IFP 用户手册

Product Name : IFP (ic flow platform)

Product Version : V1.2

Release Date : 2023.12.31

Contact : @李艳青 (liyanqing.1987@bytedance.com)

@景富义 (jingfuyi@bytedance.com)

@节喜 (jiexi@bytedance.com)

@马琨（makun.226@bytedance.com）

@张静文（zhangjingwen.silvia@bytedance.com）

目录

前言 1
1. 简介 1
1.1 IC 流程演化 1
1.2 IC 流程平台的主要作用 2
1.3 IFP 开发思路 2
1.4 IFP 基本概念和控制逻辑 4
1.4.1 任务拆解（从 Block 到 Task） 4
1.4.2 单个任务的行为（从 Task 到 Action） 5
1.4.3 单个行为的属性（从 Action 到 Attribute） 5
1.4.4 IFP 控制逻辑 6
2. 工具安装 7
2.1 环境依赖 7
2.1.1 操作系统依赖 7
2.1.2 python 版本依赖 7
2.1.3 集群管理系统 7
2.1.4 共享存储 7
2.2 工具下载 7
2.3 工具安装 9
3. 管理员配置 12
3.1 系统配置 12
3.2 任务配置 14
3.3 环境配置 18
4. 工具介绍 21
4.1 工具载入 21
4.2 帮助信息 21
4.3 图形界面介绍 22
4.3.1 菜单栏 23
4.3.1.1 File 23
4.3.1.2 View 25
4.3.1.3 Setup 28
4.3.1.4 Control 35
4.3.1.5 Tool 36
4.3.1.6 Help 38
4.3.2 功能区 38
4.3.3 工作区 39
4.3.3.1 ENV 页 39
4.3.3.2 CONFIG 页 40
4.3.3.3 MAIN 页 48
4.3.4 辅助工具 52
5. 基于 Demo Case 快速使用 IFP 54
5.1 用户配置 54

5.1.1 EDA 配置文件.....54
5.1.2 启动 IFP.....55
5.1.3 IFP 用户配置文件.....55
5.1.4 配置任务间依赖关系.....58
5.2 IFP 运行.....60
5.2.1 Build.....60
5.2.2 Run.....61
5.2.3 Kill.....62
5.2.4 Check.....62
5.2.5 Summarize.....64
5.2.6 Release.....64
5.2.7 操作视频.....65
6. 技术支持.....66
附录.....67
附一、变更历史.....67
附二、外部贡献者.....68
附三、Demo Case 案例介绍.....69
1. 前言.....69
2. Flow 介绍.....69
2.1 文件和目录.....69
2.2 工具版本.....70
2.3 技术库信息.....70
3. 任务运行.....71
3.1 目录创建.....71
3.2 任务运行.....73
3.3 结果检查.....75
附四、auto_check 检查机制及配置方法.....79
1. 检查机制介绍.....79
2. checklist 配置方法.....81
3. Checklist Excel -> Checklist scripts 转换方式.....81
4. 执行 checklist script.....82
5. 汇总并展示 checklist 报告.....84

前言

IFP 是 ByteDance 开源的芯片设计流程平台，全称为 ic flow platform，主要用于超大规模数字集成电路设计的流程规范管理和数据流转控制。

1. 简介

1.1 IC 流程演化

数字集成电路设计流程演进可以笼统地分为三个阶段。

手工阶段:

集成电路设计流程以手工为主，包括手工创建运行目录并解决环境依赖，手工运行 EDA 工具并检查运行结果，手工收集报告并 release 数据。

这阶段的主要瓶颈是，在集成电路设计规模较大，并且人力资源不够充足的情况下，手工操作效率较低，工作量巨大。

脚本阶段:

通过脚本化，包括但不限于 shell/Makefile/perl/python 等，将设计流程步骤中的手工操作转化为脚本执行，这样不但可以极大地提升运行效率，而且可以将流程步骤固化。

这阶段的主要瓶颈是，在超大规模集成电路设计中，每个流程往往需要多人协作，多个流程之间数据流转也更加紧密，因此对流程运行规范和行为一致性提出了更高的要求。

平台阶段:

通过一个统一的流程平台，在一个公司（或团队）范围内统一流程运行规范，数据集约化管理和展示，可以大幅降低流程运行成本（包括时间成本和交互成本）。

这个阶段的主要制约因素是，大家的操作对象从分立脚本切换到统一平台，有一个一次性的学习成本；部分流程管理员失去了对操作平台功能的直接修改权限会有一

些不适应。

1.2 IC 流程平台的主要作用

- **统一流程运行规范**

超大规模集成电路设计需要多流程参与，每个流程内部也需要多人协同工作，流程平台可以将 IC 流程运行规范通过平台工具的方式固化，确保所有的人采用统一的运行模式和规则，降低交互成本。

同分立的自动化脚本相比，流程平台是多流程唯一入口且无法随意篡改的。

- **数据集约化管理和展示**

流程平台本身承载数据检查及展示（checklist/summary）和数据管理（release）的作用，同时可以嵌入更长维度的数据保存及检索能力，可以对多项目/多流程/多用户的数据进行集约化管理和展示，对多流程间的数据交互更友好。

- **降低流程运行门槛**

GUI 界面的流程平台简单易操作，可以大幅降低新人的流程运行门槛。同时“脚本自动化层”和“EDA 设置层”通过松耦合的方式嵌入，也不影响用户了解和配置单步自动化的实现方式以及具体的 EDA 设置项。

- **提升多模块运行效率**

流程平台内嵌基于 multi-thread 或者 LSF / openlava 的多模块并行运行控制，能够在 block / version / flow / vendor / branch / task 不同的层面实现串并行精准管理。

1.3 IFP 开发思路

不同于很多公司将 EDA 配置和流程平台打包的方式（比如 AMD 的 TileBuilder），为了更好地适配各家不同的 EDA 环境，IFP 采用层次解耦的设计思路，将整个 IC 流程组织架构分为三个逻辑层次。

底层是 **EDA 设置层**，也是绝大多数 project engineer 日常接触到的部分。用户可根据自己所参加的项（如 cpu/gpu/ai/...），项目所使用的工艺节点（16/7/5/...），自己所选用的 EDA 工具（dc_shell/pt_shell/innovus/...），自己所负责的模块（PCIE/DDR/...），来决定如何设置 EDA 工具的配置文件，以保证 EDA 工具能够根据自己的合理配置运行得到期望的结果。

中间是**脚本自动化层**，也是每个 flow administrator 所维护的部分。针对单个的 block 而言，实际的 IC 流程运行并非单纯运行 EDA 工具，还需要预先准备运行环境（目录结构，input 文件，环境配置等），按照 checklist 检查 EDA 运行结果，从运行目录收集和整理重要报告及数据，结果符合预期的情况下进行数据 release，有时候还有一些后处理的工作要做，而通过脚本，无论是 shell/Makefile 还是 perl/python，都可以大幅降低这部分工作的操作难度。除此之外，多模块/多流程同时运行也可以借助脚本实现并行控制和顺序管控。

