

Introduction of Openroad design flow

M11215060 陳軒緯

Part1:

Openroad的運作流程會將RTL Verilog(.v), constraints (.sdc), liberty (.lib) 和 technology (.lef) 作為 input, 最終可得到 tapeoutready GDSII file.(此檔案表示積體電路已設計完成, 可交由代工廠生產)

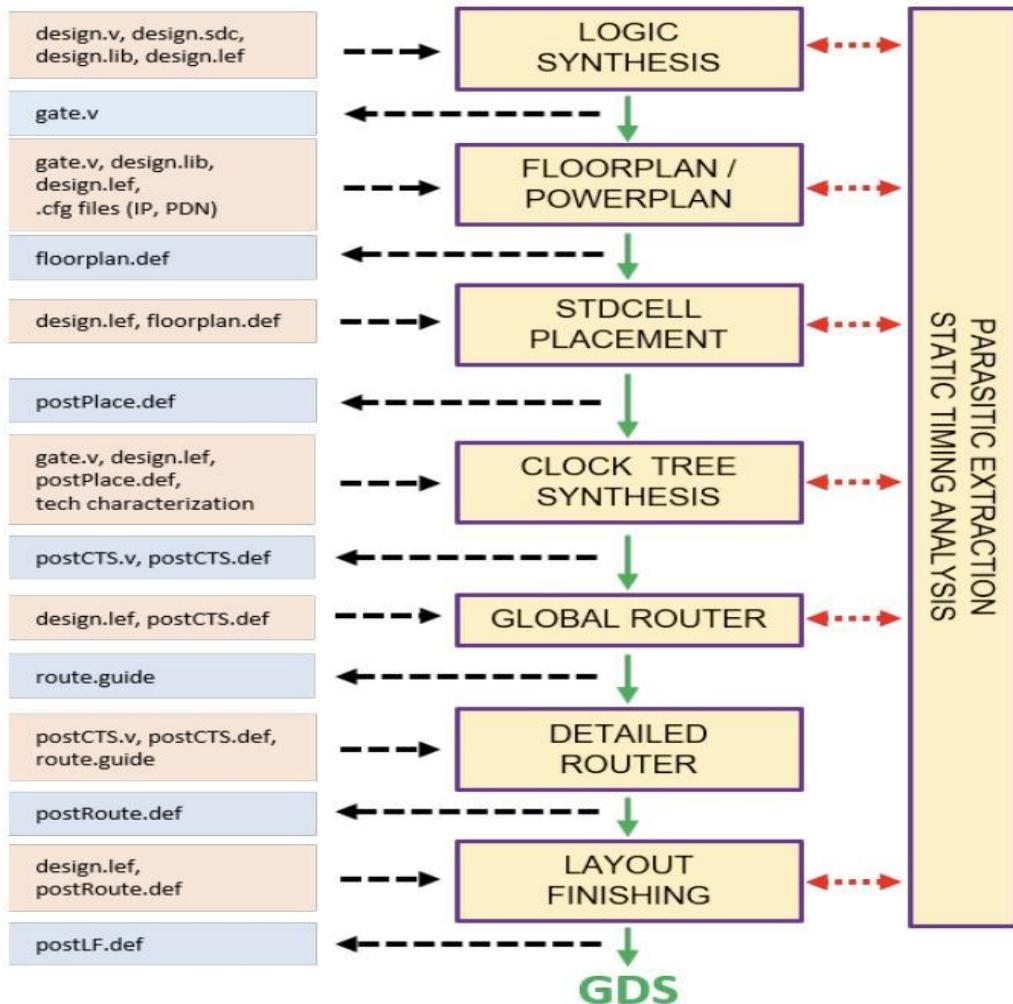


Figure 1: The OpenROAD flow.

Synthesis:

Openroad使用yosys 框架, 並且使用強化式學習開發了可以自動生成 step-by-step synthesis scripts, 以滿足時間限制, 同時最小化總面積。此階段主要是將RTL code轉成netlist 形式(邏輯閘gate-level), gate-level指的是把全部描述語言轉換成邏輯表示式, 之後再做placement和routing時, 就是以netlist為input。

```

    .resp_msg  ( dpath$resp_msg ),
    .is_a_lt_b ( dpath$is_a_lt_b )
);

// signal connections
assign ctrl$clk      = clk;
assign ctrl$is_a_lt_b = dpath$is_a_lt_b;
assign ctrl$is_b_zero = dpath$is_b_zero;
assign ctrl$req_val   = req_val;
assign ctrl$reset     = reset;
assign ctrl$resp_rdy  = resp_rdy;
assign dpath$a_mux_sel = ctrl$a_mux_sel;
assign dpath$a_req_en  = ctrl$a_req_en;
assign dpath$b_mux_sel = ctrl$b_mux_sel;
assign dpath$b_req_en  = ctrl$b_req_en;
assign dpath$clk       = clk;
assign dpath$req_msg_a = req_msg[31:16];
assign dpath$req_msg_b = req_msg[15:0];
assign dpath$reset     = reset;
assign req_rdy        = ctrl$req_rdy;
assign resp_msg        = dpath$resp_msg;
assign resp_val        = ctrl$resp_val;

endmodule // GcdUnit

//-----
// GcdUnitCtrlRTL_0x4d0fc71ead8d3d9e
//-----
// dump-vcd: False
// verilator-xinit: zeros
module GcdUnitCtrlRTL_0x4d0fc71ead8d3d9e
(
    output reg [ 1:0] a_mux_sel,
    output reg [ 0:0] a_req_en,
    output reg [ 0:0] b_mux_sel,
    output reg [ 0:0] b_req_en,
    input wire [ 0:0] clk,
    input wire [ 0:0] ts_a_lt_b,

```

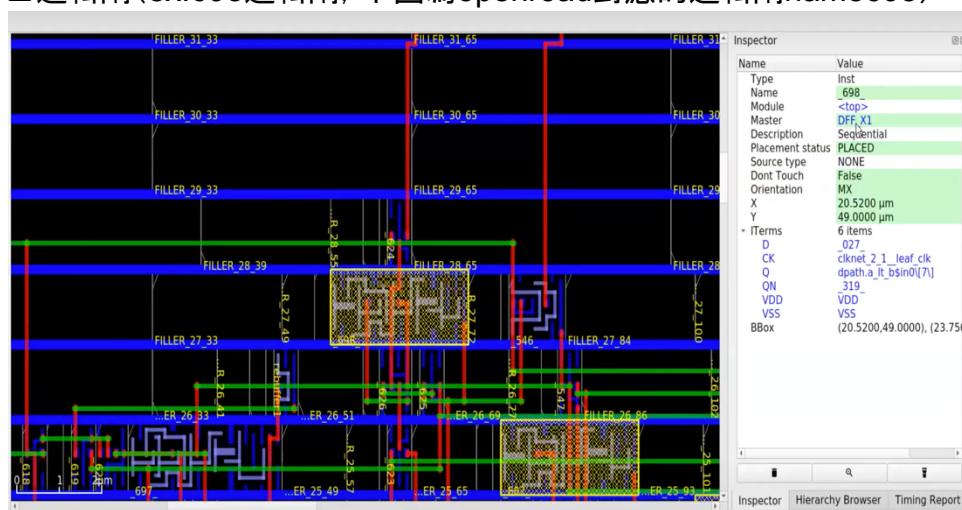
▲Rtl code

```

2206 DFF_X1_695_ (
2207     .CK(clk),
2208     .D1_024_,
2209     .Q(\`dpath.a_lt_b$in0[4]),
2210     .QN(_322_)
2211 );
2212 DFF_X1_696_ (
2213     .CK(clk),
2214     .D1_025_,
2215     .Q(\`dpath.a_lt_b$in0[5]),
2216     .QN(_321_)
2217 );
2218 DFF_X1_697_ (
2219     .CK(clk),
2220     .D1_026_,
2221     .Q(\`dpath.a_lt_b$in0[6]),
2222     .QN(_320_)
2223 );
2224 DFF_X1_698_ [
2225     .CK(clk),
2226     .D1_027_,
2227     .Q(\`dpath.a_lt_b$in0[7]),
2228     .QN(_319_)
2229 ];
2230 DFF_X1_699_ (
2231     .CK(clk),
2232     .D1_028_,
2233     .Q(\`dpath.a_lt_b$in0[8]),
2234     .QN(_318_)
2235 );
2236 DFF_X1_700_ (
2237     .CK(clk),
2238     .D1_029_,
2239     .Q(\`dpath.a_lt_b$in0[9]),
2240     .QN(_317_)
2241 );
2242 DFF_X1_701_ (
2243     .CK(clk),

```

▲邏輯閘(ex:698邏輯閘, 下圖為openroad對應的邏輯閘name698)



Floorplanning:

Floorplan主要內容為確認晶片的形狀和尺寸, IO單元(隨機)的placement, macro的placement和 Tapcell and Welltie的規劃, 最後生產電源分佈網路, 也就是電源系統整體路線規劃。

Step 1: Translate Verilog to odb

Step 2: IO placement (random)

Step 3: Timing Driven Mixed Sized Placement

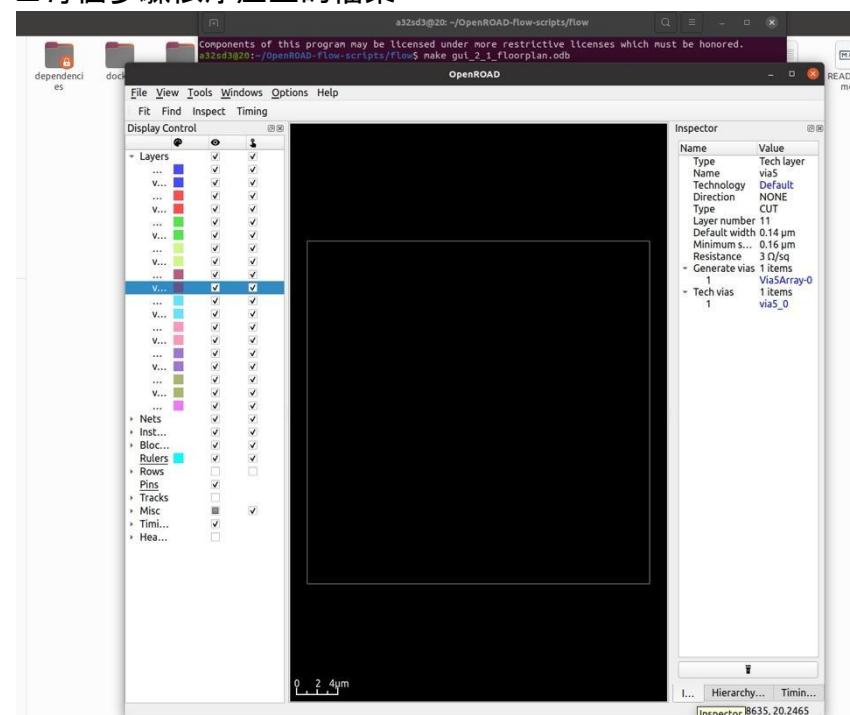
Step 4: Macro Placement

Step 5: Tapcell and Welltie insertion

Step 6: PDN generation

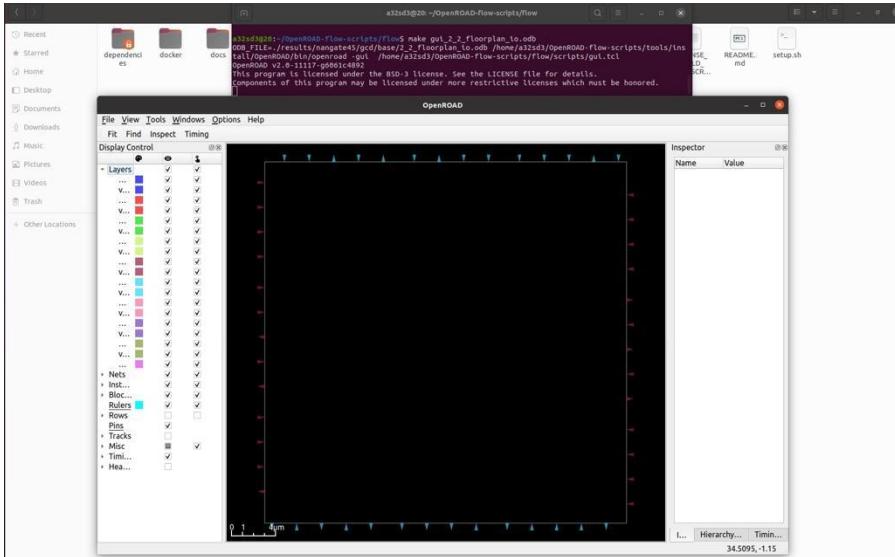
```
2_1_floorplan.odb  
2_2_floorplan_io.odb  
2_3_floorplan_tdms.odb  
2_4_floorplan_macro.odb  
2_5_floorplan_tapcell.odb  
2_6_floorplan_pdn.odb
```

