如何使用 3DIC Compiler 实现芯片堆叠设计

前言

先进封装从 MCM 发展到 2.5D/3D 堆叠封装，目前发展最快的制造商是 TSMC。TSMC 从 Foundry 端延伸入 2.5D/3D 先进封装，称为 3D Fabric。近十年来 TSMC 的 2.5D 先进封装技术经历了 5 代更新，硅载板的面积已经达到 3 倍光罩尺寸，最新技术可以集成 8 个 HBM，支持 eDTC，同时改进了 TSV 和 TIM 材料，以及厚铜互连，对于 3D 堆叠的 PI、SI、Thermal 和 Stress 等方面都有了很好的的改善。与此同时 NVIDIA 的 A100 单芯片面积为 826mm2，逼近光罩尺寸极限，而大芯片面临低良率、成本增加的问题，这直接导致了 Chiplet 技术成为炙手可热的话题。TSMC 在 SoIC 的未来十年的计划中描述每两个相邻工艺节点的 bump pitch 缩小 70%，从而达到每工艺节点 2 倍带宽密度的扩展，以及 EEP（Energy Efficiency Performance)提升。

3D 堆叠的设计流程

3DIC 先进封装之前并没有统一的 EDA 设计流程和设计平台，Fabless 只能通过使用传统的脱节的点工具链和流程来进行 3DIC 的设计。由于数据切换的不连贯性和不一致性，以及不同点工具间的设计环境的改变，这些都对 3DIC 先进封装的设计收敛带来巨大的挑战。芯和半导体为了应对这些挑战，提出了自主可控的 3DIC 先进封装设计和仿真平台。其中 3D IC Compiler（以下简称“3DICC”）作为设计平台，涵盖 HBM 堆叠、异构逻辑芯片堆叠以及 logic 芯片上的 memory 堆叠流程，涉及 TSV 互连、basic logic die、中介层、局部硅桥等信息定义。3DICC 堆叠设计流程主要包含如下七个步骤：

1. Prepare design library
2. Create die designs
3. Create floorplan/bumps/TSVs
4. Routing TSVs to backside bumps
5. Create top-level design
6. Assign nets, bumps, and TSVs

7. Perform 3D check

1. 设计文件导入（DRAM NDM 创建）

首先将 3DIC 设计所需的 tf, ref_libs, csv 等文件设置到 tcl 脚本中, 创建所需 ndm 数据;

```
sh rm -rf DRAM.ndm

source scripts/dims.tcl
set_host_option -max_core 8
create_lib -tech ./input_data/die_bmetal.tf DRAM.ndm
set_ref_libs -ref_libs "bump_creation/die_bmetal_bumps.ndm"

create_block -dimensions "$dram_width $dram_height" DRAM \
-design_type die
read_design_io -file_name_list ./input_data/DRAM.csv -overlap_check none -power {VDD} -ground {VSS} -create_ports

create_bump_array -lib_cell UBUMP_30_HEX -name "DRAM_VDD_" -origin {85 135} -delta {300 300} -pattern inline
create_bump_array -lib_cell UBUMP_30_HEX -name "DRAM_VSS_" -origin {85 285} -delta {300 300} -pattern inline

#remove cell at data bump
remove_cells [get_cells -filter "name =~ *DRAM_V*" -within {{766 370} {1166 1070}}]
#prevent PG mesh at data bump
create_routing_blockage -layers {M8 M9 AP} -boundary {{926.00 530.00} {1036.00 940.00}}

#####################################################################
## Create PG mesh
#####################################################################
create_port VSS -port_type ground
create_port VDD -port_type power
create_net VSS -ground
create_net VDD -power
connect_net -net VSS [get_port VSS]
connect_net -net VDD [get_port VDD]

set_app_options -name plan.pgroute.disable_floating_removal -value true
set_app_options -name plan.pgroute.via_site_threshold -value 1.0
set_app_options -name plan.pgroute.maximize_total_cut_area -value all
set_app_options -name plan.pgroute.fix_via_drc_multiple_viadef -value true
set_app_options -name plan.pgroute.treat_fixed_stdcell_as_macro -value true
set_app_options -name plan.pgroute.patch_via_enclosure -value false
set_app_options -name plan.pgroute.realign_straps_for_cell_gap -value true

set_pg_strategy_via_rule NO_VIA  -via_rule { {{intersection: undefined}{via_master: NIL}} }
```