顶层是**流程平台层**，是 IC 用户 EDA 流程的入口。比较知名的 IC 流程平台有 AMD 的 TileBuilder/达索的 Altair FlowTracer/synopsys 的 Lynx 等，一般采用 GUI 界面的方式，支持多 block/version/flow/vendor/branch/task 的并行运行，并集成自动化脚本实现各个流程的自动化运行。

总地来说：

- IFP 主要涵盖流程平台层，用来调度和管理用户任务。
- 自动化脚本仍然由每个 flow 的管理员自行维护和管理，并非 IFP 内嵌的一部分。
- EDA 设置由 flow 的管理员和相关用户自行维护和管理，并非 IFP 内嵌的一部分。
- 每个流程既可以采用流程平台运行，也可以采用脚本运行，只是流程规范和功能完备性上的区别。

1.4 IFP 基本概念和控制逻辑

1.4.1 任务拆解（从 Block 到 Task）

IFP 的任务拆解逻辑如下，下面用 mid-end 流程做例子。

- **Block**，即集成电路模块。之所以把 Block 放到任务划分的顶层，是因为一般自顶向下的数字集成电路设计流程，大多按照 Block 来划分任务。
- **Version**，即 Block 不同设计阶段的版本信息。比如 PN85、PN95_v1 等。
- **Flow**，即集成电路设计的流程环节。比如 regression，DFT、synthesis、fv、sta 等，不同公司的划分和命名方式会有些许不同。
- **Vendor**，即运行某个流程所用到的哪家 EDA 厂商的工具。比如针对 synthesis 流程，常用的工具有 cadence 的 genus 和 synopsys 的 dc_shell，所以存在 cadence 和 synopsys 两个 Vendor 可选。(不同 Vendor 会影响 checklist/summary 等脚本的选择)。
- **Branch**，用于同一 Vendor 工具的不同模式区分。比如用 synopsys 的 dc_shell 来跑 synthsis 流程，也可能存在 dc 和 dcg 两种不同的运行模式。
- **Task**，每一个运行环节的最小的运行逻辑，即一个任务。比如 dc_shell 跑 synthesis，可以细分为 analyze/elaborate/initopt/advopt/finalopt 等不同的步骤，每个步骤都是一个 Task。

1.4.2 单个任务的行为（从 Task 到 Action）

每个 IC 设计流程的最小任务（task），都可以通过几个逻辑行为来实现，把每个逻辑行为称为一个 Action。

每一个 task 的实现步骤，都可以抽象为如下 5 种 Action。

每一个 Task 可能只用到其中的一种或者几种 Action，也可能用到全部 Action。

1.4.3 单个行为的属性（从 Action 到 Attribute）

对于每个 Action 而言，需要设定一些基本属性（Attribute）以帮助 IFP 确定如何实现这一 Action，其基本的属性如下：

PATH：在哪个路径下运行这个行为。

COMMAND：如何运行这个行为（执行的命令）。

RUN_METHOD：针对任务运行前的一些设置。

也就是说，要完成一个任务（比如 synthesis 的 initopt），都可以通过指定的几个步骤完成（创建环境、运行工具、结果检查...），每个步骤的行为都是“**到指定 PATH 下面，执行 COMMAND，COMMAND 的运行方式为 RUN_METHOD**”。

针对 CHECK 和 SUMMARIZE 这两个 Action，因为会生成报告，还需要查看报告，所以还有如下两个额外项目需要配置。

VIEWER：是用什么工具或者命令来查看报告，可以带参数。

REPORT_FILE：报告文件的具体位置。

这样就可以把最小任务（task）的完成，拆解为几个行为（Action），又通过定义每个行为的属性来最终实现任务。

1.4.4 IFP 控制逻辑

IFP 本质上是一个带图形界面的逻辑控制机，输入复杂的用户需求，吐出统一的流程数据。

用户输入任务需求，任务需求包括三方面：

- Block/Version/Flow/Vendor/Branch/Task 信息，在用户配置文件（user config file）中定义。
- 每个任务（Task）的行为（Action）信息，可以在用户配置文件（user config file）中实现，也可以让流程管理员在默认配置文件（default config file）中预先定义。
- 每个行为（Action）的属性（Attribute）信息，可以在用户配置文件（user config file）中实现，也可以让流程管理员在默认配置文件（default config file）中预先定义。

IFP 主要负责按照用户的初始信息（用户配置文件）和操作指令（GUI 界面的点击操作），并行或者串行运行用户指定的任务，并最终汇总输出结果。

2. 工具安装

2.1 环境依赖

2.1.1 操作系统依赖

IFP 的开发和测试 OS 为 **CentOS Linux release 7.9.2009 (Core)**。

centos6/centos7/centos8，及对应的 redhat 版本应该都可以运行，主要的潜在风险在于系统库版本差异可能会影响部分组件的运行。

建议使用 centos7.9 操作系统。

2.1.2 python 版本依赖

IFP 基于 python 开发，其开发和测试的 python 版本为 **python3.8.8**，推荐使用 **Anaconda3-2021.05** 以解决库依赖问题。

不同版本的 python 会有 python 库版本问题。

2.1.3 集群管理系统

IFP 的多任务并行控制主要通过集群管理系统来实现，所以需要安装 LSF 或者 openlava 集群管理系统。

2.1.4 共享存储

由于 IFP 可以借助集群管理系统实现多服务器的资源使用，因此依赖共享存储以保证多服务器上所访问到的数据一致性。

2.2 工具下载

IFP 的 github 路径位于 https://github.com/bytedance/ic_flow_platform。

可以采用“git clone https://github.com/bytedance/ic_flow_platform.git”的方式拉取源代码。

```Bash
bytedance@C02FT5LHMD6R Downloads % git clone
https://github.com/bytedance/ic_flow_platform.git
Cloning into 'ic_flow_platform'...
remote: Enumerating objects: 111, done.
remote: Counting objects: 100% (111/111), done.
remote: Compressing objects: 100% (101/101), done.
remote: Total 111 (delta 1), reused 108 (delta 1), pack-reused 0
Receiving objects: 100% (111/111), 3.80 MiB | 1.17 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

也可以在 IFP 的 github 页面上，Code -> Download ZIP 的方式拉取代码包。

2.3 工具安装

工具安装之前，首先参照 2.1 节“环境依赖”满足 IFP 的环境依赖关系。

将安装包拷贝到安装目录，并给与合适的目录名。

安装包下的文件和目录如下。

```
Bash
[liyanqing.1987@n212-206-194 tools]$ cd ic_flow_platform/
[liyanqing.1987@n212-206-194 ic_flow_platform]$ ls
action  bin  common  config  data  docs  install.py  LICENSE.txt
Notice.txt  patch  README  requirements.txt  third_part  tools
```

首先确认当前 python 版本正确。

```
Bash
[liyanqing.1987@n212-206-194 ic_flow_platform]$ which python3
/ic/software/tools/python3/3.8.8/bin/python3
```

基于安装包中的 requirements.txt 安装 python 依赖库。（可能需要 root 权限）

```
Bash
[root@ic-admin1 ic_flow_platform]# pip3 install -r
```

```
requirements.txt
Looking in indexes: https://bytedpypi.byted.org/simple/
Collecting matplotlib==3.6.3
  Downloading
https://bytedpypi.byted.org/packages/pypi/matplotlib/matplotlib-
3.6.3-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl
(9.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.4/9.4 MB 174.4
MB/s eta 0:00:00
...
Installing collected packages: matplotlib
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.3.4
    Uninstalling matplotlib-3.3.4:
      Successfully uninstalled matplotlib-3.3.4
Successfully installed matplotlib-3.6.3
```

在安装目录下，使用命令“python3 install.py”安装 IFP。

```Bash
[liyanqing.1987@n212-206-194 ic_flow_platform]$ python3 install.py
>>> Generate wrapper script "bin/ifp" ...
>>> Generate wrapper script
"action/check/scripts/gen_checklist_scripts" ...
>>> Generate wrapper script
"action/check/scripts/gen_checklist_summary" ...
>>> Generate wrapper script "action/check/scripts/ic_check" ...
>>> Generate wrapper script
"action/check/scripts/view_checklist_report" ...

