

Synthesis: by DC

EE291F

Speaker Yuxuan Li (liyx22025@shanghaitech.edu.cn)

VSPLab@SIST



上海科技大学

ShanghaiTech University

Contents

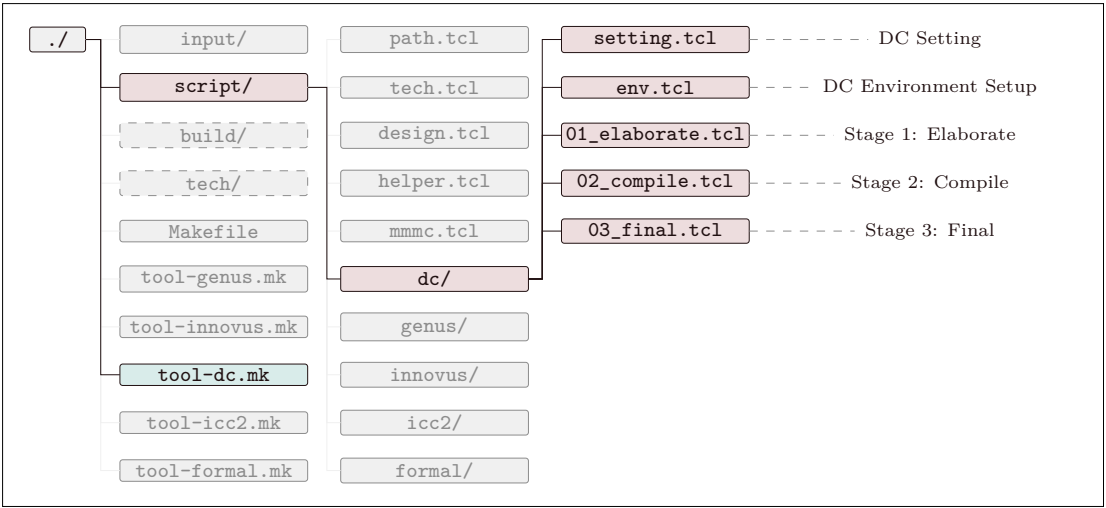
目录

1 DC：环境准备

2 DC：生成通用网表

3 DC：生成工艺网表

DC 文件组织总览


./
input/
script/
build/
tech/
Makefile
tool-genus.mk
tool-innovus.mk
tool-dc.mk
tool-icc2.mk
tool-formal.mk
path.tcl
tech.tcl
design.tcl
helper.tcl
mmmc.tcl
dc/
genus/
innovus/
icc2/
formal/
setting.tcl
env.tcl
01_elaborate.tcl
02_compile.tcl
03_final.tcl
DC Setting
DC Environment Setup
Stage 1: Elaborate
Stage 2: Compile
Stage 3: Final

DC：环境准备

DC 的启动

DC 可以通过以下方式从命令行启动，`dc_shell` 是 DC 的交互式 Shell

```
1 dc_shell
```

tool-dc.mk

创建一个独立的 tool-dc.mk，并在主 Makefile 中包含它

```
include tool-dc.mk
```

定义 DC 的命令名称

```
DC:=dc_shell
```

定义 DC 的构建目录和日志文件，**所有日志文件都用时间戳命名**

```
DC_BUILD:=${BUILD_DIR}/dc
DC_CMD:=${DC_BUILD}/${RUN_STAMP}-dc.cmd
DC_LOG:=${DC_BUILD}/${RUN_STAMP}-dc.log
DC_COMMIT:=${DC_BUILD}/${RUN_STAMP}-git-commit.txt
```



tool-dc.mk

实现构建目录的创建和删除

```
1 ${DC_BUILD}: | ${BUILD_DIR}
2     mkdir -p $@
```

```
1 dc-clean:
2     rm -r -v -I ${DC_BUILD}
```

定义 DC 的交互启动目标 dc-shell

```
dc-shell: | ${DC_BUILD}
    echo "${GIT_COMMIT}" >> $(abspath ${DC_COMMIT})
    echo "${RUN_COMMAND}" >> $(abspath ${DC_COMMIT})
    cd ${DC_BUILD} && ${DC} -x "\
            set_app_var sh_command_log_file {$(abspath ${DC_CMD})}; \
            set_app_var sh_output_log_file  {$(abspath ${DC_LOG})}"
```

- -x 选项用于从命令行向 DC 传递一条 Tcl 指令，多条指令用 ; 分隔。
- DC 没有 -log 选项，它的日志文件路径是通过 set_app_var 的 Tcl 指令指定的。

定义 DC 的脚本运行目标 dc-run

```
dc-run: | ${DC_BUILD}
    echo "${GIT_COMMIT}" >> $(abspath ${DC_COMMIT})
    echo "${RUN_COMMAND}" >> $(abspath ${DC_COMMIT})
    cd ${DC_BUILD} && ${DC} -x "\
            set_app_var sh_command_log_file {$(abspath ${DC_CMD})}; \
            set_app_var sh_output_log_file  {$(abspath ${DC_LOG})}" \
        -f $(abspath ${DC_MAIN}) < /dev/null
```