▲每個步驟依序產生的檔案

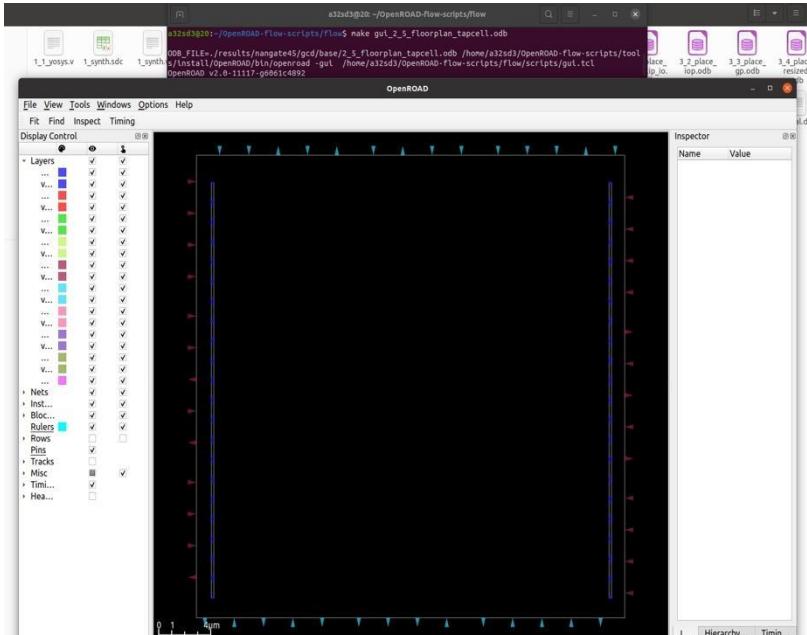


gui2_1_floorplan.odb

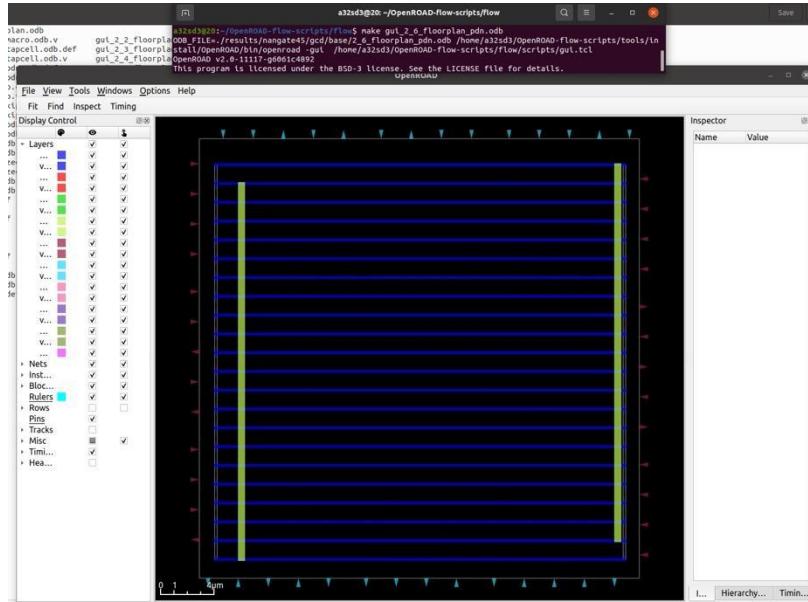
由RTL轉為odb, 因為還沒有任何placement, 所以可以看到畫面什麼東西都沒有。



gui2_2_floorplan_io.odb
此階段將輸入/輸出端口(I/O)隨機擺放



gui2_5_floorplan_tapcell.odb
此階段加入了多個Tap Cell。Tap Cell 是用於提供對VDD 和 VSS 等電源網絡
(在下一個階段會放入)的接入點的組合邏輯元件。它們通常被用於連接不同電
源區域或提供存取點，以確保電源傳遞網絡的有效性和穩定性。



gui2_6_floorplan_pdn.odb

此階段加入PDN(power delivery network), 包含VDD以及VSS。奇數排和右邊(高的)的導線是連通的VDD, 偶數排和左邊(低的)是連通的VSS。此階段確保VDD和VSS在整個集成電路上被正確且有效地分佈。

Placement:

Placement決定標準元件在晶片上的實體位置, 決定IO 的placement(非隨機)。這個階段, 會接收synthesized circuit netlist和technology library並產出一個合法的layout。這個layout根據前面步驟的優化已準備好resizing和buffering。(即下圖step3至step4)

```

# PLACE
#
# place: ${RESULTS_DIR}/3_place.odb \
#         ${RESULTS_DIR}/3_place.sdc
#
# STEP 1: Global placement without placed IOs, timing-driven, and routability-driven.
# -----
${RESULTS_DIR}/3_1_place_gp_skip_io.odb: ${RESULTS_DIR}/2_floorplan.odb ${RESULTS_DIR}/2_floorplan.sdc ${LIB_FILES}
    ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/global_place_skip_io.tcl -metrics ${LOG_DIR}/3_1_place_gp_skip_io.json) 2>&1 | tee ${LOG_DIR}/3_1_place_gp_skip_io.log

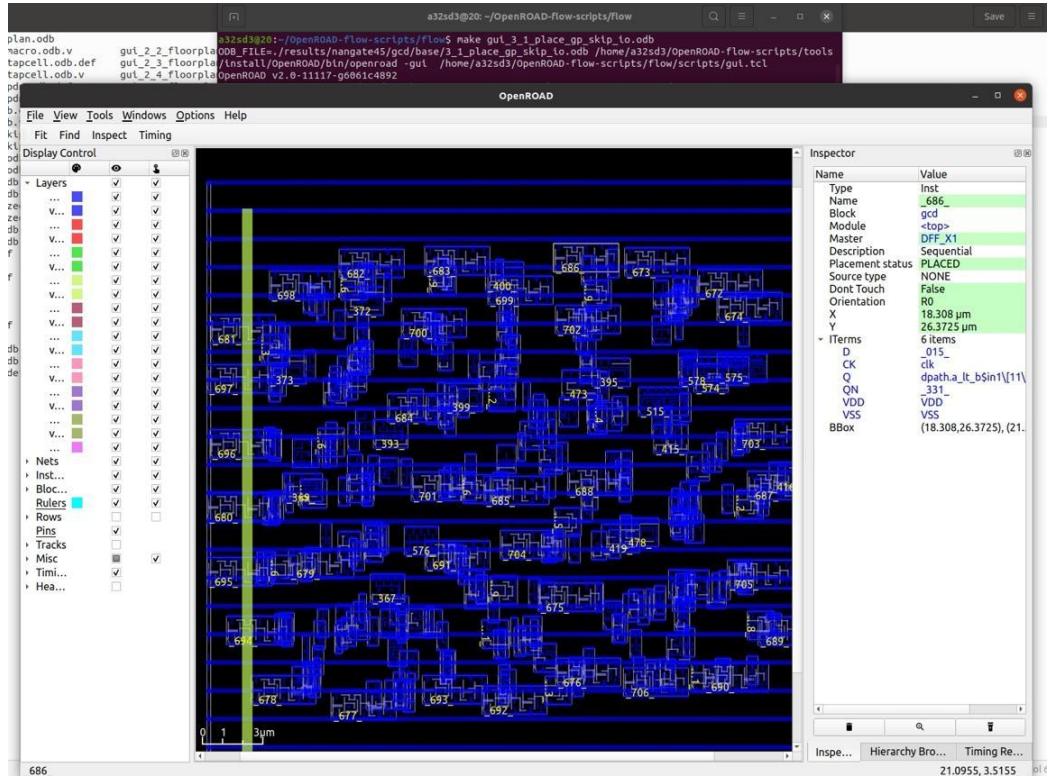
# STEP 2: IO placement (non-random)
# -----
${RESULTS_DIR}/3_2_place_iop.odb: ${RESULTS_DIR}/3_1_place_gp_skip_io.odb ${IO_CONSTRAINTS}
ifndef IS_CHIP
    ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/io_placement.tcl -metrics ${LOG_DIR}/3_2_place_iop.json) 2>&1 | tee ${LOG_DIR}/3_2_place_iop.log
else
    cp $< $o
endif.

# STEP 3: Global placement with placed IOs, timing-driven, and routability-driven.
# -----
${RESULTS_DIR}/3_3_place_gp.odb: ${RESULTS_DIR}/3_2_place_iop.odb ${RESULTS_DIR}/2_floorplan.sdc ${LIB_FILES}
    ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/global_place.tcl -metrics ${LOG_DIR}/3_3_place_gp.json) 2>&1 | tee ${LOG_DIR}/3_3_place_gp.log

# STEP 4: Resizing & Buffering
# -----
${RESULTS_DIR}/3_4_place_resized.odb: ${RESULTS_DIR}/3_3_place_gp.odb ${RESULTS_DIR}/2_floorplan.sdc
    ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/resize.tcl -metrics ${LOG_DIR}/3_4_resizer.json) 2>&1 | tee ${LOG_DIR}/3_4_resizer.log
clean_resize:
    rm -f ${RESULTS_DIR}/3_4_place_resized.odb

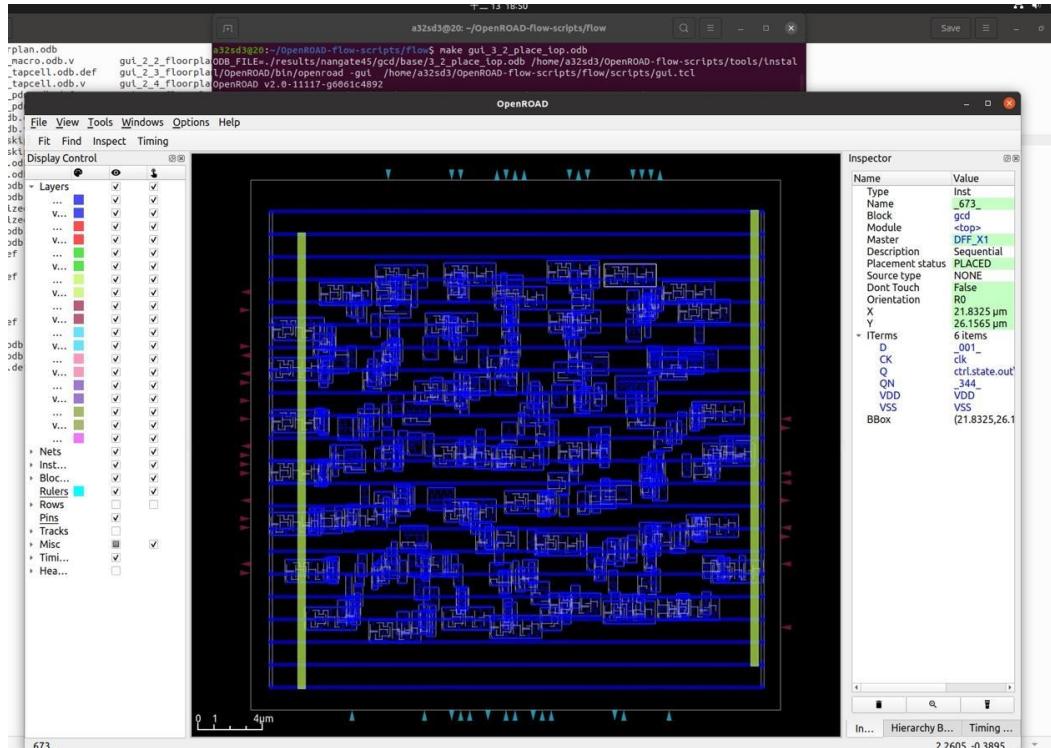
# STEP 5: Detail placement
# -----