>>> Generate config file
"/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/config/config.
py".
>>> Generate top sh environment file
"/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/config/env.sh"
...
```

```
>>> Generate top csh environment file
"/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/config/env.csh
" ...
>>> Update EXPECTED_PYTHON/IFP_INSTALL_PATH settings for specified
tools ...
>>> Install tool "lsfMonitor" ...

Install successfully!
```

3. 管理员配置

3.1 系统配置

config/config.py：ifp 的主配置文件。如下为系统的默认值:

```
Bash
## admin settings: Only administrator can modify below settings
# Only default_yaml_administrators can edit default.yaml on ifp
GUI directory.
default_yaml_administrators = ""

# system log
system_log_path = ""

# Specify lmstat path, example
"/eda/synopsys/scl/2021.03/linux64/bin/lmstat".
lmstat_path = ""

## IFP Config settings: Users can define personal settings in
~/.ifp/config/config.py, Below are default value.
# send result command
send_result_command = ""

# xterm command.
xterm_command = "xterm -e"

# launch ifp in fullscreen
fullscreen_flag = True

# remind user to confirm if rerun passed tasks
rerun_flag = True

# task will run even if dependent tasks failed
ignore_fail = False
```

```
# send result to users after action done
send_result = False

# import all tasks when add new block/version
auto_import_tasks = True

# Auto rerun CHECK(SUMMARIZE) command before view check(summarize)
report
rerun_check_or_summarize_before_view = True
```

示例设置为:

```Bash
# Only default_yaml_administrators can edit default.yaml on ifp
GUI directory.
default_yaml_administrators = "liyanqing.1987 jiexi jingfuyi"

# system log
system_log_path = "/ic/data/CAD/flows/ic_flow_platform/system.log"

# Specify lmstat path, example
"/eda/synopsys/scl/2021.03/linux64/bin/lmstat".
lmstat_path =
"/ic/software/cad_tools/it/licenseMonitor/tools/lmstat"

# send result command
send_result_command = "/ic/software/cad_tools/bin/send_lark -c
RESULT -r USER"

# xterm command.
xterm_command = "/bin/xterm -e"
```

- 管理员需要配置的系统命令:

default_yaml_administrators，用于指定谁可以在 IFP 的 GUI 界面中修改

default.yaml，多用户之间用空格隔开。

system_log_path，用于指定系统日志路径，存放用户启动记录，若不配置，不记录相关信息。

lmstat_path，用于指定 lmstat path，获取用户 license 信息，若不配置，则无法执行 action 前的 license 检查功能。

xterm_command，用于指定 MAIN 界面最后一列交互终端模式，可自行配置 xterm/gnome-terminal/konsole 等。

send_result_command，在 RUN 这个行为后执行什么命令，一般用于给 IFP 用户发送汇总结果，也可以不设。如果设置的话，支持 RESULT 和 USER 两个变量，其中 RESULT 会被 Task 当前状态的文本取代，USER 会被当前用户名取代。

- IFP 的行为开关，管理员配置默认值，用户可在 IFP 界面进行个性化调整，并且会将用户自己的配置记录在家目录下：

fullscreen_flag，用于指定启动 IFP GUI 界面是否全屏模式，默认是 True。
rerun_flag，提醒用户已经 run_pass 的任务是否需要 rerun，默认是 True。
ignore_fail，用户是否忽略前置任务 fail，继续后置任务的运行，默认是 False。
send_result，用于 action 完成后是否开启结果通知，默认是 False。
auto_import_tasks ，新增 block/version 信息时是否自动 import default.yaml 中所有的 tasks，默认是 True。
rerun_check_or_summarize_before_view，查看报告之前，是否重新执行 CHECK/SUMMARIZE 操作，默认是 True。

3.2 任务配置

config/default.yaml：用于定义常用 Task 的默认 Action/Attribute 设置，从而简化用户的配置流程。如下为原始设置示例：

```Bash
## Format ##
# VAR:
#     key: value
#
# TASK:
```

```
#     flow:vendor:task:
#      action:
#          attribute: value
#
#
## Supported Variables (default) ##
# CWD: <ifp start directory>
# IFP_INSTALL_PATH: <ifp install path>
# BLOCK: <block name>
# VERSION: <version name>
# FLOW: <flow name>
# VENDOR: <vendor name>
# BRANCH: <branch name>
# TASK: <task name>
#
#
## Supported action ##
# BUILD/RUN/CHECK/SUMMARIZE/POST_RUN/RELEASE
#
#
## Supportedd action attribute ##
# PATH/COMMAND/RUN_METHOD/VIEWER/REPORT_FILE
#
#
## Example ##
# VAR:
#     BSUB_QUEUE: ai_syn
#     DEFAULT_PATH:
${CWD}/${BLOCK}/<VERSION>/<FLOW>/<VENDOR>/<BRANCH>
#
# TASK:
#     synthesis:synopsys:intopt:
#      BUILD:
#          PATH: $DEFAULT_PATH
#          COMMAND: make build
#      RUN:
#          PATH: ${DEFAULT_PATH}/dc
```

```
#          COMMAND: make run_initopt
#          RUN_METHOD: bsub -q $BSUB_QUEUE -n 8 -R
"rusage[mem=50000]"
#      CHECK:
#          PATH: ${DEFAULT_PATH}/dc
#          COMMAND:
${IFP_INSTALL_PATH}/action/check/syn/synopsys/syn_synopsys.syn_dc.
py -b ${BLOCK}
#          VIEWER:
${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report.py
-i
#          REPORT_FILE: file_check/file_check.rpt
#      SUMMARIZE:
#          PATH: ${DEFAULT_PATH}/dc
#          COMMAND:
${IFP_INSTALL_PATH}/action/summarize/collect_syn_qor.py
#          VIEWER: /bin/soffice
#          REPORT_FILE: syn_qor.xlsx
#      POST_RUN:
#          PATH: ${DEFAULT_PATH}/dc
#          COMMAND: make post_run
#          RUN_METHOD: bsub -q $BSUB_QUEUE -n 8 -R
"rusage[mem=50000]"
#      RELEASE:
#          PATH: ${DEFAULT_PATH}/dc
#          COMMAND: make release
```

每个 Task 行为及属性信息的编辑，会带来较大的工作量。因为同一类型 Task 的行为及行为属性配置都是固定的，所以管理员可以提前将这些设置放置到一个默认的配置文件中（IFP 安装目录下的 config/default.yaml），让 IFP 在读取用户配置的时候直接引用这些默认的 Task 属性，这样可以极大减轻配置工作量。

管理员可通过 IFP 自带的 config default yaml 工具（Tool -> Config Default Yaml）进行编辑配置，工具样式如下。

编辑完的示例如下:

