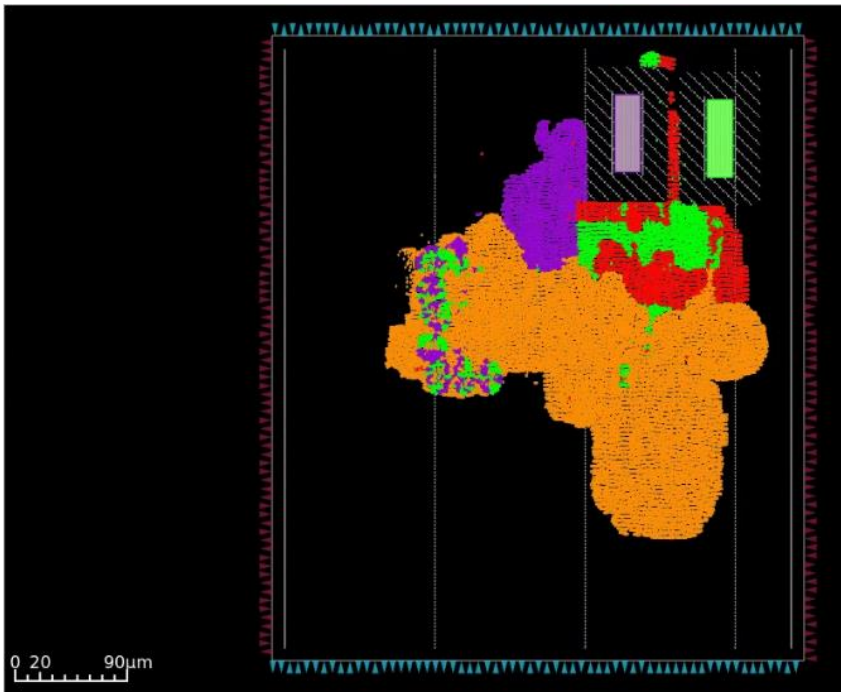
nangate45/tinyRocket (desity 0.3)



nangate45/tinyRocket (density 0.5)



nangate45/tinyRocket (density 0.9)