```



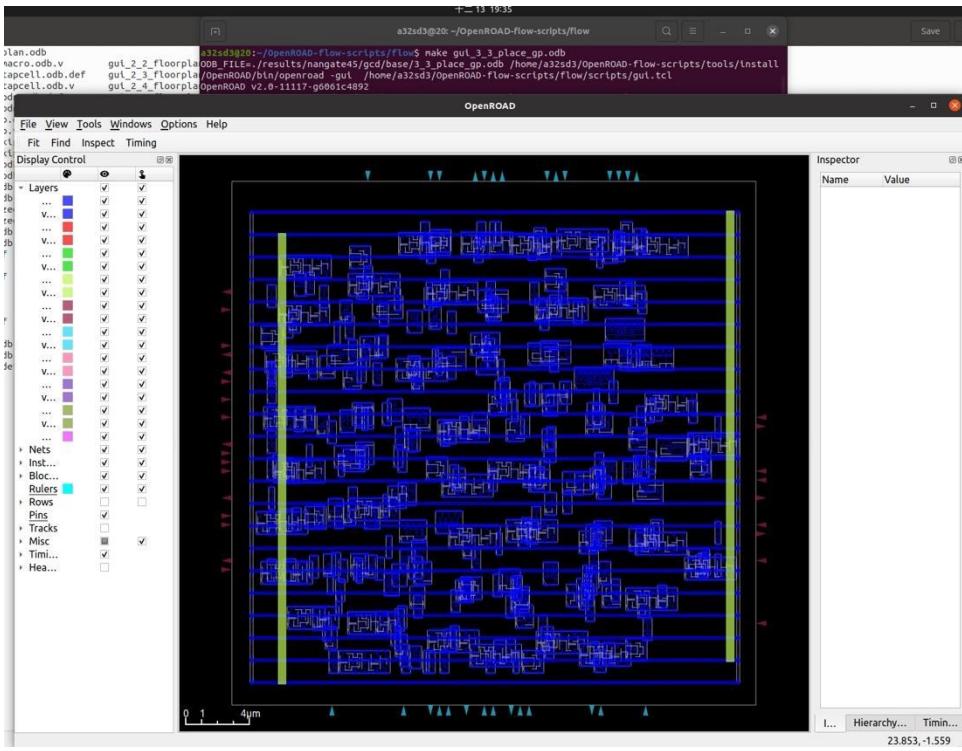
gui_3_1_place_gp_skip_io.odb

先將邏輯元件在不考慮輸入/輸出端口(IO)的狀況對元件做擺放。



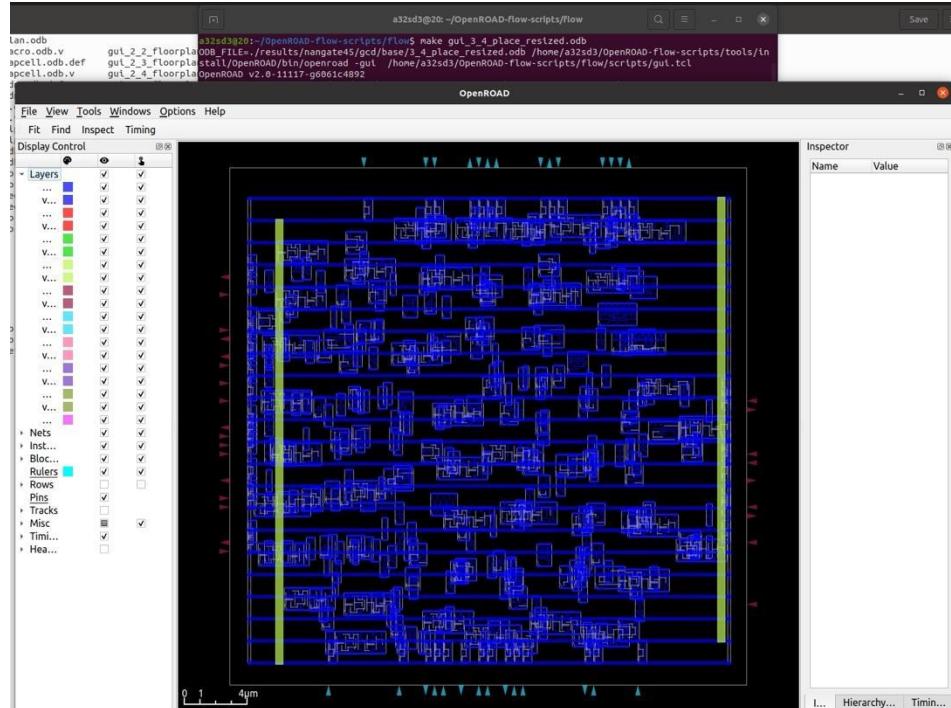
gui_3_2_place_iop.odb

輸入/輸出端口(IO) placement主要涉及放置設計中的輸入和輸出端口，以確保良好的連接性、訊號完整性和性能。IO placement 的目標是將 IO pins 放置在晶片上的適當位置，以滿足特定的電氣和布局需求。



gui_3_3_place_gp.odb

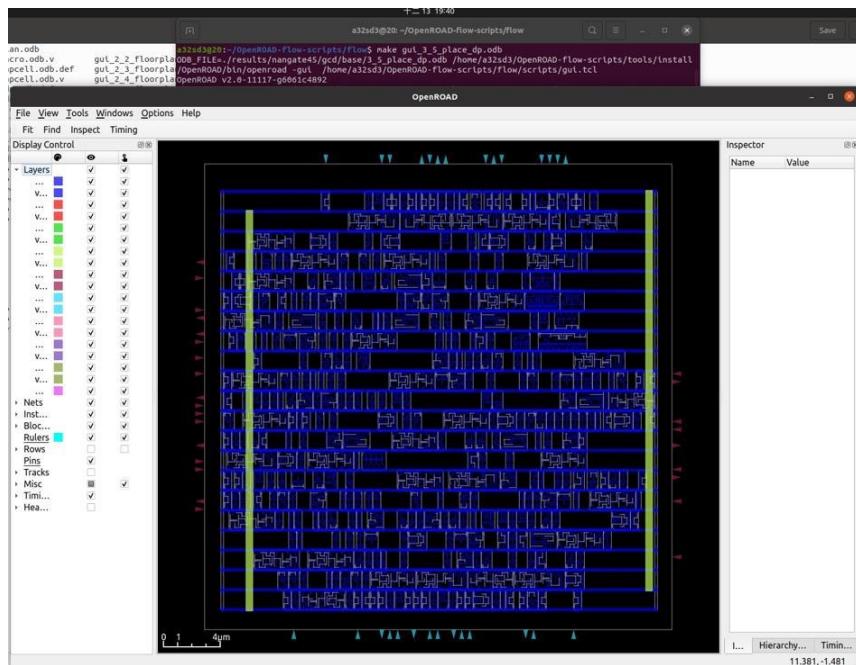
考慮全局佈局、IOs 的放置、時序需求和可路由性的同時，為後續的佈線和布局階段打下基礎。在此階段會確保整個設計在物理層面上滿足預期的性能、時序和



佈局要求。

gui_3_4_place_resized.odb

主要用於調整和緩衝邏輯元件，以優化時序、功耗和面積。Resizing考慮整體 IC 的面積，通過調整邏輯元件的大小，以實現更緊湊的佈局。這有助於減小整體 IC 的面積，提高集成度。Buffering在長線路或延遲敏感的區域插入緩衝器，以改善信號的傳輸延遲。這有助於保持信號的穩定性，避免時序問題。

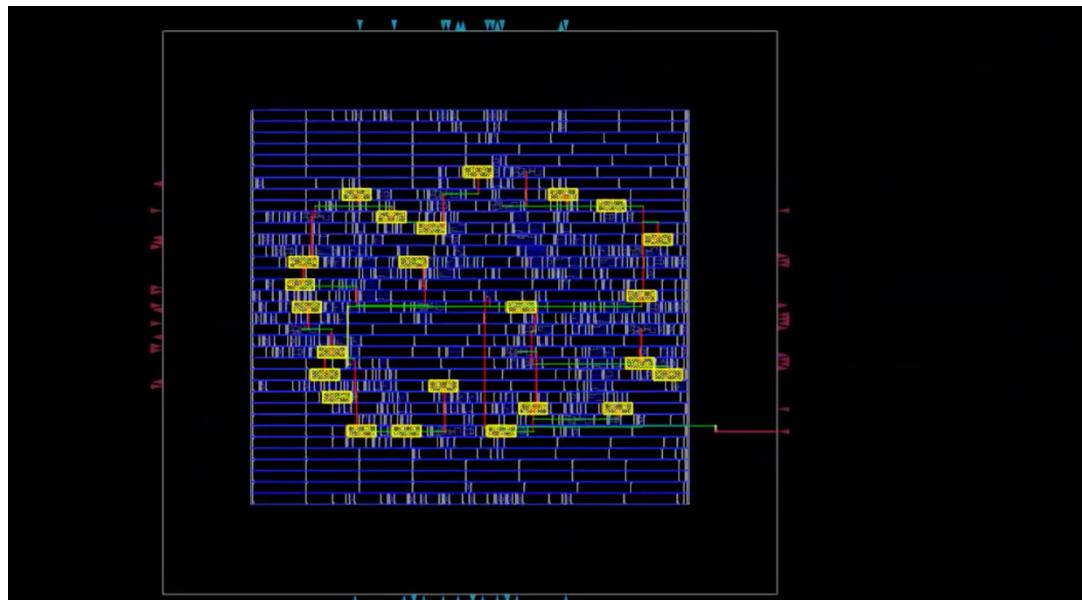


gui_3_5_place_db.odb

在 Detail Placement 階段，必須確保所有邏輯元件的放置滿足時序約束，包括最大傳輸延遲、最小傳輸延遲、時鐘周期等。同時，考慮電氣特性，確保信號完整性。