```Bash
...
VAR:
    DEFAULT_PATH: ${CWD}/${BLOCK}/${VERSION}_${BRANCH}
TASK:
    syn:synopsys:syn_dc:
     BUILD:
         PATH: ${DEFAULT_PATH}
         COMMAND: sleep 3
     RUN:
         PATH: ${DEFAULT_PATH}
         COMMAND: make syn_dc run_local=1
         RUN_METHOD: bsub -q ${BSUB_QUEUE} -Is
     CHECK:
         PATH: ${DEFAULT_PATH}/syn_dc
         COMMAND:
${IFP_INSTALL_PATH}/action/check/syn/synopsys/syn_synopsys.syn_dc.
```

```
py
          -b ${BLOCK}
         VIEWER:
${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report
          -i
         REPORT_FILE:
${DEFAULT_PATH}/syn_dc/file_check/file_check.rpt
     SUMMARIZE:
         PATH: ${DEFAULT_PATH}/syn_dc
         COMMAND:
${IFP_INSTALL_PATH}/action/summary/collect_syn_qor.py
         VIEWER: /bin/soffice
         REPORT_FILE: ${DEFAULT_PATH}/syn_dc/syn_qor.xlsx
```

只有指定的流程管理员可以直接编辑 default.yaml，以防止数据错乱。另外 IFP 支持以变量的形式定义参数，包括：

1、IFP 内置变量：${IFP_INSTALL_PATH} / ${CWD} / ${BLOCK} / ${VERSION} / ${FLOW} / ${VENDOR} / ${BRANCH} / ${TASK}

2、系统环境变量

3、管理员定义在 default yaml 中的变量

4、用户定义在 ifp.cfg.yaml 中的变量

注：如果 IFP 中有未被识别的变量（如${DEMO_PATH}），则用户 terminal 界面会有如下提示信息。

```
*Warning*: Failed on expanding variable for "python3 ${DEMO_PATH}/setup/build.py -s ${DEMO_PATH} -f ${FLOW} -d ${BLOCK} -v ${VERSION} -b ${BRANCH}" : 'DEMO_PATH'
*Warning*: Failed on expanding variable for "python3 ${DEMO_PATH}/project_setting/check/syn/synopsys/syn_synopsys.syn_dc.py -b ${BLOCK}" : 'DEMO_PATH'
*Warning*: Failed on expanding variable for "python3 ${DEMO_PATH}/setup/build.py -s ${DEMO_PATH} -f ${FLOW} -d ${BLOCK} -v ${VERSION} -b ${BRANCH}" : 'DEMO_PATH'
*Warning*: Failed on expanding variable for "python3 ${DEMO_PATH}/project_setting/fv/synopsys/fv_synopsys.rtl2gate.py -t rtl2gate -b ${BLOCK}" : 'DEMO_PATH'
```

3.3 环境配置

config/env.*：env.csh 和 env.sh 分别用于定义 c/tcsh 和 sh/bash 中的用户默认环境变量设置，一般用于指定 EDA 工具版本，下面以 env.sh 为例。

原始设置为：

```
Bash

#### Default EDA tool settings ####
# Set default TESSENT setting.

# Set default DC setting.

# Set default GENUS setting.

# Set default FORMALITY setting.

# Set default LEC setting.

# Set default PT setting.

# Set default TEMPUS setting.

# Set default ICC2 setting.

# Set default INNOVUS setting.

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

# Set lsfMonitor path.
export
PATH=/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/tools/lsfM
onitor/monitor/bin:$PATH

# Set default soffice path.
```

管理员根据项目需求可进行 EDA 工具的配置，示例设置为：

```
Bash
```

```
#### Default EDA tool settings ####
# Set default TESSENT setting.

# Set default DC setting.
module load dc_shell/T-2022.03-SP3

# Set default GENUS setting.
module load genus/21.14-s082_1

# Set default FORMALITY setting.
module load formal/T-2022.03-SP5

# Set default LEC setting.

# Set default PT setting.
module load lc_shell/S-2021.06-SP4
module load pt_shell/R-2020.09-SP5-4

# Set default TEMPUS setting.

# Set default ICC2 setting.

# Set default INNOVUS setting.

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

# Set lsfMonitor path.
export
PATH=/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/tools/lsfM
onitor/monitor/bin:$PATH

# Set default soffice path.
```

当前针对 demo_case 是增加了 dc、genus、formality 和 pt 的默认设置，即可在 IFP 的 ENV 页 PATH 项中看默认路径信息。

4. 工具介绍

4.1 工具载入

IFP 的主程序是 ifp，位于 IFP 安装路径下的 bin/ifp，安装后可以直接引用。

如果配置了 modules，也可以通过 module load 的方式引用。

配置 ifp 的 alias 也是一种比较简便的方式。

```
Bash
[liyanqing.1987@n212-206-194 ic_flow_platform]$ alias
ifp=/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/bin/ifp
[liyanqing.1987@n212-206-194 ic_flow_platform]$ which ifp
alias
ifp='/ic/data/usr/liyanqing.1987/tools/ic_flow_platform/bin/ifp'
     /ic/data/usr/liyanqing.1987/tools/ic_flow_platform/bin/ifp
```

4.2 帮助信息

ifp 的帮助信息如下。

```
Bash
[liyanqing.1987@n212-206-194 ic_flow_platform]$ ifp -h
usage: ifp.py [-h] [-config_file CONFIG_FILE] [-build] [-run] [-
check] [-summary] [-post_run] [-release] [-d]

optional arguments:
  -h, --help         show this help message and exit
  -config_file CONFIG_FILE, --config_file CONFIG_FILE
               Specify the configure file, default is
"<CWD>/ifp.cfg.yaml".
  -build, --build    Enable build function, create run
directories/files.
  -run, --run        Enable run function, run tasks/corners.
  -check, --check    Enable check function, check results of
```

```
tasks/corners with specified checklist.
  -summarize, --summarize   Enable summarize function, get summary
report with specified information requirement.bll
  -release, --release   Enable release function, release current
result to release directory.
  -d, --debug        Enable debug mode, will print more useful
messages.
```

--help: 打印帮助信息。

--config_file: 指定用户配置文件，用来声明跑什么任务，默认为“./ifp.cfg.yaml”。

--build: 功能项，开启 build 功能。

--run: 功能项，开启 run 功能。

--check: 功能项，开启 check 功能。

--summarize: 功能项，开启 summarize 功能。

--release: 功能项，开启 release 功能。

--debug: 开启 debug 功能。打印更多的中间执行过程到桌面上。

其中所有的功能项都可以直接在 GUI 界面上操作，此处只是为了增加非交互式的 command_line 接口。

4.3 图形界面介绍

在流程运行目录下执行命令“ifp”，可以启动 ifp 的 GUI 界面。