這是我測試下來的觀察低於 0.3 會 error，就像要把一堆東西塞進箱子，如果箱子太大散太開，反而沒辦法好好擺放，密度設為 0.9 的時候 make 跑得特別

久，因為要把元件塞得很密集，tools/algo. 需要花更多時間思考怎麼擺，

不同 density 的差異：

density 0.3：元件擺得很散，整個版面都攤開來用

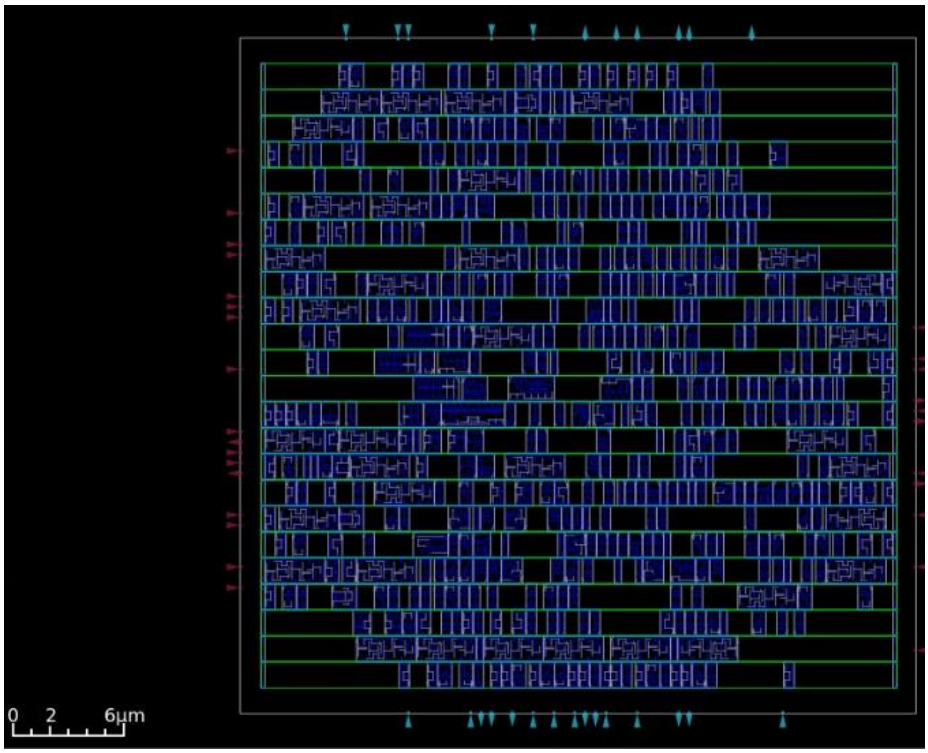
density 0.5：元件擺得比較集中，但還是有適當的分散