图 1：设置 package int bdie tdie dram 的高度宽度,block 的创建，blockage 的创建

```
set_pg_strategy_via_rule NO_VIA  -via_rule { {{intersection: undefined}{via_master: NIL}} }

create_pg_mesh_pattern PG_TOP_MESH1_PTRN \
    -layers { \
         {{horizontal_layer:M9}{width:5 }\
             {spacing:15}{pitch:150}{track_alignment: track}{trim: false}{offset:135}} \
         {{vertical_layer:M8}{width:5 }\
             {spacing:15}{pitch:150}{track_alignment: track}{trim: false}{offset:85}} \
             }\
    -via_rule {{intersection : all}{via_master:NIL}}

set_pg_strategy PG_TOP_MESH1_STR \
    -pattern { {name: PG_TOP_MESH1_PTRN}{nets: VDD VSS}} \
    -polygon {{5.0000 5.0000} {1995.0 1495.0}}

compile_pg -strategies PG_TOP_MESH1_STR -via_rule NO_VIA -tag M8_M9_MESH
# remove boundary VDD shape
remove_shape [get_shapes -within {{4.9500 1487.4500} {1995.0500 1492.5500}} -filter "layer_name == M9"]

set_app_options -name plan.pgroute.optimize_via_when_maximize_cutarea -value false
set_app_options -name plan.pgroute.fix_via_drc_multiple_viadef -value false
set_app_options -name plan.pgroute.treat_fixed_stdcell_as_macro -value true
set_app_options -name plan.pgroute.patch_via_enclosure -value true

create_pg_vias -from_layers M8 -to_layers M9 -nets {VDD VSS}
reset_app_options plan.pgroute.optimize_via_when_maximize_cutarea
reset_app_options plan.pgroute.fix_via_drc_multiple_viadef
reset_app_options plan.pgroute.treat_fixed_stdcell_as_macro
reset_app_options plan.pgroute.patch_via_enclosure

## Create PG bump via array
create_pg_bump_via_array -pg_net VDD  -pg_layer M9 -cells *DRAM_VDD_* -max_num 20
create_pg_bump_via_array -pg_net VSS  -pg_layer M9 -cells *DRAM_VSS_* -max_num 20

remove_routing_blockages -all

save_block; close_block

save_lib; close_lib
```

图 2：create PG mesh 设计定义，pg_mesh_pattern 的定义，options 的设置，Create PG bump via array 的设置