```Bash
[liyanqing.1987@n212-206-194 ifp_test]$ ifp
>>> Generating empty configure file
"/ic/data/usr/liyanqing.1987/ifp_test/ifp.cfg.yaml" ...
```

默认的启动界面包括三个区域，分为菜单栏、功能区和工作区。

菜单栏囊括了完整的功能和配置选项。

功能区则将最常用的任务控制功能放到主界面方便调用。

工作区包括 ENV/CONFIG/MAIN 三个子页，分别用来做环境信息查看、用户任务配置和运行信息展示。

4.3.1 菜单栏

菜单栏包括 File/View/Setup/Control/Tool/Help 几部分。

4.3.1.1 File

包含“Save Status File”、“Load Status File”、“Save Config File”、“Load Config File”和“Exit”五个功能。

Save Status File：保存当前任务的 Status/Job/Runtime 等信息到指定的 yaml 文件中。

点击“File” -> “Save Status File”，然后在 File_name 处命名为“status_cfg.yaml”，点击“Save”进行保存，GUI 界面下方会显示保存信息”。

Load Status File: 载入之前保存的 status file，将 Status/Job/Runtime 等信息直接展开到当前主界面中。

点击“File” -> “Load Status File”，弹出界面框后选择“status_cfg.yaml”，然后点击“Open”，GUI 界面将状态信息直接展开到当前主界面中，GUI 界面下方会显示加载信息。

Save Config File: 保存用户配置文件。

Load Config File: 载入用户配置文件。

Exit: 退出。

4.3.1.2 View

包含“Config View”、“Main View”、“Zoom In”、“Zoom Out”和“Detail Status Info”五个功能。

Config View：显示和隐藏 Config 界面的行和列。

显示和隐藏 config 界面的行和列信息，若用户不关注 flow 信息，则选择取消勾选 flow 项，config 界面 flow 列则被隐藏，如下：

Main View：显示和隐藏 main 界面的行和列。

显示和隐藏 main 界面的行和列信息，若用户取消勾选不关注的某项信息，main 界面该项均会隐藏。

比如取消勾选 Vendor 列，则 Vendor 列隐藏，如下：

比如取消勾选"vdec_top" block 的所有 task 信息，则 main 界面只剩下 mctf_register 模块信息，如下：

Zoom In：增加 IFP 主界面的高度。

增加 IFP 主界面的高度，点击一次放大一行。

Zoom Out: 减少 IFP 主界面的高度。

减少 IFP 主界面的高度，点击一次缩少一行。

Detail Status Info：展示 Task 详细状态信息。快捷键为“Ctrl + H”。

用户需要注意的是，如果某个 Task 定义了 Check 项，则运行结果以 Check Pass / Check Fail 为准，如果此 Task 未定义 Check 项，则运行结果以 Run Pass / Run Fail 为准。

4.3.1.3 Setup

包含“Setting”和“Set Dependency”两个设置项。

Setting：包含 Project settings 和 System settings 两部分。

Project settings 主要由用户设置 project name 和 user group 信息，点击保存后，IFP 会按照如下顺序获取 default.yaml：

a. 按照路径优先级而言：首先看 <HOME>/.ifp/config 存在，若存在则用此路径下的配置；如果不存在，用<IFP_INSTALL_PATH>/config 下的配置。

b. 按照文件优先级而言：如果指定了 ifp --default_yaml，用指定的文件，除此之外，按照 default.project.group.yaml --> default.group.yaml --> default.project.yaml --> default.yaml 的顺序。

System settings 包含 Appearance、Cluster management 和 Process management 三部分内容。

- Appearance 用于指定打开 IFP 时是否开启全屏模式；
- 集群管理用于设置 job 的运行队列；

- 流程管理用于任务运行流程控制，包含 Ignore fail tasks、Rerun before view、Send results 以及 Confirm reun tasks 四部分。

a. Ignore fail tasks：前置 job run fail 后，后续任务是否继续提交。

若未勾选 Ignore fail tasks，串行 Task 中，前者 task 失败后者 task 会自动取消，如下：

如果勾选了 Ignore fail tasks，则前者失败后，后者 task 仍然运行。

b. Rerun before view：查看 checklist/summary 报告前是否需要再次执行 CHECK/SUMMARIZE 操作。

c. Send results：IFP 的结果默认展示在 GUI 界面左下角的 Task Run Status 上，如果选中了 Send results，会把汇总结果信息通过管理员指定的方式发送给用户。（当前是通过 send_lark 发送到用户的飞书上）

如下选中 Send Result，然后点击 All Steps 运行 job，会将 IFP 的汇总结果信息发送给用户。

发送信息及结果如下：

Task Run Status	
Total	10
Run	0
Passed	6
Failed	4
Others	0

d. Confirm rerun tasks：提醒用户针对 Pass 的任务，再次选中后点击 run 按钮是否 rerun。

若勾选上 Confirm rerun tasks，则针对 Pass 的 job 再次点击 run 后，会弹窗列举出已经 Pass 的任务，并由用户确认是否 rerun。

点击 Yes 则任务会再次提交，点击 No 则不会再次提交当前已经 Pass 的任务，点击 Cancel 不进行任何操作；若不勾选 Confirm rerun tasks，则不会提醒用户，且默认都会 rerun。

Set Dependency：设置 task 间的 dependency 关系。

用户根据需求设置 Flow / Task 之间的 dependency 关系。

Dependency 关系主要分为与、或和多次触发类型。

- 与：syn 和 fv 均运行完之后才能继续进行 sta。

- 或：syn 跑完或者 fv 跑完就可以触发 sta，且 sta 只运行一次。

- 多次触发：在 task 处右键“Add Another Condition”，则 fm_rtl2gate 和 fm_rtl2pg 运行完后都会触发 fm_syn2pg，fm_syn2pg 会运行两次。

用户可在“Block / Version”下拉菜单选择不同模块和版本进行设置。

用户可鼠标右键将当前设置的 denpendency 关系复制和应用到其他模块和版本。

4.3.1.4 Control

包含“All Steps”、“Build”、“Run”、“Kill”、“Check”、“Summarize”和“Release”七个功能项。

All Steps: 执行如下所有的步骤。

Build: 执行 Build 行为。

Run: 执行 Run 行为。

Kill: 终止所选中的任务。

Check: 执行 Check 行为。

Summarize: 执行 Summary 行为。

Release: 执行 Release 行为。

4.3.1.5 Tool

包含“LSF monitor”和“Config Default Yaml”两个功能。

LSF monitor：执行开源工具 lsfMonitor，用于获取 LSF 的基本信息。

lsfMonitor 是一款用于 LSF/openlava 数据收集、分析及展示的开源工具，亦可用于 EDA license 实时信息检索，可以满足集成电路设计用户对于 LSF/license 的绝大部分信息查询和常规问题解决需求。

	Job	User	Status	Queue	Host	Started	Project	Slot	Rusage (G)	Mem (G)	Command
454	12267414		RUN	fpga	n212-204-144	2023-12-21 10:26:42	fpga	2	9.8	0.4	make -f /fpga/proj/fpga/...
455	12268354		RUN	fpga	n212-204-144	2023-12-21 10:37:05	fpga	2	5	0.1	make sim >/dev/null 2
456	9993565		RUN	orca	n232-132-209	2023-12-07 23:59:44	orca	8	78.1	2.7	gmake -f ../...
457	10636187		RUN	orca	n232-132-209	2023-12-12 10:23:35	orca	4	100	1.2	pt_shell -f run.tcl
458	11151929		RUN	orca	n232-132-209	2023-12-14 11:03:09	orca	8	50	13.8	/ic/software/synopsys/...
459	11865191		RUN	orca	n232-132-209	2023-12-18 17:12:18	orca	8	43.6	19.7	fc_shell
460	12189309		RUN	orca	n232-132-209	2023-12-20 18:25:07	orca	1		2.5	#!/bin/...
461	12189310		RUN	orca	n232-132-209	2023-12-20 18:25:07	orca	1		2.7	#!/bin/...
462	12195369		RUN	orca	n232-132-209	2023-12-20 19:33:34	orca	8	78.1	81.9	joules_studio -files ../scr...
463	12266486		RUN	orca	n232-132-209	2023-12-21 10:07:51	orca	8	50	4.9	innovus -stylus -wait 12...
464	10092228		RUN	vivian	n232-132-084	2023-12-08 10:27:40	vivian	1	2.2	1.1	./run -verdi
465	10274869		RUN	vivian	n232-132-084	2023-12-09 15:58:47	vivian	1	7.2	5	verdi -ssf sim/sim/...
466	11284317		RUN	vivian	n232-132-084	2023-12-15 10:29:24	vivian	1	4.9	3.1	verdi -ssf wave.vf
467	11313489		RUN	vivian	n232-132-084	2023-12-15 14:31:45	vivian	1	1.1	1.5	wv
468	11317944		RUN	vivian	n232-132-084	2023-12-15 15:15:00	vivian	1	0.6	1.5	verdi.run

Config Default Yaml：执行 config_default_yaml 工具，用于编辑默认 Task 属性。（只有 IFP 管理员有权限）

管理员可根据项目需求进行编辑 Default Config，用于定义 Task 的具体属性。

设定的基本属性内容如下：

	Item	Value
7	* RUN	
8	PATH	${DEFAULT_PATH}
9	COMMAND	make syn_dc
10	RUN_METHOD	bsub -q ${BSUB_QUEUE} -Is