- -f 用于指定要运行的 Tcl 脚本。
- DC 没有 -batch 选项，故用 < /dev/null 的方式在进入交互时立即退出。

每个阶段都是一个独立可运行的脚本！它们的执行都可以由 `make dc-run` 派生实现

```
dc-elaborate:
    $(MAKE) dc-run DC_MAIN=${DC_SCRIPT}/01_elaborate.tcl

dc-compile:
    $(MAKE) dc-run DC_MAIN=${DC_SCRIPT}/02_compile.tcl

dc-final:
    $(MAKE) dc-run DC_MAIN=${DC_SCRIPT}/03_final.tcl
```

tool-dc.mk

完整流程就是顺次执行每个阶段

```
dc:
    $(MAKE) dc-elaborate
    $(MAKE) dc-compile
    $(MAKE) dc-final
```

env.tcl

DC 所有可以独立运行的脚本都需要统一加载一些配置，这可以写在 env.tcl 内

```
set TOOL dc
source [file join [file dirname [info script]] .. path.tcl]
source [file join ${SCRIPT_DIR} tech.tcl]
source [file join ${SCRIPT_DIR} design.tcl]
source [file join ${SCRIPT_DIR} helper.tcl]
source [file join ${DC_SCRIPT} setting.tcl]
set_host_options -max_cores 8
```

DC 的数据库不保存工具配置和工艺库导入！故 DC 的这部分内容也需要放在 env.tcl 中。

env.tcl

DC 的库查找方式和大部分工具不同：**需要先设置搜索路径，随后直接用文件名导入！**[1]

设置搜索路径

```
set_app_var search_path [concat ${search_path} ${DC_LIB_DIRS}]
```

${DC_LIB_DIRS} 应当在 setting.tcl 中定义

```
set DC_LIB_DIRS [list \
    [file join ${LIB_DIR} std] \
    [file join ${LIB_DIR} mem] \
    [file join ${DB_DIR} std] \
    [file join ${DB_DIR} mem] \
]
```

[1]这是非常糟糕的，这会在很大程度上破坏我们从 tech.tcl 开始建立的秩序。

立志成才 报国裕民

env.tcl

DC 仅支持简单的 BC-WC（Best Case / Worst Case）时序分析，没有完整的 MMMC 支持。

${DC_LIB_MAX} 和 ${DC_LIB_MIN} 应当在 setting.tcl 中定义

```
set DC_LIB_MAX [list \
    [file tail ${DB_STD_RVT_SS}] \
    [file tail ${DB_STD_LVT_SS}] \
    [file tail ${DB_MEM_M1024_SS}] \
    [file tail ${DB_MEM_M0256_SS}] \
]
set DC_LIB_MIN [list \
    [file tail ${DB_STD_RVT_FF}] \
    [file tail ${DB_STD_LVT_FF}] \
    [file tail ${DB_MEM_M1024_FF}] \
    [file tail ${DB_MEM_M0256_FF}] \
]
```

env.tcl

设置工艺映射的单元库（哪些单元可以用于工艺映射?）

```
1 set_app_var target_library ${DC_LIB_MAX}
```

设置设计链接的单元库（哪些单元可以直接出现在 RTL 代码中?）[2]

```
1 set_app_var link_library [concat * ${DC_LIB_MAX}]
```

工具规定 `target_library` 和 `link_library` 都应该设置为 SS 的 `.db` 库。

[2]这里 * 代表在解析引用时，除了查库文件，也查内存中已有的设计。

设置用于时序分析的单元库

```
foreach DC_MAX_LIB ${DC_LIB_MAX} DC_MIN_LIB ${DC_LIB_MIN} {
    set_min_library ${DC_MAX_LIB} -min_version ${DC_MIN_LIB}
}
```

先给 SS 的库，在选项 `-min_version` 后给 FF 的库。

请注意这里的循环结构依赖于 `${DC_LIB_MAX}` 和 `${DC_LIB_MIN}` 中库的排列顺序是对应的。

env.tcl

禁用一批不适合用于综合的单元

```
set_dont_use [s_get_base_cell_names ${DONT_USE_CELL_PATTERNS}]
```

s_get_base_cell_names 在 helper.tcl 中定义，输入模式列表，输出名称列表

```
proc s_get_base_cell_names {patterns} {
    set names [list]
    foreach pattern ${patterns} {
        foreach_in_collection cell [get_lib_cells */${pattern}] {
            lappend names [get_object_name ${cell}]
        }
    }
    return [lsort -unique ${names}]
}
```

env.tcl

${DONT_USE_CELL_PATTERNS} 在 setting.tcl 中定义

```
set DONT_USE_CELL_PATTERNS [list \
    {*DEL*} \
    {*ANT*} \
    {*FILL*} \
    {*TAP*} \
    {*CLK*} \
    {*GCK*} \
    {*TIE*} \
    {*X20A?9} \
    {*X05A?9} \
]
```