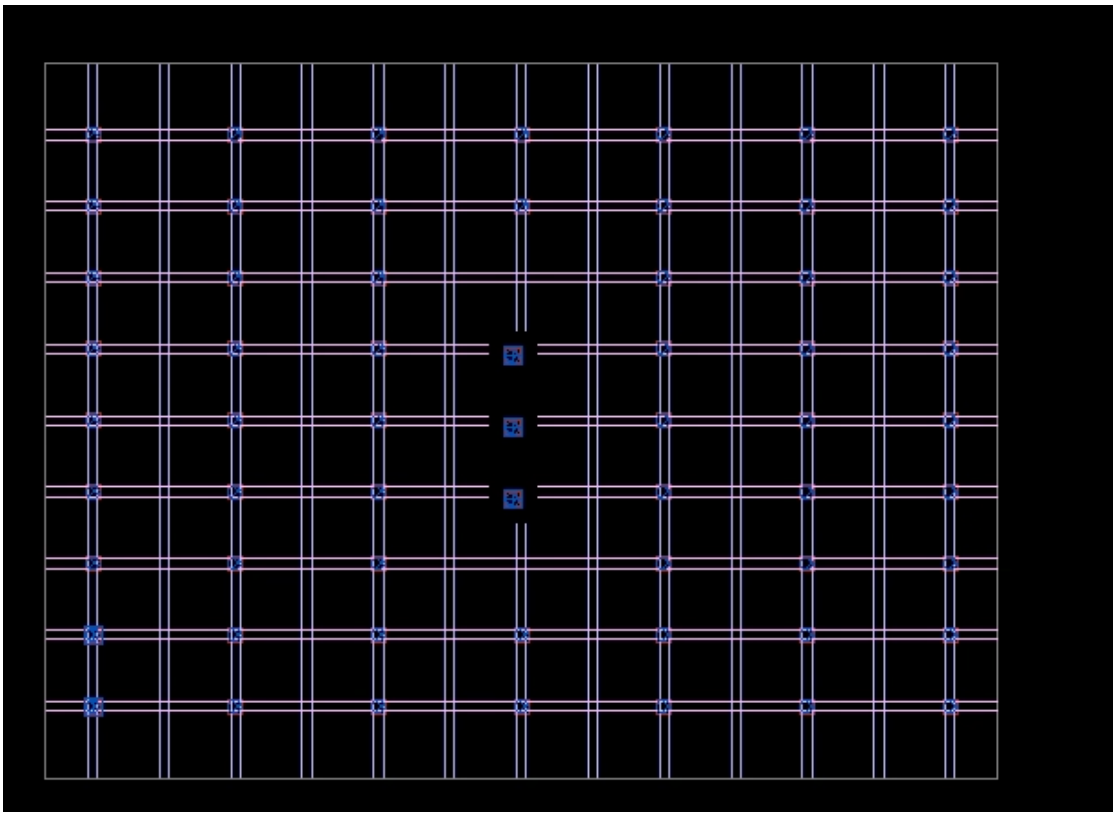

图 3：DRAM 2D 视图

2. Die_Stack interposer package 以及 top 的创建类似 DRAM 设置

Die_Stack 包含 base_die 和 top_die。

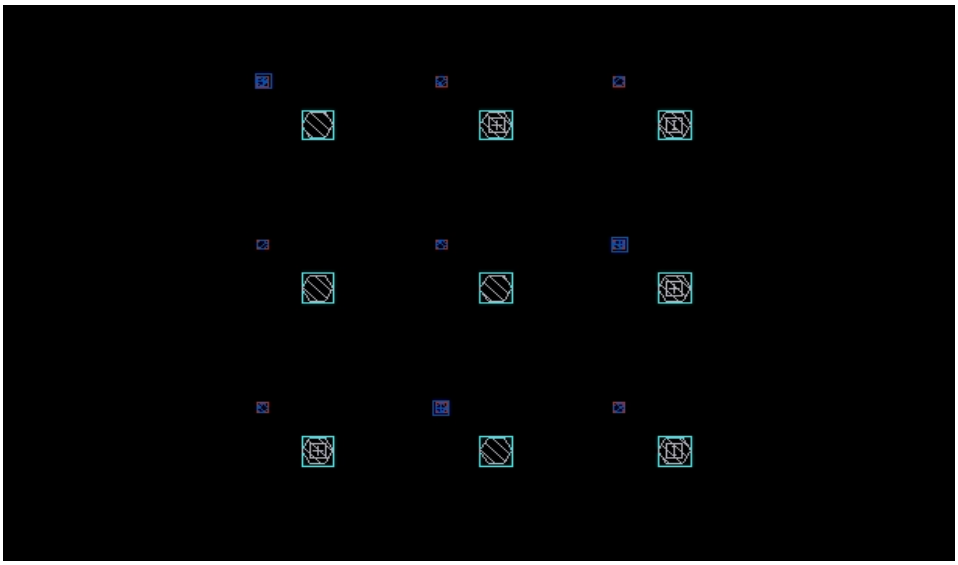

图 4：base_die.ndm

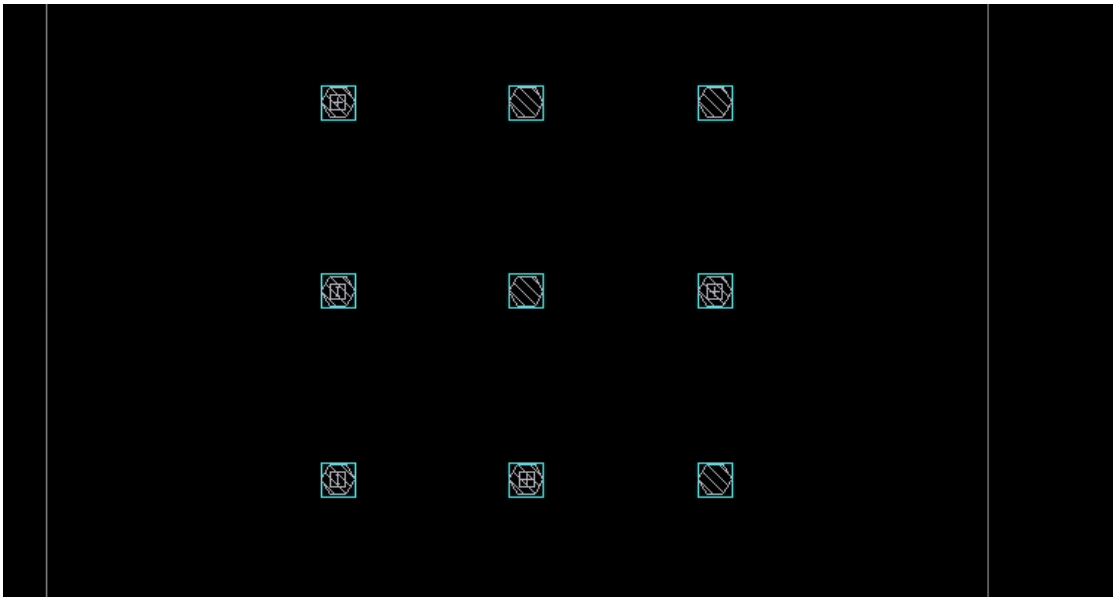

图 5：top_die.ndm

Interposer 和 package top 的 ndm 类似以上操作思路

3. Base_inst_pg 和 interposer_pg 连接的设置

下面介绍下电源线连接的设置，在 3dic_shell 中打开 base 和 interposer die database，接着连接 DRAM 和 Base die 的电源。

```
open_block base_die.ndm:base_die_i7

## connect top_die PG bumps
connect_net -net VDD [get_pins top_VDD_*/BUMP]
connect_net -net VSS [get_pins top_VSS_*/BUMP]

write_verilog basedie.post_assign.v

save_block; close_block; close_lib
```

图 6：connect_base_inst_pg 的设置

```
open_block interposer.ndm:interposer

### Create PG ports and nets
create_port {VDD} -port_type power
create_port {VSS} -port_type ground

create_net -power {VDD}
create_net -ground {VSS}

### Connect base die PG bumps
connect_net -net VDD [get_ports VDD]
connect_net -net VDD [get_pins base_VDD_*/BUMP]

connect_net -net VSS [get_ports VSS]
connect_net -net VSS [get_pins base_VSS_*/BUMP]