11	REQUIRED_LICE...	

	Item	Value
12	* CHECK	
13	PATH	${DEFAULT_PATH}/dc
14	COMMAND	python3 ${DEMO_PATH}/project_setting/check/syn/synopsys/syn_synopsys.syn_dc.py -b ${BLOCK}
15	RUN_METHOD	
16	VIEWER	${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report -i
17	REPORT_FILE	${DEFAULT_PATH}/dc/file_check/file_check.rpt

PATH：定义任务执行路径。

COMMAND：定义任务执行的命令。

RUN_METHOD：定义任务运行前的一些设置，包含：执行这个行为的时候，是否需

要 bsub 出去（指定 bsub 的命令），如果没设定，默认是本地运行；是否需要开启交互界面(如“xterm -e”或者“gnome-terminal --”)，如果没设定，默认未开启。

REQUIRED_LICENSE：定义任务运行调用的 EDA License 信息，用于 job 提交前的 license 前置检查。

针对 CHECK 和 SUMMARIZE 这两个 action，因为会生成报告，还需要查看报告，所以还有如下两个额外项目需要配置。

VIEWER：定义用什么工具或者命令来查看报告，可以带参数。

REPORT_FILE：定义报告文件的具体位置。

4.3.1.6 Help

包含“About”信息项。

Help 菜单中的“About”内容如下。

4.3.2 功能区

功能区同菜单栏的 Control 菜单内容相同，将这些常用的控制功能直接挪到了功能区。

从左到右依次是:

Run All Steps：运行后面所有步骤

Build ： 创建运行环境

Run ： 运行 EDA 工具

Kill ：终止选定的任务

Check ： checklist 检查

Summarize ： summary 报告收集

Release ： 数据 release

4.3.3 工作区

工作区是用户主要的操作区域和信息获取区域，分为 ENV/CONFIG/MAIN 三个子页面。

4.3.3.1 ENV 页

查看当前的环境变量信息，可以用户环境设置确认。

4.3.3.2 CONFIG 页

配置和查看用户配置文件。

- **Task 右键编辑**

  包含“Edit task”，“Add task”，“Copy task”和“Remove task”四部分。

Edit task：用来编辑 Task 的属性，包含“Env setting”和“Flow setting”两项。

Add task：新增 task 项目，在 block/version/flow/vendor/branch 一致的情况下，快速增加 task。

- 若新增 task 项在 default yaml 预先有定义且用户不修改 task 属性时，字体显示为黑色；
- 若新增 task 项在 default yaml 预先有定义但用户修改 task 属性时，字体显示为蓝色；
- 若新增 task 项未在 default yaml 预先有定义，字体显示为 红色，且 task 对应的 flow setting 均为空，需要用户自行定义；

Copy task：复制当前 task 项目，在 block/version/flow/vendor/branch 一致的情况下，可直接修改 task 名称，copy 后此 task 属性与之前 task 属性完全一致，用户可根据需求进行调整。

Remove task：删除当前 task 项目。

- **Branch 右键编辑**

包含“Add branch”，“Copy branch”和“Remove branch”三部分，功能同 task 项。

- **Vendor 右键编辑**

包含“Add vendor”，“Copy vendor”和“Remove vendor”三部分。 如下为新增 vendor 为 cadence 的相关 task 信息：

点击“Add vendor”，弹出如下界面供用户填写 Vendor/Branch/Task 信息，完成后点击 save 保存。

- **Flow 右键编辑**

包含“Add flow”，“Copy flow”和“Remove flow”三部分。如下新增 flow 为 sta 相关的 task 信息。

点击“Add flow”，弹出如下界面供用户填写 Flow/Vendor/Branch/Task 信息，完成后点击 save 保存。

- **Version 右键编辑**

包含“Add version”，“Copy version”和“Remove version”三部分。

- **Block 右键编辑**

包含“Add block”，“Copy block”和“Remove block”三部分。如果新增 block 需要运行的任务与上个 block 一致，则直接点击“Copy block”，然后只修改 block 名称即可，如下。

可在 New Setting 处进行编辑，根据需求为新增 block 设置对应 setting，其中灰色字体处不允许修改。

- **多选功能**

用户采用“shift+鼠标选择”或者“Ctrl+鼠标选择”可以任意选择几个或者多个 block / version / flow / vendor / branch / task 进行操作。

如下多选删除 syn_dc1 和 dataout，Ctrl+鼠标选择”选中 syn_dc1 和 dataout，然后右键 Remove tasks。

如下为快速新增 branch，“shift+鼠标选择”选中所有 test branch，右键 copy branches，弹出界面供用户编辑。

用户可以直接在 New branch 处输入 test2，则 Branch 列会统一输入，保存后结果如下：

根据用户需求也可以针对不同 flow 设置不同的 branch，则需要在 Branch 黄色背景处双击输入，保存后结果如下：

4.3.3.3 MAIN 页

Task 的运行信息展示区域。

主界面上，我们可以看到 Task 的如下信息（或者访问如下功能）。

- **Task**：鼠标单击可以实现 task 全选和取消全选；鼠标右键可批量选择 flow 和 task。
- **Check** ： 如果存在文本图标，可以查看 checklist 报告。
- **Summarize** ： 如果存在文本图标，可以查看 summary 报告。
- **Job** ： 如果显示数字，可以得知采用 LSF 分布式并行运行的 Task 任务的

jobid，点击数字，可以使用自带的工具 lsfMonitor 查看任务信息。

- **Runtime**： 如果 Action 的 Run_method 指定了 LSF，则在任务开始后会显示对应 Job 的 Runtime 实时信息。
- **Xterm**：点击黑色按钮，弹出 Xterm 供用户进行操作。

Task

1. Select Flows：批量选中主页中的 Flow 项，例如选中 flow 为 gen_dir 和 syn 项。

2. Select Tasks：批量选中主页中的 Task 项，例如选中 build、syn_dc 和 presta 三个 task 项。

Check

点击 Task 行的 Check 图标，会对当前 Task 做 checklist 检查，并弹出 checklist 检查报告（如果已设置）。其中执行过 Check，Pass 图标呈现绿色，Fail 图标呈现红色；未执行过 Check，图标呈现红色。

Summarize

点击 Task 行的 Summarize 图标，会对当前 Task 做 Summary 检查，并弹出 summary 汇总报告（如果已设置）。执行过 Summarize，Pass 图标呈现绿色，Fail 图标呈现红色；未执行过 Summarize，图标呈现红色。

Job

点击 Task 行的 Job 信息，会调用 lsfMonitor 来展示 job 的基本信息。

Xterm

点击 Task 行的 Xterm 图标，会弹出一个界面，并自动进入 Task 的路径。（Run 路径或者 Build 路径，默认是当前运行路径）

4.3.4 辅助工具

菜单栏 Tool -> lsfMonitor 可载入辅助工具。

lsfMonitor 是一款用于 LSF/openlava 数据收集、分析及展示的开源工具，亦可用于 EDA license 实时信息检索，可以满足集成电路设计用户对于 LSF/license 的绝大部分信息查询和常规问题解决需求。

	Job	User	Status	Queue	Host	Started	Project	Slot	Rusage (G)	Mem (G)	Command
454	12267414		RUN	fpga	n212-204-144	2023-12-21 10:26:42	fpga	2	9.8	0.4	make -f /fpga/proj/fpga/...
455	12268354		RUN	fpga	n212-204-144	2023-12-21 10:37:05	fpga	2	5	0.1	make sim >/dev/null 2
456	9993565		RUN	orca	n232-132-209	2023-12-07 23:59:44	orca	8	78.1	2.7	gmake -f ../...
457	10636187		RUN	orca	n232-132-209	2023-12-12 10:23:35	orca	4	100	1.2	pt_shell -f run.tcl
458	11151929		RUN	orca	n232-132-209	2023-12-14 11:03:09	orca	8	50	13.8	/ic/software/synopsys/...
459	11865191		RUN	orca	n232-132-209	2023-12-18 17:12:18	orca	8	43.6	19.7	fc_shell
460	12189309		RUN	orca	n232-132-209	2023-12-20 18:25:07	orca	1		2.5	#!/bin/...
461	12189310		RUN	orca	n232-132-209	2023-12-20 18:25:07	orca	1		2.7	#!/bin/...
462	12195369		RUN	orca	n232-132-209	2023-12-20 19:33:34	orca	8	78.1	81.9	joules_studio -files ../scr...
463	12266486		RUN	orca	n232-132-209	2023-12-21 10:07:51	orca	8	50	4.9	innovus -stylus -wait 12...
464	10092228		RUN	vivian	n232-132-084	2023-12-08 10:27:40	vivian	1	2.2	1.1	./run -verdi
465	10274869		RUN	vivian	n232-132-084	2023-12-09 15:58:47	vivian	1	7.2	5	verdi -ssf sim/sim/...
466	11284317		RUN	vivian	n232-132-084	2023-12-15 10:29:24	vivian	1	4.9	3.1	verdi -ssf wave.vf
467	11313489		RUN	vivian	n232-132-084	2023-12-15 14:31:45	vivian	1	1.1	1.5	wv
468	11317944		RUN	vivian	n232-132-084	2023-12-15 15:15:00	vivian	1	0.6	1.5	verdi.run

5. 基于 Demo Case 快速使用 IFP

5.1 用户配置

5.1.1 EDA 配置文件

Demo Case 安装包请参考如下网盘链接进行下载安装:

链接: https://pan.baidu.com/s/1ni4u0iN7R5mwvb0DU12uvQ 提取码: demo

用户拿到的一般是一个 zip 格式的压缩安装包 demo_case.zip，将其放置到安装路径下，解压缩。