density 0.9：元件擠得很緊，用的空間變得很小

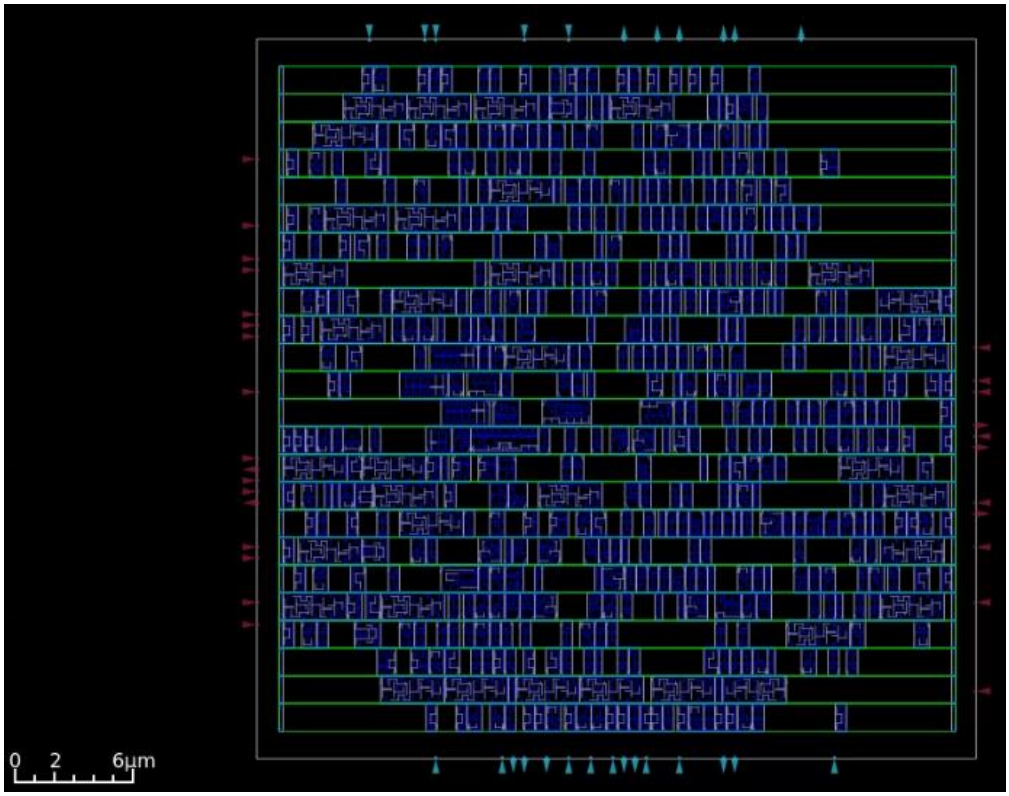
density 的意義：

- 這個值決定了每個區域能塞多少元件
- density 低的時候，工具會被迫把元件分散到更多地方
- density 高的時候，工具就會把元件塞得很緊

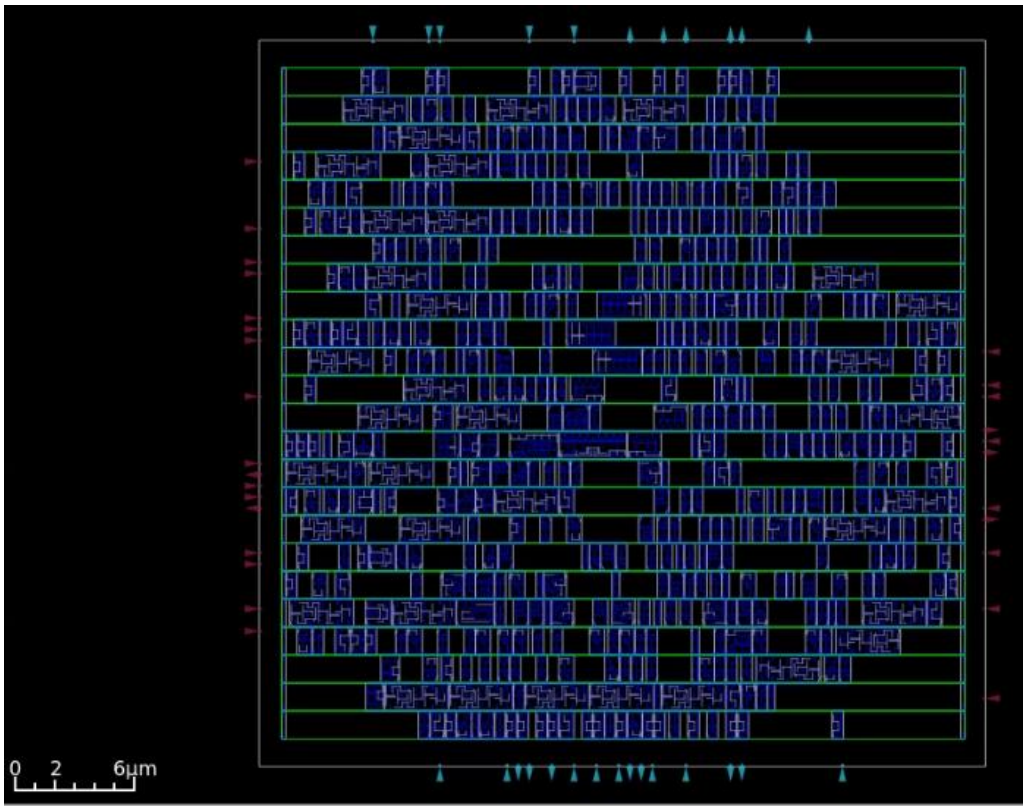
gcd-routability driven, timing driven 未開啟的狀態