### Connect DRAM PG bumps
connect_net -net VDD [get_pins dram_DRAM_VDD_*/BUMP]

connect_net -net VSS [get_pins dram_DRAM_VSS_*/BUMP]
```

图 7：connect_interposer_pg 的设置

4. Create_interposer_c4bump 的创建

下图是设置 interposer 中 c4 的创建的定义，包含 name，lib_cell ，pattern，delta，origin
在 3dic_shell 窗口下打开 interposer database 进行操作

```
create_bump_array \
  -name "C4_VSS_" \
  -lib_cell C4BUMP_90_HEX \
  -boundary {{-3500.0000 -3500.0000} {3370.0000 3370.0000}} \
  -pattern inline \
  -delta {300 150} \
  -origin {300 150}

## Remove c4bump in HBM routing area (example only)
## Follow own bump planning
remove_cells [get_cells -filter "name =~ C4_*" -within {{-1750.0000 -700.0000} {2250.0000 700.0000}}]

############################################################

connect_net -net VDD [get_pins C4_VDD_*/BUMP]
connect_net -net VSS [get_pins C4_VSS_*/BUMP]

############################################################

### Auto create and place TSVs for C4 PG bumps
derive_3d_interface \
  -from [get_cell C4_VDD_*] \
  -to_object_ref [get_via_defs -tech [get_techs interposertech] VIAB1] \
  -name_prefix TSV_VDD
```