```Bash
[n232-135-013]: /ic/data/usr/jiexi/demo_test/$ ls
demo_case.zip
[n232-135-013]: /ic/data/usr/jiexi/demo_test/$ unzip demo_case.zip
[n232-135-013]: /ic/data/usr/jiexi/demo_test/$ ls
demo_case  demo_case.zip
```

安装完毕后，我们可以看到它的完整路径结构。

```Bash
[n232-135-013]: /ic/data/usr/jiexi/demo_test/$ tree -L 3 demo_case
.
demo_case
├── ifp_working
├── project_setting
│   ├── check
│   └── demo
├── setup
│   ├── build.py
│   ├── Makefile
│   └── module_load
└── tools
    ├── dc
    ├── fm
    └── pt
```

5.1.2 启动 IFP

用户进到 ifp_working 目录下，执行命令“python3 ../setup/build.py -yaml”生成用户配置文件 ifg.cfg.yaml（可用 cat 命令查看），然后启动 IFP 程序。

```Bash
#步骤 1：进入目录 ifp_working，生成 ifp.cfg.yam 并定义变量${DEMO_PATH}
[/ic/data/usr/jiexi/demo_test/demo_case/]$ cd ifp_working
[/DEMO_PATH/ifp_working]$ python3 ../setup/build.py -yaml
    COMMAND : write ifp.cfg.yaml
/ic/data/usr/jiexi/demo_test/demo_case/ifp_working/ifp.cfg.yaml
[/DEMO_PATH/ifp_working]$ cat ifp.cfg.yaml
VAR:
  DEMO_PATH: /ic/data/usr/jiexi/demo_test/demo_case