CTS:

產生clock tree，將外部clock妥善分配給內部的各個元件(clock tree synthesis)。CTS 是一個 clock balancing 的技術，用以維持訊號的完整性，降低 clock skew 和 clock latency。在CTS這個階段會對 clock tree 作分析、優化 clock 的擺放位置、在clock路徑上加buffer來推動clock tree。



▲在Openroad中Clock tree(綠、紅線) 將Flip-Flop(黃色區塊)連接起來

Routing:

在placement擺好cell之後，對擺放好的 cell 作拉線。在Openroad中，會先run global route，接著再run detailed route。

```
# STEP 1: Run global route
#-----
#${RESULTS_DIR}/5_1_grt.odb; ${RESULTS_DIR}/4_cts.odb ${FASTROUTE_TCL} ${PRE_GLOBAL_ROUTE}
#      ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/global_route.tcl -metrics ${LOG_DIR}/5_1_fastroute.json 2>&1 | tee ${LOG_DIR}/5_1_fastroute.log

# STEP 2: Run detailed route
#-----
ifeq (${USE_WXL},)
${RESULTS_DIR}/5_2_route.odb: ${RESULTS_DIR}/5_1_grt.odb
else
${RESULTS_DIR}/5_2_route.odb: ${RESULTS_DIR}/4_cts.odb
endif
      ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/detail_route.tcl -metrics ${LOG_DIR}/5_2_TritonRoute.json 2>&1 | tee ${LOG_DIR}/5_2_TritonRoute.log

${RESULTS_DIR}/5_route.odb: ${RESULTS_DIR}/5_2_route.odb
cp $< $@

${RESULTS_DIR}/5_route.sdc: ${RESULTS_DIR}/4_cts.sdc
cp $< $@
cp $< $0

clean_route:
rm -rf output/* results*.out.dmp layer_*_mps
rm -rf *.gdid *.log *.met *.sav *.res.dmp
rm -rf ${RESULTS_DIR}/route.guide ${RESULTS_DIR}/output_guide.mod ${RESULTS_DIR}/updated_clks.sdc
rm -rf ${RESULTS_DIR}/5_*_odb ${RESULTS_DIR}/5_route.sdc
rm -f ${REPORTS_DIR}/5_*
rm -f ${LOG_DIR}/5_*

klayout_tr_rpt: ${RESULTS_DIR}/5_route.def ${OBJECTS_DIR}/klayout.lyt
${KLAUT_LAYOUT_FOUND}
${KLAUT_CMD} -rd in_drc="${REPORTS_DIR}/5_route_drc.rpt" \
    -rd in_def="`$`" \
    -rd tech_file=${OBJECTS_DIR}/klayout.lyt \

```

++ 13 21:34
a32sd3@20: ~/OpenROAD-flow-scripts/flow

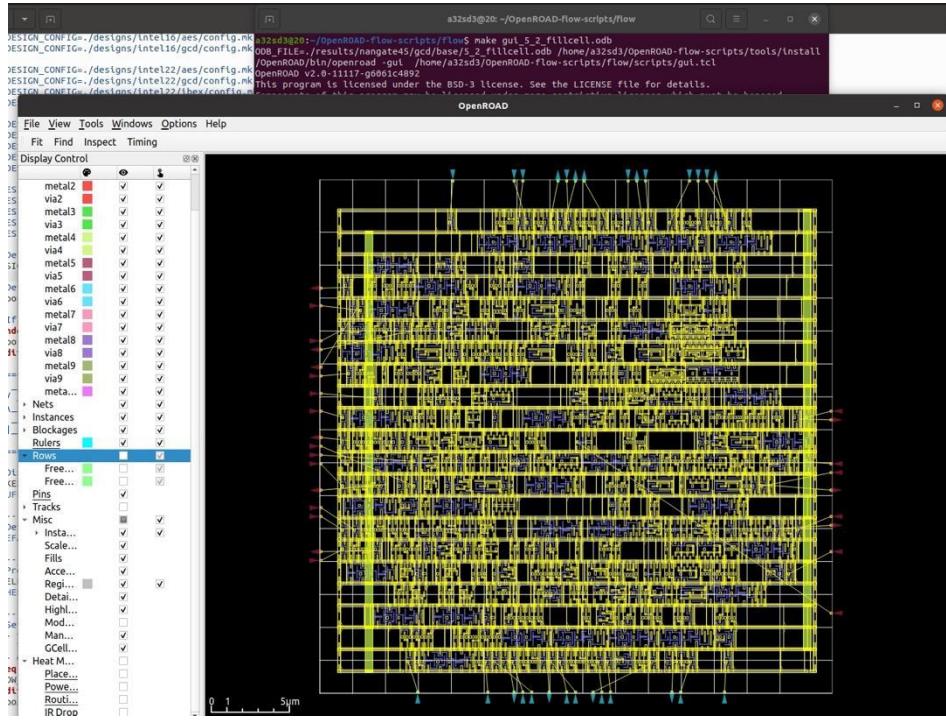
OpenROAD

Inspector

Name	Value
Type	Net
Name	VSS
Block	gcd
Signal type	DONE
Wire type	ROUTED
Special	True
Dont Touch	False
TERMS	389 items
BTERMS	0 items
Bbox	(1,14,1,315), (31,7)

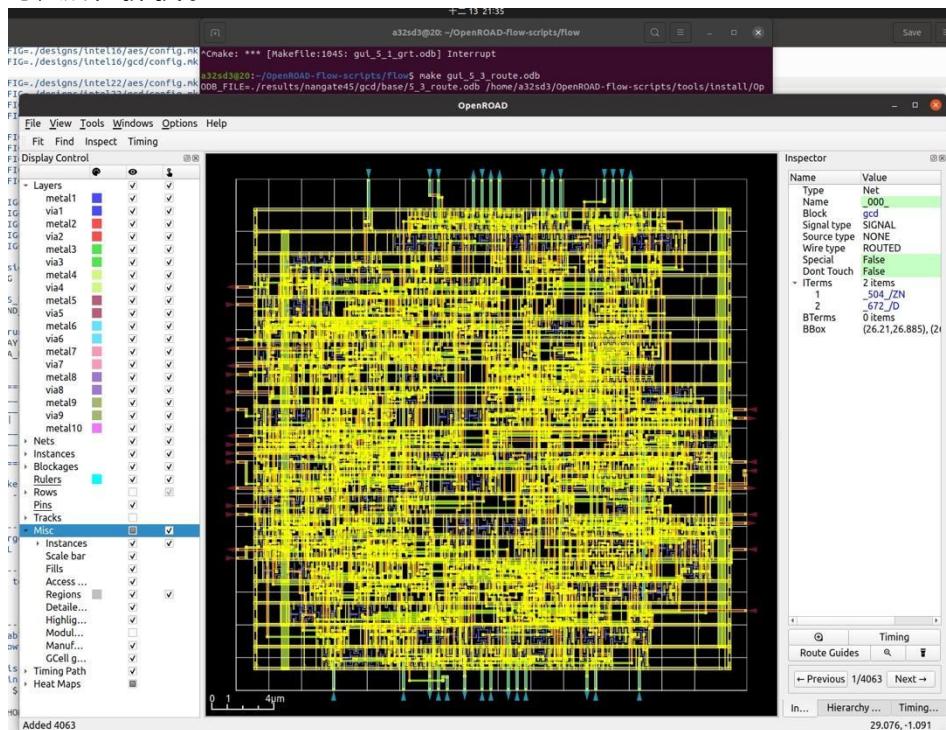
gui_5_1_grt.odb

做global routing，主要關注的是整個芯片的全局佈局，以確保各個元件的大致連接關係和路徑。全局布線的目的是確保元件之間的連接關係，並解決可能的區域布線擁擠問題。



gui_5_2_fillcell.odb

Fill cell routing 可以使用填充單元(fill cell)來填充空白區域，從而使布線更加均勻，減輕擁擠。



gui_5_3_route.odb

最後做detailed routing，可以發現一些細節上面線路變整齊許多。

Finishing:

執行metal fill insertion，目的是把電源線接在一起，並且檢查其設計有無錯誤，以此提升晶片的良率。此階段會回報任務結束時間。

```
$LOG_DIR/6_report.log: ${RESULTS_DIR}/6_1_fill.odb ${RESULTS_DIR}/6_1_fill.sdc  
    ($TIME_CMD) ${OPENROAD_CMD} ${SCRIPTS_DIR}/final_report.tcl -metrics ${LOG_DIR}/6_report.json 2>&1 | tee ${LOG_DIR}/6_report.log  
  
${RESULTS_DIR}/6_final.def: ${LOG_DIR}/6_report.log
```

▲把 fill.odb接起來

```

$(WRAPPED_GDOAS): $(OBJECTS_DIR)/klayout_wrap.lyt $(WRAPPED_LEFS)
$(call KLAYOUT_FOUND)
($TIME_CMD) $(K_LAYOUT_CMD) -zz -rd design_name=$(basename $(notdir $@)) \
-rd in_def=$(OBJECTS_DIR)/def$(notdir $(@:$(STREAM_SYSTEM_EXT)=def)) \
-rd in_files="$(ADDITIONAL_GDOAS)" \
-rd config_file=$(FILE_CONFIG) \
-rd seal_file="" \
-rd out_file=@ \
-rd tech_file=$(OBJECTS_DIR)/klayout_wrap.lyt \
-rd layer_map=$(GDS_LAYER_MAP) \
-r $(UTILS_DIR)/def2stream.py 2>&1 | tee $(LOG_DIR)/6_merge_$(basename $(notdir $@)).log

```

▲使用Klayout 把 wrapped macros 合併起來

```
GDS_MERGED_FILE = ${RESULTS_DIR}/6_1_merged.${STREAM_SYSTEM_EXT}
$(GDS_MERGED_FILE): ${RESULTS_DIR}/6_final.def ${OBJECTS_DIR}/klayout.lyt ${GDSOAS_FILES} ${WRAPPED_GDSOAS} ${SEAL_GDSOAS}
    $call KLUAYOUT_FOUND)
    ($TIME_CMD) ${STDBUF_CMD} ${KLUAYOUT_CMD} -zz -rd design_name=${DESIGN_NAME} \
        -rd in_def=< \
        -rd in_file="${GDSOAS_FILES} ${WRAPPED_GDSOAS}" \
        -rd config_file=${FILL_CONFIG} \
        -rd seal_file=${SEAL_GDSOAS} \
        -rd out_file=q0 \
        -rd tech_file=${OBJECTS_DIR}/klayout.lyt \
        -rd layer_map=${GDS_LAYER_MAP} \
        -r ${UTILS_DIR}/def2stream.py 2>&1 | tee ${LOG_DIR}/6_1_merge.log
```