图 8：create_interposer_c4bump 的设置定义

5. Create_pg_routes 的创建

在 3dic_shell 中输入脚本中的设置

```
########################################################################
# PG routing
########################################################################

open_block interposer.ndm:interposer

## hbm routing region
create_routing_blockage -layers {M1 M2 M3 M4 AP} -boundary {{-1750.0000 -700.0000} {2190.0000 700.0000}}

##
set_pg_strategy_via_rule no_via_macro -via_rule {{macro_pins : all} {via_master: NIL}}
```

```
## M1 (VDD)
create_pg_wire_pattern VDD_M1_pt1 -direction vertical \
                                  -layer M1 \
                                  -width 24 \
                                  -spacing 20 \
                                  -pitch 300

create_pg_composite_pattern VDD_M1_pt2 -nets {VDD} \
        -add_patterns {{ {nets: {VDD}} {pattern: VDD_M1_pt1} {offset: 195 150} }}

set_pg_strategy VDD_M1_S1 \
        -pattern { {pattern: VDD_M1_pt2} {nets: VDD} } -design_boundary

compile_pg -strategies {VDD_M1_S1} -via_rule no_via_macro

## M1 (VSS)
create_pg_wire_pattern VSS_M1_pt1 -direction vertical \
                                  -layer M1 \
                                  -width 24 \
                                  -spacing 20 \
                                  -pitch 300

create_pg_composite_pattern VSS_M1_pt2 -nets {VSS} \
        -add_patterns {{ {nets: {VSS}} {pattern: VSS_M1_pt1} {offset: 345 300} }}

set_pg_strategy VSS_M1_S1 \
        -pattern { {pattern: VSS_M1_pt2} {nets: VSS} } -design_boundary

compile_pg -strategies {VSS_M1_S1} -via_rule no_via_macro
```

```
## remove extra vertical VSS without bump
remove_shapes [get_shapes -within {{3432.5000 -3500.5000} {3457.5000 3500.5000}} -filter "net_type == ground"]
## remove extra horizontal VSS without bump
remove_shapes [get_shapes -within {{-3500.5000 3432.5000} {3500.5000 3457.5000}} -filter "layer_name == M2"]

## VIA creation
create_pg_vias -nets {VDD VSS} -from_types stripe -to_types stripe -from_layers M2 -to_layers M1
create_pg_vias -nets {VDD VSS} -from_types stripe -to_types stripe -from_layers M3 -to_layers M2
create_pg_vias -nets {VDD VSS} -from_types stripe -to_types stripe -from_layers M4 -to_layers M3
```