DC：生成通用网表

01_elaborate.tcl

第一阶段的文件头，DC 的数据库文件使用 .ddc 作为后缀

```
source [file join [file dirname [info script]] env.tcl]
set STAGE_NAME  01_elaborate
set STAGE_DIR   [stage_make_dir ${DC_BUILD} ${STAGE_NAME}]
set STAGE_DDC   [file join ${STAGE_DIR} ${DESIGN}.ddc]
```

01_elaborate.tcl

SVF 文件是综合中的变换记录，用于形式化验证，在 design.tcl 中定义

```
set DC_SVF_01                [file join ${DC_BUILD} 01_elaborate ${DESIGN}.svf]
set DC_SVF_02                [file join ${DC_BUILD} 02_compile ${DESIGN}.svf]
```

SVF 文件按阶段分开，01_elaborate 和 02_compile 各写一份！

立志成才 报国裕民

01_elaborate.tcl

开始记录变换，并写入第一阶段的 SVF 文件

```
set_svf ${DC_SVF_01}
```

定义设计库

```
define_design_lib main -path [file join ${STAGE_DIR} main]
```

解析设计

```
analyze -format verilog ${RTL_LIST} -library main
```

01_elaborate.tcl

elaborate 用于产出通用网表，**这是第一阶段的核心命令**

```
elaborate ${DESIGN} -library main
```

保护时钟网络

```
set_dont_touch_network [get_ports ${PORT_CLK}]
```

保护复位网络

```
set_dont_touch_network [get_ports ${PORT_RST}]
```

综合阶段不应触碰时钟网络和复位网络，它们需要在 CTS 阶段专门处理。

立志成才 报国裕民

01_elaborate.tcl

检查设计

```
check_design > [file join ${STAGE_DIR} check_design.rpt]
```

导出网表

```
write -hierarchy -format verilog -output [file join ${STAGE_DIR} netlist.v]
```

保存数据库

```
write -hierarchy -format ddc -output ${STAGE_DDC}
```

结束 SVF 记录

```
set_svf -off
```

DC：生成工艺网表

02_compile.tcl

第二阶段的文件头，请注意 SVF 记录需要在读数据库前启用

```
source [file join [file dirname [info script]] env.tcl]
set STAGE_NAME  02_compile
set STAGE_DIR   [stage_make_dir ${DC_BUILD} ${STAGE_NAME}]
set STAGE_DDC   [file join ${STAGE_DIR} ${DESIGN}.ddc]
set_svf ${DC_SVF_02}
```

读取上一阶段的数据库

```
read_file -format ddc [file join ${DC_BUILD} 01_elaborate ${DESIGN}.ddc]
```

02_compile.tcl

读取设计约束（SDC 文件实际上也是 Tcl 脚本）

```
source -echo -verbose ${SDC_FILE}
```

compile_ultra 用于产出工艺网表，**这是第二阶段的核心命令**

```
compile_ultra
```

DC 没有 Genus 那样的 syn_generic、syn_map、syn_opt 分解，映射与优化在一条命令中完成。

02_compile.tcl

导出网表

```
1 write -hierarchy -format verilog -output [file join ${STAGE_DIR} netlist.v]
```

保存数据库

```
1 write -hierarchy -format ddc -output ${STAGE_DDC}
```

结束 SVF 记录

```
1 set_svf -off
```

03_final.tcl

第三阶段的文件头，该阶段用于正式报告和交付，不需要 SVF 记录

```
source [file join [file dirname [info script]] env.tcl]
set STAGE_NAME  03_final
set STAGE_DIR   [stage_make_dir ${DC_BUILD} ${STAGE_NAME}]
set STAGE_DDC   [file join ${STAGE_DIR} ${DESIGN}.ddc]
```

读取上一阶段的数据库

```
read_file -format ddc [file join ${DC_BUILD} 02_compile ${DESIGN}.ddc]
```

03_final.tcl

生成一份总结报告

```
report_qor  > [file join ${STAGE_DIR} report_qor.rpt]
```

生成面积报告

```
report_area > [file join ${STAGE_DIR} report_area.rpt]
```

生成时序报告

```
report_timing -transition_time -nets -attributes -nosplit \
    > [file join ${STAGE_DIR} report_timing.rpt]
```


立志成才 报国裕民

03_final.tcl

导出网表

```
write -hierarchy -format verilog -output ${DC_NETLIST}
```

导出时序约束

```
write_sdc ${DC_SDC}
```

导出延时信息

```
write_sdf ${DC_SDF}
```

03_final.tcl

交付产物的路径在 design.tcl 中定义

```
set DC_FINAL_DIR          [file join ${DC_BUILD} 03_final]
set DC_NETLIST            [file join ${DC_FINAL_DIR} ${DESIGN}_syn.v]
set DC_SDC                [file join ${DC_FINAL_DIR} ${DESIGN}_syn.sdc]
set DC_SDF                [file join ${DC_FINAL_DIR} ${DESIGN}_syn.sdf]
```

保存数据库

```
write -hierarchy -format ddc -output ${STAGE_DDC}
```

Thank You

谢谢观看