▲使用Klayout 把 GDS接起來

參考資料：

<https://vlsicad.ucsd.edu/Publications/Conferences/371/c371.pdf>

<https://shininglionking.blogspot.com/2013/05/2-ic.html>

[https://en.wikipedia.org/wiki/Placement_\(electronic_design_automation\)](https://en.wikipedia.org/wiki/Placement_(electronic_design_automation))

https://www.youtube.com/watch?v=mUgyantHBhg&ab_channel=Yi-YuLiu%28%E5%8A%89%E4%B8%80%E5%AE%87%29

Part 2: 深入探討Floorplaning

在Makefile中，.PHONY 是一個特殊的目標，表示它不代表實際的文件，而是一個偽目標，用於指示make在執行該目標時不要檢查是否存在對應的文件。

ifneq (\$FOOTPRINT,) 和 else ifneq (\$FOOTPRINT_TCL,) 這兩個條件語句用於檢查變數 FOOTPRINT 和 FOOTPRINT_TCL 是否被定義，如果其中一個被

定義，則將變數 IS_CHIP 設置為1。

UNSET_VARS包含一條Bash命令的變數，它用於在運行目標之前取消設置一組變數(UNSET_VARIABLES_NAMES 中列出的變數)。

SUB_MAKE 和 UNSET_AND_MAKE包含了一些用於調用子Makefile的命令。
SUB_MAKE 用於將命令行參數傳遞給子Makefile，而 UNSET_AND_MAKE 包裝了一個Bash命令，用於在運行子Makefile之前取消設置一組變數。

```
--  
514 # Separate dependency checking and doing a step. This can  
515 # be useful to retest a stage without having to delete the  
516 # target, or when building a wafer thin layer on top of  
517 # ORFS using CMake, Ninja, Bazel, etc. where makefile  
518 # dependency checking only gets in the way.  
519 #  
520 # Note that there is no "do-synth" step as it is a special  
521 # first step that for usecases such as Bazel where it should  
522 # always be built when invoked. Latter stages in the build process  
523 # are conditionally built by the Bazel implementation.  
524 #  
525 # A "do-synth" step would be welcomed, but it isn't strictly necessary  
526 # for the Bazel use-case.  
527 #  
528 # do-floorplan, do-place, do-cts, do-route, do-finish are the  
529 # supported interface to execute those stages without checking  
530 # for dependencies.  
531 #  
532 # The do- substeps of each of these stages are subject to change.  
533 #  
534 # $(1) stem, e.g. 2_1_floorplan  
535 # $(2) dependencies  
536 # $(3) tcl script step  
537 # $(4) extension of result, default .odb  
538 # $(5) folder of target, default ${RESULTS_DIR}  
539 define do-step  
540 $(if ${5}, ${5}, ${RESULTS_DIR})/${1}${(if ${4}, ${4}, .odb)}: ${2}  
541     $$(${UNSET_AND_MAKE} do-${1})  
542 |  
543 .PHONY: do-${1}  
544 do-${1}:  
545     (trap 'mv ${LOG_DIR}/${1}.tmp.log ${LOG_DIR}/${1}.log' EXIT; \  
546     ${TIME_CMD} ${OPENROAD_CMD} ${SCRIPTS_DIR}/${3}.tcl -metrics ${LOG_DIR}/${1}.json 2>&1 | \  
547     tee ${LOG_DIR}/${1}.tmp.log  
548 endef  
549
```

do-step用於構建過程中的不同階段。每個階段，比如平面佈局或放置，都有特定的依賴關係，並由相應的tcl檔執行。規則指定了如何構建這些階段，並記錄執行日誌。簡單來說，這是一組構建軟體專案不同部分的說明。do-step模板是每個步驟的藍圖，確保它們按照正確的順序構建，並具有必要的依賴關係。

```

# generate make rules to copy a file, if a dependency change and
# a do- sibling rule that copies the file unconditionally.
#
# The file is copied within the $(RESULTS_DIR)
#
# $(1) stem of target, e.g. 2_2_floorplan_io
# $(2) basename of file to be copied
# $(3) further dependencies
# $(4) target extension, default .odb
#define do-copy
$(RESULTS_DIR)/$(1)$($if $(4),$(4),.odb): $(RESULTS_DIR)/$(2) $(3)
    $$($UNSET_AND_MAKE) do-$1$($if $(4),$(4),)

.PHONY: do-$1$($if $(4),$(4),)
do-$1$($if $(4),$(4),):
    cp $(RESULTS_DIR)/$(2) $(RESULTS_DIR)/$(1)$($if $(4),$(4),.odb)
#endif

```

產生make規則，如果依賴項發生更改，則複製文件，並產生一個do-同級規則，無條件地複製文件。檔案將被複製到\$(RESULTS_DIR)目錄中

\$1)，例如2_2_floorplan_io

\$2) 要複製的檔案的基本名稱

\$3) 其他依賴項

\$4) 目標副檔名，預設為.odb

```

569 # STEP 1: Translate verilog to odb
570 #
571 $eval $(call do-step,2_1_floorplan,$(RESULTS_DIR)/1_synth.v $(RESULTS_DIR)/1_synth.sdc $(TECH_LEF) $(SC_LEF) $(ADDITIONAL_LEFS) $(FOOTPRINT) $(SIG_MAP_FILE) $(FOOTPRINT_TCL),floorplan)
572
573 # STEP 2: IO Placement (random)
574 #
575 ifndef IS_CHIP
576 $eval $(call do-step,2_2_floorplan_io,$(RESULTS_DIR)/2_1_floorplan.odb $(IO_CONSTRAINTS),io_placement_random)
577 else
578 $eval $(call do-copy,2_2_floorplan_io,2_1_floorplan.odb,$(IO_CONSTRAINTS))
579 endif
580
581 # STEP 3: Timing Driven Mixed Sized Placement
582 #
583 ifeq ($(MACRO_PLACEMENT)$(MACRO_PLACEMENT_TCL)$(RTLMP_FLOW),)
584 $eval $(call do-step,2_3_floorplan_tdns,$(RESULTS_DIR)/2_2_floorplan_io.odb $(RESULTS_DIR)/1_synth.v $(RESULTS_DIR)/1_synth.sdc $(LIB_FILES),tdns_place)
585 else
586 $eval $(call do-copy,2_3_floorplan_tdns,2_2_floorplan_io.odb,$(RESULTS_DIR)/1_synth.v $(RESULTS_DIR)/1_synth.sdc $(LIB_FILES))
587 endif
588
589 # STEP 4: Macro Placement
590 #
591 $eval $(call do-step,2_4_floorplan_macro,$(RESULTS_DIR)/2_3_floorplan_tdns.odb $(RESULTS_DIR)/1_synth.v $(RESULTS_DIR)/1_synth.sdc $(MACRO_PLACEMENT_TCL),macro_place)
592
593 $eval $(call do-step,2_5_floorplan_debug_macros,$(RESULTS_DIR)/2_1_floorplan.odb $(RESULTS_DIR)/1_synth.v $(MACRO_PLACEMENT) $(MACRO_PLACEMENT_TCL),floorplan_debug_macros)
594
595 # STEP 5: Tapcell and Welltie insertion
596 #
597 $eval $(call do-step,2_5_floorplan_tapcell,$(RESULTS_DIR)/2_4_floorplan_macro.odb $(TAPCELL_TCL),tapcell)
598
599 # STEP 6: PDN generation
600 #
601 $eval $(call do-step,2_6_floorplan_pdn,$(RESULTS_DIR)/2_5_floorplan_tapcell.odb $(PDN_TCL),pdn)
602
603 $eval $(call do-copy,2_floorplan,2_6_floorplan_pdn.odb,)
604
605 $(RESULTS_DIR)/2_floorplan.sdc: $(RESULTS_DIR)/2_1_floorplan.odb
606
607 PHONY: do-floorplan
608 do-floorplan:
609     mkdir -p $(LOG_DIR) $(REPORTS_DIR)
610     $$($UNSET_AND_MAKE) do-2_1_floorplan do-2_2_floorplan_io do-2_3_floorplan_tdns do-2_4_floorplan_macro do-2_5_floorplan_tapcell do-2_6_floorplan_pdn do-2_floorplan
611
612 PHONY: clean_floorplan
613 clean_floorplan:
614     rm -f $(RESULTS_DIR)/2_*floorplan*.odb $(RESULTS_DIR)/2_floorplan.sdc $(RESULTS_DIR)/2_*.v $(RESULTS_DIR)/2_*.def
615     rm -f $(REPORTS_DIR)/2_*
616     rm -f $(LOG_DIR)/2_*
617
618 # =====

```

makefile中floorplanning做的事

Step1主要作用是將Verilog檔轉成.odb，以下為詳細代碼解析：

`$($eval $(call do-step, ...))`是一個Makefile中的內建函數`eval`的使用，用於在Makefile中動態產生規則。`do-step`是一個模板，接受一系列參數產生用於執行特定步驟的規則。

`2_1_floorplan`:這是產生的目標的名稱(target name)的一部分，表示執行的步驟。在這裡，目標名稱為"2_1_floorplan"。

`$(RESULTS_DIR)/1_synth.v $(RESULTS_DIR)/1_synth.sdc $(TECH_LEF)
$(SC_LEF) $(ADDITIONAL_LEFS) $(FOOTPRINT) $(SIG_MAP_FILE)`
`$(FOOTPRINT_TCL)`: 這是產生目標所需的依賴項 列表。這包括Verilog程式碼
檔案、時序約束檔案、技術庫檔案(TECH LEF)、場景庫檔案(SC LEF)、其他附
加程式庫檔案(ADDITIONAL LEFS)、印記(FOOTPRINT)、訊號對應檔案(
SIG_MAP_FILE)和印記TCL腳本(FOOTPRINT_TCL) 等。

floorplan是執行該步驟的TCL腳本的名稱。此TCL腳本的路徑是由
`$(SCRIPTS_DIR)/floorplan.tcl`建構的。

Step 2:

`ifndef IS_CHIP`是一個條件判斷，檢查變數IS_CHIP是否未定義。如果未定義，
表示不是晶片等級的設計。

`$(eval $(call do-step, ...))`如果不是晶片層級的設計，請呼叫do-step模板來產生
執行步驟的規則。這裡的步驟名稱為"2_2_floorplan_io"，依賴項包括
"2_1_floorplan.odb"(前一步的結果)和IO約束文件(IO_CONSTRAINTS)，執行
的TCL腳本為"io_placement_random"。

`else` 如果IS_CHIP已定義，表示是晶片等級的設計。

`$(eval $(call do-copy, ...))`: 在晶片層級的設計中，呼叫do-copy模板產生規則，用
於複製"2_1_floorplan.odb"到目前步驟的目標。這是因為在晶片層級的設計中，
IO放置步驟不需要執行隨機放置，而是直接複製前一步的結果。

Step 3:

`ifeq($(MACRO_PLACEMENT)$(MACRO_PLACEMENT_TCL)$(RTLMP_FLOW),)`: 這是一個條件判斷，檢查(MACRO_PLACEMENT)、(
MACRO_PLACEMENT_TCL)和RTLMP_FLOW是否都未定義。如果未定義，
表示不使用巨集放置。

`$(eval $(call do-step, ...))`:如果不使用macro placement, 呼叫do-step模板來產
生執行步驟的規則。這裡的步驟名稱為"2_3_floorplan_tdms"，依賴項包括
"2_2_floorplan_io.odb"(前一步的結果)、"1_synth.v"、"1_synth.sdc"和庫檔(
LIB_FILES)，執行的TCL腳本為"tdms_place"。