图 9：pg_routes 的设置界面

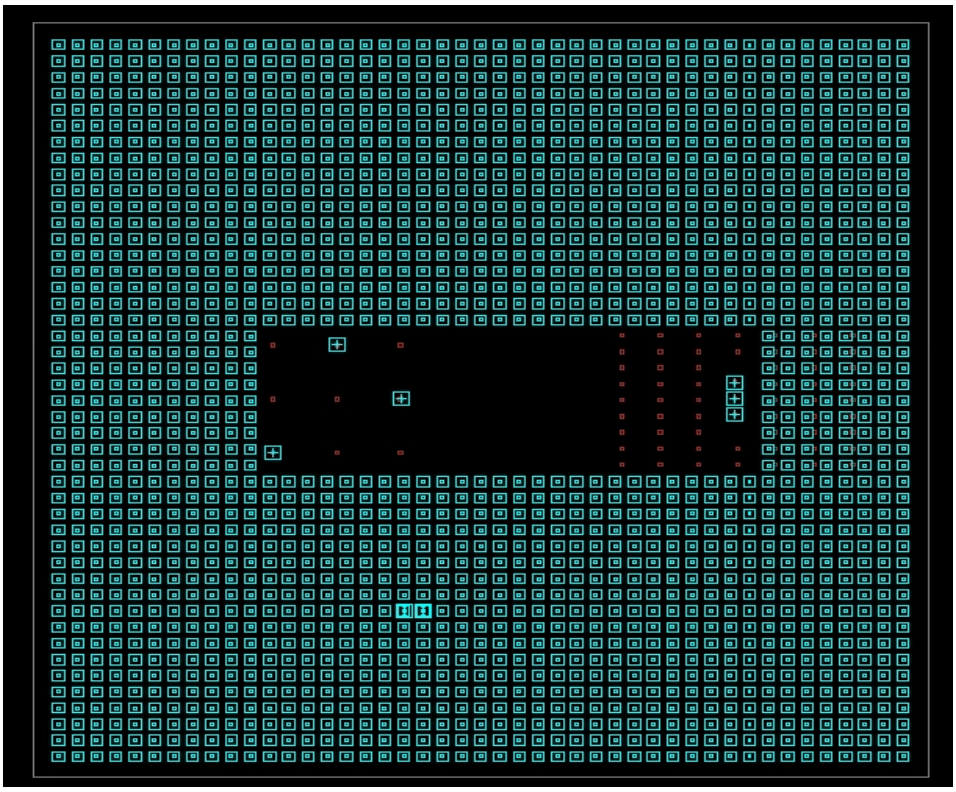

图 10：interposer 2D 视图界面

6. Mirror_bump_to_package

在 3dic_shell 里面打开之前创建的 top.ndm

Mirror_bump 从 interposer to NCPackage

```
create_3d_mirror_bumps -from interposer_inst -to NCPackage_inst \
    -ref_cell C4BUMP_90_HEX -prefix "interposer_"

save_block -hierarchical
close_lib -all

## connect Package PG
open_block NCPackage.ndm:NCPackage

## Create BGA bump in whole area
create_bump_array \
   -name "BGA_VDD_" \
   -lib_cell BGABUMP_500_64POLY \
   -boundary {{-4000.0000 -4000.0000} {4000.0000 4000.0000}} \
   -pattern inline \
   -delta {2000 1000} \
   -origin {250 250}

create_bump_array \
   -name "BGA_VSS_" \
   -lib_cell BGABUMP_500_64POLY \
   -boundary {{-4000.0000 -4000.0000} {4000.0000 4000.0000}} \
   -pattern inline \
   -delta {2000 1000} \
   -origin {1250 250}
```

图 11：mirror bump

7. Commit_upf_connect_top_pg

在 3dic_shell 里面打开之前所建立的 ndm database，比如：
Interposer.ndm　DRAM.ndm　base_die.ndm　top_die.ndm　NCPackage.ndm
读入各自的 upf 文件，再连接各自的电源
创建 top-level VDD/VSS ports and connect to interposer/die/package
创建 UPF for the top-level

```
### Create UPF for the top level system
create_power_domain PD

create_supply_net {VDD} -domain PD
create_supply_port {VDD}

create_supply_net {VSS} -domain PD
create_supply_port {VSS}

connect_supply_net -ports {VDD interposer_inst/VDD DRAM_inst/VDD base_inst/VDD top_die/VDD NCPackage_inst/VDD} VDD
connect_supply_net -ports {VSS interposer_inst/VSS DRAM_inst/VSS base_inst/VSS top_die/VSS NCPackage_inst/VSS} VSS

set_voltage -object [get_supply_nets VDD] 0.8
set_voltage -object [get_supply_nets VSS] 0

set_net_type -net VDD -type power
set_net_type -net VSS -type ground

set_domain_supply_net PD -primary_power_net VDD -primary_ground_net VSS

add_port_state VDD -state {HV 1.08} -state {LV 0.8}
add_port_state VSS -state {GND 0}
create_pst design_pst -supplies {VDD VSS}
add_pst_state HI -pst design_pst -state {HV GND}
add_pst_state LO -pst design_pst -state {LV GND}

commit_upf
```