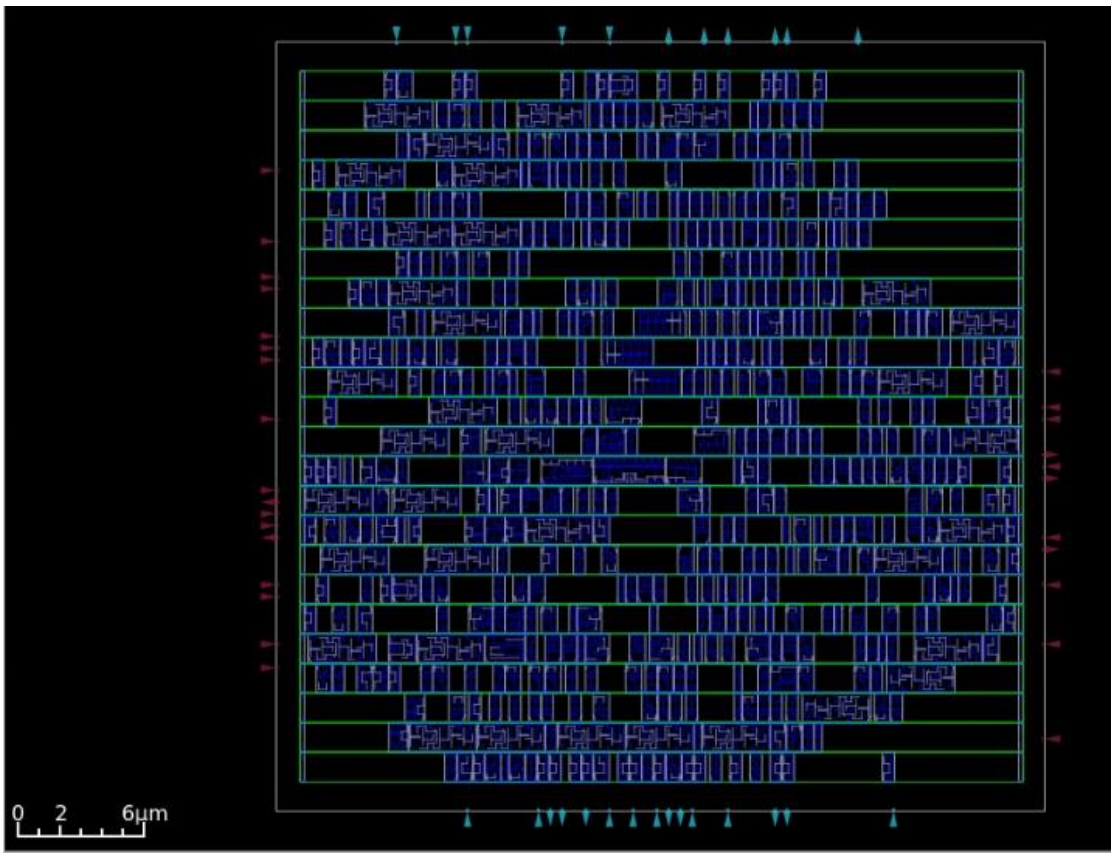
gcd-Routability driven 開啟的狀態



gcd-timing driven 開啟的狀態



gcd-都開啟的狀態



因為 tinyRocket 不好觀察其差異，所以在後續選擇用 GCD 來觀察，這些 cell 的分布位置都有些許不同，但 GCD desgin 比較簡單且小，有差異但不是特別明顯，以下是我觀察到:

1.當開啟 timing driven 時：

工具會試圖將時序相關的 cell 放得更近

2.當開啟 routability driven 時：

cell 的擺放會考慮到後續布線的需求，試圖避免某些區域太過擁擠

feedback: 這次特別選擇研究 OpenROAD 的 placement 部分，是因為這是我在之前 codinng 作業中成績最差。秉持在哪跌倒，就在哪裡爬起的精神，我希望能透過深入了解 OpenROAD 如何實作 placement，來提升自己在這方面的認知。這個專題不僅讓我能夠改進自己的弱項，也讓我有機會深入了解工業界實際使用的工具。

通過這次專題，我不僅對整個 OpenROAD 工具有了全面的認識，更深入理解了每個步驟背後的細節。這讓我意識到，成功的晶片設計不僅僅依賴於 placement 或 routing 的優化，還需要考慮諸多關鍵步驟，如 finishing、buffering、timing setup 等，這些步驟缺一不可，都是確保設計質量的重要環節。透過實際操作和調整參數，我更加理解了這些步驟之間的相互關係，以及如何影響最終的設計結果。

特別感謝這門 CAD 課程的設計如此用心，從理論基礎到實務應用都做了完整的規劃。讓我們能夠在理解理論的基礎上，實際操作完整的 EDA flow，並透過調整參數來理解每個步驟的影響。這種理論與實作結合的學習方式，對於培養實際解決問題的能力非常有幫助。

對於課程的建議，我認為可以考慮建立一個類似 HackerRank 的線上平台，讓學生能夠即時看到自己的分數並獲得回饋可以去改進自己的程式碼。雖然黑盒測試具有挑戰性，但若能提供更多的回饋機制，將更有助於學生理解自己的不足並加以改進。

此外，也建議學校可以開設更專精的進階課程，針對特定主題做更深入的探討，幫助有興趣深入研究的同學能夠繼續發展。

作為碩二生，這次的學習經驗讓我對 EDA 領域產生了更大的興趣。感謝教授和實驗室團隊提供了這麼好的學習機會，不僅讓我學到了寶貴的知識，也為未來的職業發展打下了良好的基礎。希望這門課程能持續發展，為更多對 EDA 領域有興趣的學生提供優質的學習機會