`else`: 如果macro placement相關的變數已定義。

`$(eval $(call do-copy, ...))`: 如果使用macro placement, 呼叫do-copy模板產生規
則，用於複製"2_2_floorplan_io.odb"到目前步驟的目標。這是因為在巨集放置
的情況下，不需要執行混合大小放置，而是直接複製前一步驟的結果。

透過這段程式碼，實現了根據是否啟用macro placement選擇執行混合大小放置
或直接複製前一步驟的結果。

Step 4:

第一個`$(eval $(call do-step, ...))`: 呼叫do-step模板產生執行macro placement的
規則。步驟名稱為"2_4_floorplan_macro"，依賴項包括
"2_3_floorplan_tdms.odb"(前一步驟的結果)、"1_synth.v"、"1_synth.sdc"、

macro placement相關檔案(MACRO_PLACEMENT和MACRO_PLACEMENT_TCL)，執行的macro placement相關檔案為"macro_place"。

第二個\$(eval \$(call do-step, ...))：呼叫do-step模板產生執行macro placement的偵錯規則。步驟名稱為"2_floorplan_debug_macros"，相依性包括"2_1_floorplan.odb"(先前的步驟結果)、"1_synth.v"、macro placement相關檔案(MACRO_PLACEMENT和MACRO_PLACEMENT_TCL)，執行的TCL腳本為"floorplan_debug_macrosplan"。

透過這段程式碼，實現了macro placement的執行以及相關的調試規則。

Step 5:

\$(eval \$(call do-step, ...))：呼叫do-step模板產生插入Tapcell和Welltie的規則。步驟名稱為"2_5_floorplan_tapcell"，依賴項包括"2_4_floorplan_macro.odb"(前步驟的結果)和Tapcell相關的TCL腳本(TAPCELL_TCL)，執行的TCL腳本為"tapcell"。

透過這段程式碼，實現了插入Tapcell和Welltie的操作。

Step 6:

第一列是透過呼叫do-step模板產生PDN產生的規則。步驟名稱為"2_6_floorplan_pdn"，依賴項包括"2_5_floorplan_tapcell.odb"(前步驟的結果)和PDN相關的TCL腳本(PDN_TCL)，執行的TCL腳本為"pdn"。

第二列是透過呼叫do-copy模板產生複製檔案的規則。目標檔案是"2_floorplan.odb"，原始檔案是"2_6_floorplan_pdn.odb"(目前步驟的結果)。

透過這段程式碼，實現了PDN生成，並將結果複製到"2_floorplan.odb"。

在makefile中flopping的最後部分：

```
$(RESULTS_DIR)/2_floorplan.sdc: $(RESULTS_DIR)/2_1_floorplan.odb  
定義目標檔案"2_floorplan.sdc"，它依賴"2_1_floorplan.odb"檔案。
```

接著定義了一個偽目標"do-floorplan"，用於執行與floorplan相關的一系列步驟。這些步驟包括產生floorplan、IO放置、timing driven mixed-size placement、macro placement、Tapcell和Welltie插入、PDN產生以及複製檔案。執行這些步驟前，會建立日誌目錄和報表目錄。

最後則是定義了一個偽目標"clean_floorplan"，用於清理與floorplan相關的生成檔案和日誌。刪除了與floorplan步驟相關的odb、sdc、v、def文件，以及報告文件。

接下來看floorplan.tcl的細部內容

```
> #Run check_setup
> puts "\n===="
> puts "Floorplan check_setup"
> puts "-----"
> check_setup
>
> set num_instances [llength [get_cells -hier *]]
> puts "number instances in verilog is $num_instances"
>
```

check_setup 步驟用於驗證 floorplan 程序的設定是否正確。

check_setup 是一個檢查步驟，用於驗證 floorplan 程序的設定是否符合要求。

輸出一些用於標識和區分輸出資訊的文字。

"====" 和"Floorplan check_setup" 之間的文字是為了提供資訊的可讀性。

使用 get_cells 指令取得 Verilog 中的實例數量。

輸出 Verilog 檔案中的實例數量。

```
# Initialize floorplan by reading in floorplan DEF
#
if {[info exists ::env(FLOORPLAN_DEF)]} {
    puts "Read in Floorplan DEF to initialize floorplan: $env(FLOORPLAN_DEF)"
    read_def -floorplan_initialize $env(FLOORPLAN_DEF)
# Initialize floorplan using ICeWall FOOTPRINT
#
} elseif {[info exists ::env(FOOTPRINT)]} {
    ICeWall load_footprint $env(FOOTPRINT)

    initialize_floorplan \
        -die_area  [ICeWall get_die_area] \
        -core_area [ICeWall get_core_area] \
        -site      $::env(PLACE_SITE)

    ICeWall init_footprint $env(SIG_MAP_FILE)
```

透過讀取 floorplan DEF 進行初始化

如果環境變數 FLOORPLAN_DEF 存在，腳本將輸出一個訊息，指示正在讀取 floorplan DEF 檔案以初始化 floorplan。

使用 read_def 指令執行 floorplan 初始化，其中 -floorplan_initialize 選項指示初始化 floorplan。

如果環境變數 FOOTPRINT 存在，腳本將載入 ICeWall FOOTPRINT。

使用 ICeWall 提供的函數獲取 die 區域和核心區域的信息，並透過 initialize_floorplan 指令初始化 floorplan。

initialize_floorplan 指令的參數包括 die 區域、核心區域和站點資訊。

如果以上兩個條件都不滿足，腳本會嘗試使用 ICeWall 初始腳本初始化 FOOTPRINT。

使用 ICeWall 提供的 init_footprint 函數，並傳遞 SIG_MAP_FILE 作為參數。

這段腳本的目的是根據不同的條件選擇不同的初始化方式，以確保 floorplan 設定正確且完整。

```

# Initialize floorplan using CORE_UTILIZATION
#
} elseif {[info exists ::env(CORE_UTILIZATION)] && $::env(CORE_UTILIZATION) != "" } {
    set aspect_ratio 1.0
    if {[info exists ::env(CORE_ASPECT_RATIO)] && $::env(CORE_ASPECT_RATIO) != ""} {
        set aspect_ratio $::env(CORE_ASPECT_RATIO)
    }
    set core_margin 1.0
    if {[info exists ::env(CORE_MARGIN)] && $::env(CORE_MARGIN) != ""} {
        set core_margin $::env(CORE_MARGIN)
    }
    initialize_floorplan -utilization $::env(CORE_UTILIZATION) \
        -aspect_ratio $aspect_ratio \
        -core_space $core_margin \
        -sites $::env(PLACE_SITE)

```

基於 CORE_UTILIZATION 使用特定的參數初始化 floorplan:

如果環境變數 CORE_UTILIZATION 存在且不為空, 執行下列操作。

設定 `aspect_ratio` 預設值為 1.0。

設定 `core_margin` 預設值為 1.0。

如果環境變數 CORE_ASPECT_RATIO 存在且不為空, 將 aspect_ratio 設定為該值。

如果環境變數 CORE_MARGIN 存在且不為空, 將 core_margin 設定為該值。

使用 initialize_floorplan 指令進行 floorplan 初始化。

-utilization \$::env(CORE_UTILIZATION): 使用 CORE_UTILIZATION 提供的值進行初始化。

-aspect_ratio \$aspect_ratio 使用設定的 aspect_ratio。

-core_space \$core_margin: 使用設定的 core_margin。

-sites \$::env(PLACE_SITE): 使用 PLACE_SITE 提供的站點資訊。

```

# Initialize floorplan using DIE_AREA/CORE_AREA
#
} else {
    initialize_floorplan -die_area $::env(DIE_AREA) \
        -core_area $::env(CORE_AREA) \
        -sites $::env(PLACE_SITE)
}

if { [info exists ::env(MAKE_TRACKS)] } {
    source $::env(MAKE_TRACKS)
} elseif {[file exists $::env(PLATFORM_DIR)/make_tracks.tcl]} {
    source $::env(PLATFORM_DIR)/make_tracks.tcl
} else {
    make_tracks
}

if {[info exists ::env(FOOTPRINT_TCL)]} {
    source $::env(FOOTPRINT_TCL)
}

# remove buffers inserted by yosys/abc
remove_buffers

```

如果環境變數 CORE_UTILIZATION 不存在或為空，執行 floorplan 初始：

-die_area \$::env(DIE_AREA): 使用 DIE_AREA 提供的值進行初始化。
-core_area \$::env(CORE_AREA): 使用 CORE_AREA 提供的值進行初始化。
-sites \$::env(PLACE_SITE): 使用 PLACE_SITE 提供的站點資訊。

如果存在環境變數 MAKE_TRACKS，執行該變數指定的腳本。
否則，如果存在檔案 \$::env(PLATFORM_DIR)/make_tracks.tcl，執行該檔案。
如果上述兩者都不存在，執行預設的 make_tracks。
處理額外的腳本：

如果存在環境變數 FOOTPRINT_TCL，執行該變數指定的腳本。

呼叫 remove_buffers 命令，該命令可能用於移除 Yosys/ABC 工具插入的緩衝。

```
##### Restructure for timing #####

```

Restructure of timing:

檢查環境變數 RESYNTH_TIMING_RECOVER 是否存在且為 1。
如果需要重新結構，則呼叫 repair_design 和 repair_timing 指令，修復設計和時序。
輸出在重新結構之前的面積和時序報告（理想時脈情況）。
如果未設定 save_checkpoint 或 save_checkpoint 為真，將設計輸出到 Verilog 檔案 2_pre_abc_timing.v 中。

使用 restructure 指令，目標為時序 (-target timing)。
使用給定的 Liberty 檔案 (-liberty_file \$::env(DONT_USE_SC_LIB))。
指定工作目錄為 \$::env(RESULTS_DIR)。
輸出在重新結構之後的 Verilog 檔案：

如果未設定 save_checkpoint 或 save_checkpoint 為真，將重新結構之後的設計輸出到 Verilog 檔案 2_post_abc_timing.v 中。

```

# post restructure area/timing report (ideal clocks)
remove_buffers
repair_design
repair_timing

puts "Post restructure-opt wns"
report_worst_slack -max -digits 3
puts "Post restructure-opt tns"
report_tns -digits 3

# remove buffers inserted by optimization
remove_buffers
}