图 12：upf 的 power 连接

8. 整体 3D 堆叠设计 2D 和 3D 效果示例图

在 3DIC 中设计的二维设计和三维立体视图的对比

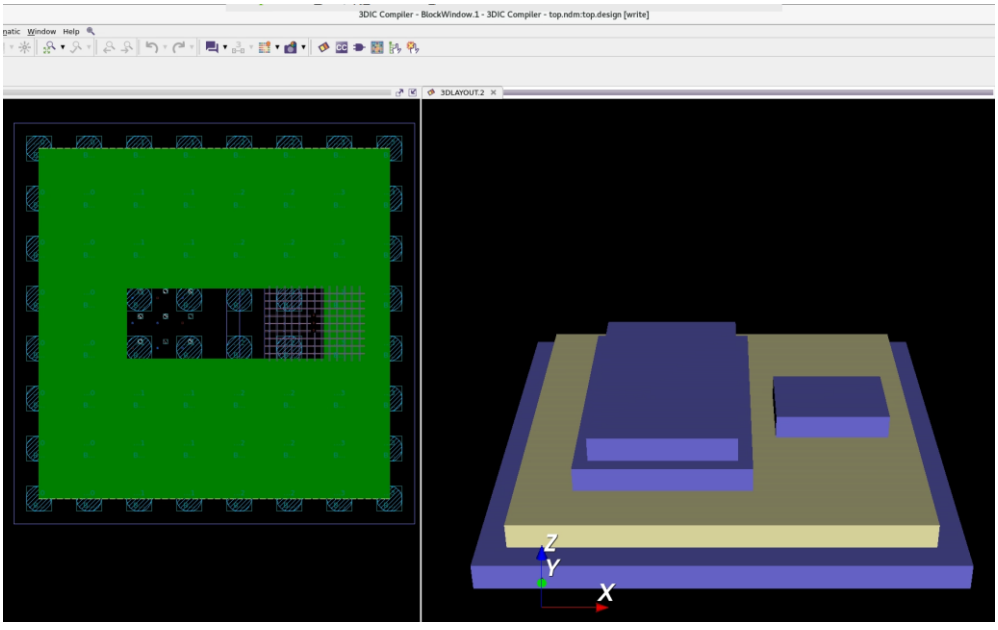

图 13：2D 和 3D 堆叠设计界面

9. 总结

本文介绍了使用 3DICC 工具进行芯片堆叠设计的步骤。3DICC 设计平台覆盖从架构规划、设计创建、设计实现到分析验证等全流程。能够支持多芯片系统集成，工作环境灵活高效，自动化布线支持。

芯和半导体 EDA 介绍

芯和半导体成立于 2010 年，是国内唯一提供“半导体全产业链仿真 EDA 解决方案”的供应商。芯和半导体 EDA 是新一代智能电子产品中设计高频/高速电子组件的首选工具，它包括了三大产品线：

- 芯片设计仿真产品线为晶圆厂提供了精准的 PDK 设计解决方案，为芯片设计公司提供了片上高频寄生参数提取与建模的解决方案；
- 先进封装设计仿真产品线为传统型封装和先进封装提供了高速高频电磁场仿真的解决方案；
- 高速系统设计仿真产品线为 PCB 板、组件、系统的互连结构提供了快速建模与无源参数抽取的仿真平台，解决了高速高频系统中的信号、电源完整性问题。

芯和半导体 EDA 的强大功能基于：自主知识产权的多种尖端电磁场和电路仿真求解技术、繁荣的晶圆厂和合作伙伴生态圈（芯和半导体 EDA 在所有主流晶圆厂的先进工艺节点和先进封装上得到了不断验证）、以及支持基于云平台的高性能分布式计算技术，在 5G、智能手机、物联网、汽车电子和数据中心等领域已得到广泛应用。

关于芯和半导体

芯和半导体是 EDA 软件、集成无源器件 IPD 和系统级封装领域的领先供应商。公司致力于为半导体芯片设计公司和系统厂商提供差异化的软件产品和芯片小型化解决方案，包括射频 IC 设计、模拟混合信号设计、系统级封装设计和高速数字系统设计等。这些产品和方案在 5G、智能手机、物联网、人工智能和数据中心等领域得到广泛应用。

芯和半导体凭借以客户需求驱动发展的理念，赢得了众多客户的青睐。随着公司自有知识产权的不断开发，芯和半导体已经成为中国集成电路自动化软件技术和微电子技术行业的标杆企业。

芯和半导体创建于 2010 年，企业总部位于上海浦东张江，并在美国硅谷、中国北京、深圳、苏州、成都、西安设有销售和技术支持中心。如欲了解更多详情，敬请访问 www.xpeedic.cn。