```

進行完 Restructure of timing 之後，執行一些後續操作

呼叫 remove_buffers 指令，從設計中移除由最佳化工具插入的緩衝器。

呼叫 repair_design 和 repair_timing 指令，對設計進行修復，確保重新結構後的設計仍然符合時序要求。

輸出 Post restructure-opt wns，即worst negative slack。

輸出 Post restructure-opt tns，即total negative slack。

重複一次呼叫 remove_buffers 指令，確保在後續最佳化中插入的任何緩衝器都被移除。

```

110 puts "Default units for flow"
111 report_units
112 report_units_metric
113 source ${::env(SCRIPTS_DIR)}/report_metrics.tcl
114 report_metrics 2 "floorplan final" false false
115
116 if { [info exist ::env(RESYNTH_AREA_RECOVER)] && ${::env(RESYNTH_AREA_RECOVER)} == 1 } {
117
118     utl::push_metrics_stage "floorplan_{}_pre_restruct"
119     set num_instances [llength [get_cells -hier *]]
120     puts "number instances before restructure is $num_instances"
121     puts "Design Area before restructure"
122     report_design_area
123     report_design_area_metrics
124     utl::pop_metrics_stage
125
126     if {![$info exists save_checkpoint] || $save_checkpoint} {
127         write_verilog ${::env(RESULTS_DIR)}/2_pre_abc.v
128     }
129
130     set tielo_cell_name [lindex ${env(TIELO_CELL_AND_PORT)} 0]
131     set tielo_lib_name [get_name [get_property [lindex [get_lib_cell $tielo_cell_name] 0] library]]
132     set tielo_port ${tielo_lib_name}/${tielo_cell_name}/[lindex ${env(TIELO_CELL_AND_PORT)} 1]
133
134     set tiehi_cell_name [lindex ${env(TIEHI_CELL_AND_PORT)} 0]
135     set tiehi_lib_name [get_name [get_property [lindex [get_lib_cell $tiehi_cell_name] 0] library]]
136     set tiehi_port ${tiehi_lib_name}/${tiehi_cell_name}/[lindex ${env(TIEHI_CELL_AND_PORT)} 1]
137
138     restructure -liberty_file ${::env(DONT_USE_SC_LIB)} -target "area" \
139         -tiehi_port $tiehi_port \
140         -tielo_port $tielo_port \
141         -work_dir ${::env(RESULTS_DIR)}
142

```

使用 puts 輸出 "Default units for flow"，然後呼叫 report_units 和

report_units_metric 指令。

呼叫 source 指令載入腳本檔案 \$::env(SCRIPTS_DIR)/report_metrics.tcl, 然後呼叫 report_metrics 指令, 報告與指定設計階段相關的指標。這些指標將包括資源使用、功耗等。

如果存在環境變數 RESYNTH_AREA_RECOVER, 且其值為1, 表示需要在面積恢復之前執行一些操作。

使用 utl::push_metrics_stage 和 utl::pop_metrics_stage 命令, 將當前指標的階段推送和彈出, 以便後續的指標報告能夠區分這一階段。

取得設計中實例的數量, 並輸出 "number instances before restructure is \$num_instances"。

輸出 Design Area before restructure, 然後呼叫 report_design_area 和 report_design_area_metrics 指令, 報告設計的面積資訊和相關指標。

如果環境變數 save_checkpoint 不存在或為真, 則使用 write_verilog 指令將設計儲存為 Verilog 檔案。這是在面積恢復之前的備份。

取得環境變數中 TieLO 和 TieHI 單元的信息, 包括單元名稱、庫名稱和連接埠資訊。

使用 restructure 指令, 目標為 "area", 在工作目錄為 \$::env(RESULTS_DIR) 下進行重新結構。這可能涉及移動和重新佈局單元, 以優化設計的面積。

```
# remove buffers inserted by abc
remove_buffers

if {![$info exists save_checkpoint] || $save_checkpoint} {
    write_verilog $::env(RESULTS_DIR)/2_post_abc.v
}
utl::push_metrics_stage "floorplan_{$_}_post_restruct"
set num_instances [llength [get_cells -hier *]]
puts "number instances after restructure is $num_instances"
puts "Design Area after restructure"
report_design_area
report_design_area_metrics
utl::pop_metrics_stage
}

if {[$info exists ::env(POST_FLOORPLAN_TCL)] } {
    source $::env(POST_FLOORPLAN_TCL)
}

if {![$info exists save_checkpoint] || $save_checkpoint} {
    if {[${info exists ::env(GALLERY_REPORT)}] && ${::env(GALLERY_REPORT)} != 0} {
        write_def ${::env(RESULTS_DIR)}/2_1_floorplan.def
    }
    write_db ${::env(RESULTS_DIR)}/2_1_floorplan.odb
    write_sdc ${::env(RESULTS_DIR)}/2_floorplan.sdc
}
```

使用 remove_buffers 指令, 移除由ABC插入的緩衝器。

如果環境變數 save_checkpoint 不存在或為真, 則使用 write_verilog 指令將設計儲存為 Verilog 檔案。這是在面積恢復之後的備份。

使用 `utl::push_metrics_stage` 和 `utl::pop_metrics_stage` 命令，將當前指標的階段推送和彈出，以便後續的指標報告能夠區分這一階段。

取得設計中實例的數量，並輸出 "number instances after restructure is \$num_instances"。

輸出 "Design Area after restructure"，然後呼叫 `report_design_area` 和 `report_design_area_metrics` 指令，報告設計的面積資訊和相關指標。

如果環境變數 `POST_FLOORPLAN_TCL` 存在，則使用 `source` 指令執行使用者定義的腳本。

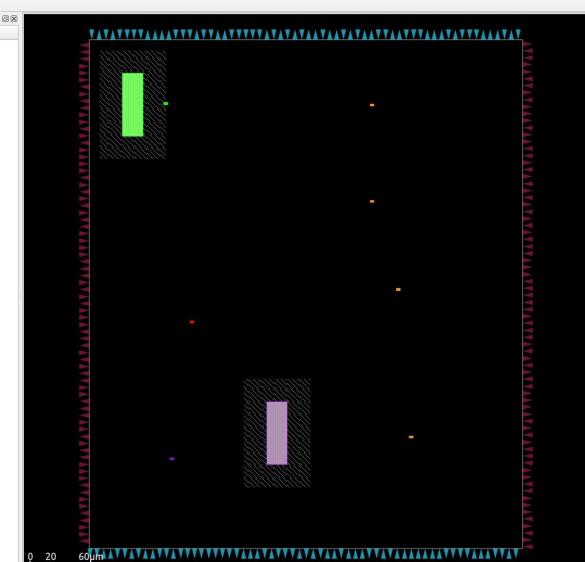
如果環境變數 `save_checkpoint` 不存在或為真，且存在環境變數 `GALLERY_REPORT` 且其值不為0，則使用 `write_def` 指令將設計儲存為 DEF 檔案。

`write_db` 指令將設計儲存為 ODB 檔案，`write_sdc` 指令將設計儲存為 SDC 檔案。

參數調整：

`aspect_ratio`: 此參數通常表示 floorplan 中核心區域寬度與高度的比例。將其設為 0.5 表示你指定寬度是高度的一半，即核心區域相對於其高度來說會更寬。

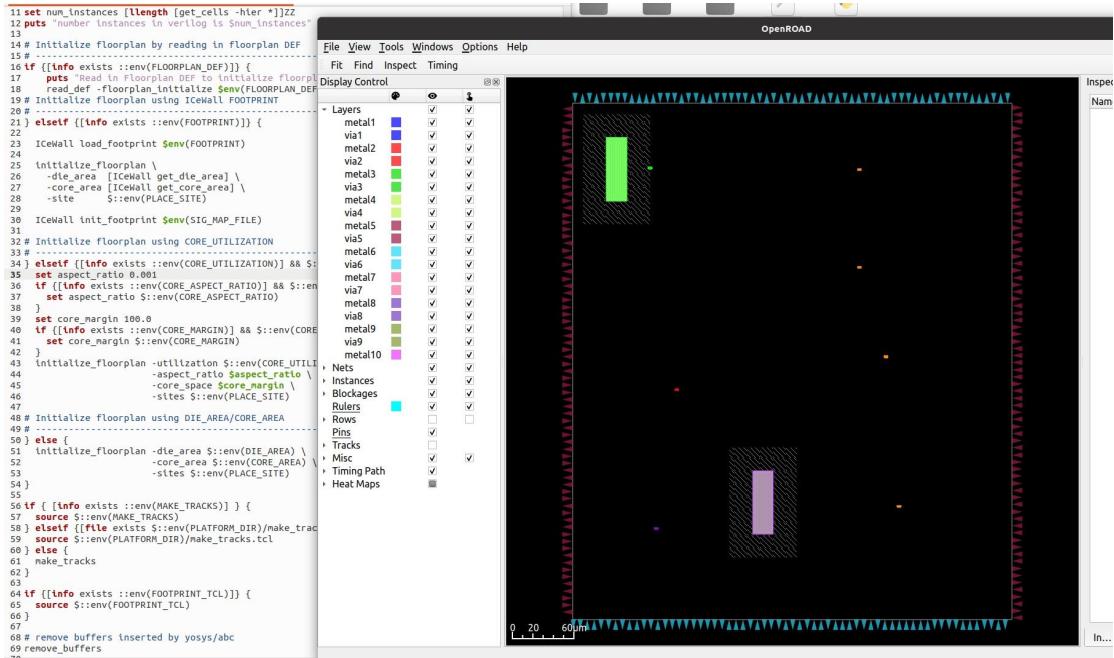
`core_margin`: 此參數表示核心區域周圍的空間。



The screenshot shows a floorplanning interface with a central core area. The core is a green rectangle with a width-to-height ratio of 1.0. It is surrounded by a white margin of width 1.0. The entire design area is bounded by a black frame. A legend on the right indicates layer settings for metal1 through metal10, nets, instances, blockages, and other parameters. A scale bar at the bottom left shows 0, 20, and 60 micrometers.

```
14 # Initialize floorplan by reading in Floorplan DEF
15 #
16 if {[info exists ::env(FLOORPLAN_DEF)]} {
17     puts "Read in Floorplan DEF to initialize floorplan: $env(FLOORPLAN_DEF)"
18     read_def -floorplan_initialize $env(FLOORPLAN_DEF)
19 }
20 # Initialize floorplan using ICeWall footprint
21 #
22 if {[info exists ::env(FOOTPRINT)]} {
23     ICeWall Load_footprint $env(FOOTPRINT)
24 }
25 # Initialize floorplan
26 -die_area [ICeWall get_die_area] \
27 -core_area [ICeWall get_core_area] \
28 -site $::env(PLACE_SITE)
29
30 ICeWall Int_footprint $env(SIG_MAP_FILE)
31
32 # Initialize floorplan using CORE_UTILIZATION
33 #
34 if {[info exists ::env(CORE_UTILIZATION)] & $::env(CORE_UTILIZATION) > 0} {
35     set aspect_ratio 1.0
36     if {[info exists ::env(CORE_ASPECT_RATIO)] && $::env(CORE_ASPECT_RATIO) > 0} {
37         set aspect_ratio $::env(CORE_ASPECT_RATIO)
38     }
39     set core_margin 1.0
40     if {[info exists ::env(CORE_MARGIN)] && $::env(CORE_MARGIN) > 0} {
41         set core_margin $::env(CORE_MARGIN)
42     }
43     initialize_floorplan -utilization $::env(CORE_UTILIZATION) \
44             -aspect_ratio $aspect_ratio \
45             -core_margin $core_margin \
46             -sites $::env(PLACE_SITE)
47 }
48 # Initialize floorplan using DIE_AREA/CORE_AREA
49 #
50 if {[info exists ::env(DIE_AREA)] & $::env(DIE_AREA) != ""} {
51     initialize_floorplan -die_area $::env(DIE_AREA) \
52             -core_area $::env(CORE_AREA) \
53             -sites $::env(PLACE_SITE)
54 }
55
56 if {[info exists ::env(MAKE_TRACKS)]} {
57     source $::env(MAKE_TRACKS)
58 } else {
59     if {[file exists $::env(PLATFORM_DIR)/make_tracks.tcl]} {
60         source $::env(PLATFORM_DIR)/make_tracks.tcl
61     } else {
62         make_tracks
63     }
64 if {[info exists ::env(FOOTPRINT_TCL)]} {
65     source $::env(FOOTPRINT_TCL)
66 }
67
68 # remove buffers inserted by yosys/abc
```

上圖為 `aspect_ratio` 設為 1.0 和 `core margin` 設為 1.0 時的模擬情形



上圖為 aspect_ratio 設為 0.001 和 core magin 設為 100.0 時的模擬情形，可以看到核心區域更寬，周圍的空間間隔更大。

```

1 # settings.mk is not under source control. Put variables into this
2 # file to avoid having to add the to the make command line.
3 -include settings.mk
4
5 # =====
6 # Uncomment or add the design to run
7 # =====
8
9 # DESIGN_CONFIG=./designs/nangate45/aes/config.mk
10 # DESIGN_CONFIG=./designs/nangate45/ariane133/config.mk
11 # DESIGN_CONFIG=./designs/nangate45/ariane136/config.mk
12 # DESIGN_CONFIG=./designs/nangate45/black_parrot/config.mk
13 # DESIGN_CONFIG=./designs/nangate45/bp_be_top/config.mk
14 # DESIGN_CONFIG=./designs/nangate45/bp_fe_top/config.mk
15 # DESIGN_CONFIG=./designs/nangate45/bp_multi_top/config.mk
16 # DESIGN_CONFIG=./designs/nangate45/bp_quad/config.mk
17 # DESIGN_CONFIG=./designs/nangate45/dynamic_node/config.mk
18 # DESIGN_CONFIG=./designs/nangate45/gcd/config.mk
19 # DESIGN_CONFIG=./designs/nangate45/ibex/config.mk
20 # DESIGN_CONFIG=./designs/nangate45/jpeg/config.mk
21 # DESIGN_CONFIG=./designs/nangate45/mempool_group/config.mk
22 # DESIGN_CONFIG=./designs/nangate45/swerv/config.mk
23 # DESIGN_CONFIG=./designs/nangate45/swerv_wrapper/config.mk
24 DESIGN_CONFIG=./designs/nangate45/tinyRocket/config.mk
25 |
26 # DESIGN_CONFIG=./designs/ttmcfg10/noc/config.mk

```

將 config 從預設的 gcd 換成 nangate45/tinyrocket 看看

```

14 # Initialize Floorplan by reading in floorplan DEF
15 #
16 tf {[info exists ::env(FLOORPLAN_DEF)]}
17 puts "Read in Floorplan DEF To Initialize Floorplan"
18 read_def -floorplan_initialize $env(FLOORPLAN_DEF)
19 # Initialize floorplan using ICewall FLOORPRINT
20 #
21 ) elseif {[info exists ::env(FOOTPRINT)}]
22 ) else
23 ) Icewall_load_footprint $env(FOOTPRINT)
24 #
25 initialize_floorplan
26 -die_area [Icewall get_die_area] \
27 -core_area [Icewall get_core_area] \
28 -site $env(PLACE_SITE)
29
30 Icewall_init_footprint $env(SIG_MAP_FILE)
31
32 # Initialize Floorplan using CORE_UTILIZATION
33 #
34 ) else
35 ) if {[info exists ::env(CORE_UTILIZATION)] && $::env(CORE_UTILIZATION) != ""}
36 ) set aspect_ratio 1.0
37 ) if {[info exists ::env(CORE_ASPECT_RATIO)] && $::env(CORE_ASPECT_RATIO) != ""}
38 ) set aspect_ratio $::env(CORE_ASPECT_RATIO)
39 ) set core_margin 1.0d
40 ) if {[info exists ::env(CORE_MARGIN)] && $::env(CORE_MARGIN) != ""}
41 ) set core_margin $::env(CORE_MARGIN)
42 )
43 initialize_floorplan -utilization $::env(CORE_UTILIZATION) \
44 -aspect_ratio $aspect_ratio \
45 -core_margin $core_margin \
46 -site $env(PLACE_SITE)
47
48 # Initialize floorplan using DIE/AREA/CORE/AREA
49 #
50 ) else (
51 initialize_floorplan -die_area $::env(DIE_AREA) \
52 -core_area $::env(CORE_AREA) \
53 -site $env(PLACE_SITE)
54 )
55
56 tf {[info exists ::env(MAKE_TRACKS)]} {
57 source $::env(MAKE_TRACKS)
58 ) elseif {[file exists $::env(PLATFORM_DIR)/make_tracks.tcl]}
59 source $::env(PLATFORM_DIR)/make_tracks.tcl
60 ) else {
61 make_tracks
62 }
63
64 tf {[info exists ::env(FOOTPRINT_TCL)]} {
65 source $::env(FOOTPRINT_TCL)
66 }
67
68 # remove buffers inserted by yosys/abc
69 remove_buffers

```

在tinyrocket下執行make gui_2_4_floorplan_macro.odb
可以看到macro和其他instance之間的區別，他的外圍有包圍灰色的細部元件

```

13 puts "number Instances in verilog is $num_instances"
14 #
15 # Initialize Floorplan by reading in floorplan DEF
16 tf {[info exists ::env(FLOORPLAN_DEF)]}
17 puts "Read in Floorplan DEF To Initialize Floorplan: $env(FLOORPLAN_DEF)"
18 read_def -floorplan_initialize $env(FLOORPLAN_DEF)
19 # Initialize floorplan using ICewall FLOORPRINT
20 #
21 ) elseif {[info exists ::env(FOOTPRINT)}]
22 ) else
23 ) Icewall_load_footprint $env(FOOTPRINT)
24 #
25 initialize_floorplan
26 -die_area [Icewall get_die_area] \
27 -core_area [Icewall get_core_area] \
28 -site $env(PLACE_SITE)
29
30 Icewall_init_footprint $env(SIG_MAP_FILE)
31
32 # Initialize Floorplan using CORE_UTILIZATION
33 #
34 ) else
35 ) if {[info exists ::env(CORE_UTILIZATION)] && $::env(CORE_UTILIZATION) != ""}
36 ) set aspect_ratio 0.8m
37 ) if {[info exists ::env(CORE_ASPECT_RATIO)] && $::env(CORE_ASPECT_RATIO) != ""}
38 ) set aspect_ratio $::env(CORE_ASPECT_RATIO)
39 ) set core_margin 1.0d
40 ) if {[info exists ::env(CORE_MARGIN)] && $::env(CORE_MARGIN) != ""}
41 ) set core_margin $::env(CORE_MARGIN)
42 )
43 initialize_floorplan -utilization $::env(CORE_UTILIZATION) \
44 -aspect_ratio $aspect_ratio \
45 -core_margin $core_margin \
46 -site $env(PLACE_SITE)
47
48 # Initialize floorplan using DIE/AREA/CORE/AREA
49 #
50 ) else (
51 initialize_floorplan -die_area $::env(DIE_AREA) \
52 -core_area $::env(CORE_AREA) \
53 -site $env(PLACE_SITE)
54 )
55
56 tf {[info exists ::env(MAKE_TRACKS)]} {
57 source $::env(MAKE_TRACKS)
58 ) elseif {[file exists $::env(PLATFORM_DIR)/make_tracks.tcl]}
59 source $::env(PLATFORM_DIR)/make_tracks.tcl
60 ) else {
61 make_tracks
62 }
63
64 tf {[info exists ::env(FOOTPRINT_TCL)]} {
65 source $::env(FOOTPRINT_TCL)
66 }
67
68 # remove buffers Inserted by yosys/abc
69 remove_buffers
70

```

上圖在tinyrocket下make gui_floorplan

```

11 set num_instances [llength [get_cells -hier *]]*2
12 puts "number Instances in verilog is $num_instances"
13
14 #
15 # Initialize Floorplan by reading in floorplan DEF
16 tf {[info exists ::env(FLOORPLAN_DEF)]}
17 puts "Read in Floorplan DEF To Initialize Floorplan"
18 read_def -floorplan_initialize $env(FLOORPLAN_DEF)
19 # Initialize floorplan using ICewall FLOORPRINT
20 #
21 ) elseif {[info exists ::env(FOOTPRINT)}]
22 ) else
23 ) Icewall_load_footprint $env(FOOTPRINT)
24 #
25 initialize_floorplan
26 -die_area [Icewall get_die_area] \
27 -core_area [Icewall get_core_area] \
28 -site $env(PLACE_SITE)
29
30 Icewall_init_footprint $env(SIG_MAP_FILE)
31
32 # Initialize Floorplan using CORE_UTILIZATION
33 #
34 ) else
35 ) if {[info exists ::env(CORE_UTILIZATION)] && $::env(CORE_UTILIZATION) != ""}
36 ) set aspect_ratio 1.0
37 ) if {[info exists ::env(CORE_ASPECT_RATIO)] && $::env(CORE_ASPECT_RATIO) != ""}
38 ) set aspect_ratio $::env(CORE_ASPECT_RATIO)
39 ) set core_margin 1.0d
40 ) if {[info exists ::env(CORE_MARGIN)] && $::env(CORE_MARGIN) != ""}
41 ) set core_margin $::env(CORE_MARGIN)
42 )
43 initialize_floorplan -utilization $::env(CORE_UTILIZATION) \
44 -aspect_ratio $aspect_ratio \
45 -core_margin $core_margin \
46 -site $env(PLACE_SITE)
47
48 # Initialize floorplan using DIE/AREA/CORE/AREA
49 #
50 ) else (
51 initialize_floorplan -die_area $::env(DIE_AREA) \
52 -core_area $::env(CORE_AREA) \
53 -site $env(PLACE_SITE)
54 )
55
56 tf {[info exists ::env(MAKE_TRACKS)]} {
57 source $::env(MAKE_TRACKS)
58 ) elseif {[file exists $::env(PLATFORM_DIR)/make_tracks.tcl]}
59 source $::env(PLATFORM_DIR)/make_tracks.tcl
60 ) else {
61 make_tracks
62 }
63
64 tf {[info exists ::env(FOOTPRINT_TCL)]} {
65 source $::env(FOOTPRINT_TCL)
66 )
67
68 # remove buffers Inserted by yosys/abc
69 remove_buffers

```

和gcd一樣奇數排是VDD, 偶數排是VSS, 不同的是, 可以發現粉紅色偏左的導線為VSS, 而偏右的為VDD。

Part 3 :

我覺得openroad能幫助我清楚地理解每一個流程在做的事情，並且能夠透過GUI可視化來幫助我理解流程。他還有一個優點是他是免費的，任何人都可以直接透過github下載。老師開的這堂課從synthesis一路到routing都有探討熱門的演算法，而從openroad也可以看出來一些相似的概念，利用這些演算法能夠幫助我們實現電路設計自動化。老師在CAD這堂課上面教學十分用心，受益良多。有一點小小的建議是希望老師能夠在作業截止後能夠分享一下作的最佳解法，像是project 3是透過kernel extraction和substitution來化簡literal count，我的方法可能很暴力也比較直觀，所以不清楚效能如何，如果老師能夠分享一下老師的做法，讓我們學生能參考一下，這樣或許能讓我們對作業的理解更加透徹。總而言之，我覺得上這堂課收穫很多，openroad也讓我更清楚理解整個design flow，十分感謝老師的教導。