#步骤 2：启动 IFP 程序
[/DEMO_PATH/ifp_working]$ <IFP_INSTALL_PATH>/bin/ifp
```

5.1.3 IFP 用户配置文件

用户在“Setup -> Setting -> Project settings -> Project”设置 Project name=demo / User group=syn，勾选上 auto_import_tasks，这时候工具会自动匹配上 default.demo.syn.yaml，点击保存，如下：

用户在“Setup -> Setting -> System settings -> Cluster management”设置 $BSUB_QUEUE 队列名称，点击保存，如下：

然后用户再去 config 界面 create block，填写 Block（当前支持 bit_slice/bit_top）、Version 和 Branch 信息，并根据项目需求勾选上需要运行的任务，点击保存。

最后用户可以右键“Edit Task”编辑调整具体设置，点击保存即可。

5.1.4 配置任务间依赖关系

用户编辑完需要运行的任务后，需要为各 flow/task 之间指定依赖关系，可以通过“菜单栏 Setup -> Set Dependency”进行设置。

首先为 flow 间设置串并行关系，默认所有 flow 均并行。设置执行顺序为“syn -> fv|sta”，则可以直接在 dependency 处进行选择，右边会直接以树状图形式展示对应的依赖关系，方便用户一目了然，如下：

然后需要为各 flow 内部的所有 tasks 设置串并行关系，默认所有 task 均并行。设置方法同上。

设置完其中一个 block 后，如果其他 block 的 task 和 dependency 完全一致，可以在 block -> version 处右键选择“Apply Dependency To All”应用到其他 block。

如果其他 block 的 task 和 dependency 不一致，则用户需要点击 Dependency Setting 页面的 Block 和 Version 为不同 block/version 进行设置，设置完成后点击 Save 即可。

5.2 IFP 运行

5.2.1 Build

点击 ***Build*** 图标，完成目录创建。

```
Bash
[n212-206-218]: /ic/data/usr/jiexi/IFP_v1.2/ ls */*/
bit_slice/IFP_V1.2_test/:
```

```
datain  dc  fm  Makefile  module_load  pt

bit_top/IFP_V1.2_test/:
datain  dc  fm  Makefile  module_load
```

5.2.2 Run

点击 ***Run*** 图标，调用 EDA 工具提交 job。

注：用户需要确认下管理员是否配置了 EDA 环境（即 3.3 章节内容)，如果未配置，请通知管理员进行配置，然后再进行 job 提交操作。

若用户执行任务时需要开启交互界面，则可编辑 task 属性中的 RUN_METHOD 信息，如下为利用 xterm 界面进行交互。

注：如果用户开启了交互界面(如 xterm)，任务完成之前请勿关闭 xterm，否则 LSF 认为此 job 正常退出 EDA 工具，状态会更新为 RUN PASS，后续的任务会直接提交。

另外如果针对某些固定的 task(如 syn_dc)需要常开启交互界面，则管理员可以在 default.yaml 中统一设置，这样可避免用户单独进行配置。

5.2.3 Kill

点击 ***Kill*** 图标，终止选中的正在运行的 job，Status 从“Killing”变为“Killed”，则证明 job 完成终止，如下：

5.2.4 Check

点击 ***Check*** 图标，进行 checklist 项目检查。

点击对应 task check 列的图标，查看具体 checklist 检查项。

checklist 检查机制及配置方法，用户可参考 IFP 用户手册“附四 auto_check 检查机制及配置方法”内容，安装包中添加了针对该 demo case 使用的简单示例，详见 ${DEMO_PATH}/project_setting/check 目录，用户可直接将该路径下的检查脚本配置到 task CHECK 属性部分。

Flow setting :

	Item	Value
10	RUN_METHOD	
11	REQUIRED_LICENSE	
12	* CHECK	
13	PATH	
14	COMMAND	
15	RUN_METHOD	
16	VIEWER	
17	REPORT_FILE	

若用户在 task 属性中定义了 check 属性，点击 ***Check*** 图标，进行 checklist 项目

检查。

5.2.5 Summarize

点击 ***Summarize*** 图标，调用 qor 脚本自动收集 PPA 数据信息。

5.2.6 Release

选中需要 Release 的 task，点击 ***Release*** 图标，进行数据信息的交付。

release 数据结果如下:

```
11:49 [n232-134-194]: ~/ll /ic/proj/vivian/public/release/syn/vdec_top/vivian_initial_IFP_V1.2_S1225A
total 48
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 15 14:52 SAIF
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 15 14:52 DB
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 15 14:52 UPF
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 15 14:52 DEF
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 19 15:05 SDC
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 19 15:05 RPT
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 19 15:05 CONS
drwxr-xr-x  2 ic_admin ic_work_group 4096 Dec 25 10:52 NETLIST
drwxr-xr-x 10 ic_admin ic_work_group 4096 Dec 25 10:56 .
-rw-r--r--  1 ic_admin ic_work_group  185 Dec 25 10:56 .release
drwxr-xr-x  5 ic_admin ic_work_group 4096 Dec 25 10:56 ..
-rwxr-xr-x  1 ic_admin ic_work_group  806 Dec 25 10:56 release.20231225_105552_jiexi.log
```

5.2.7 操作视频

可以通过如下链接查看 IFP 的视频演示。

https://www.douyin.com/video/7202142518065073465

6. 技术支持

本工具为开源工具，由开源社区维护，可以提供如下类型的技术支持：

- 部署和使用技术指导。
- 接收 bug 反馈并修复。
- 接收功能修改建议。（需审核和排期）

获取技术支持的方式包括：

- 通过 Contact 邮箱联系开发者。
- 加入官方技术支持群。

附录

附一、变更历史

日期	版本	变更描述
2023.2.2	1.0	工具开源，第一个正式版本发布。
2023.7.14	1.1	修正部分操作逻辑 bug，优化 CONFIG 界面操作方式。
2023.8.31	1.1	优化菜单栏功能和界面操作方式。
2023.12.31	1.2	支持更为复杂的任务执行逻辑; 新增系统设置界面便于用户进行个性化设置。

附二、外部贡献者

感谢如下外部贡献者，为 IFP 提供了大量的 bug 反馈和代码优化建议，促进了 IFP 的快速迭代开发。

Soren Zhao

附三、DemoCase 案例介绍

1. 前言

该部分包含 Synopsys SYN/FORMALITY/PT 等流程的案例详细信息。所有适用的设计数据，包括 RTL、SDC、UPF、Testbench、技术库、详细的 Makefile 和工具脚本，都包含在参考包中。此软件包用户可通过网盘下载，并与该用户指南一起使用。

2. Flow 介绍

2.1 文件和目录

参考包的文件目录和结构如下:

```bash
.
├── ifp_working        ⟹ 用户启动 IFP 目录
├── project_setting        ⟹ 项目信息设置
│   ├── check          ⟹ checklist 示例
│   │   ├── fv        ⟹ fv 相关 checklist 脚本
│   │   ├── sta       ⟹ sta 相关 checklist 脚本
│   │   └── syn       ⟹ syn 相关 checklist 脚本
│   ├── demo          ⟹ project
│    ├── def      ⟹ DEF file for implementation
│    ├── lib      ⟹ Tech libraries
│    ├── rtl      ⟹ RTL Source Code
│    ├── sdc      ⟹ SDC Source Constraints
│    ├── tech     ⟹ Tech libraries
│    ├── upf      ⟹ UPF Source
│    ├── verification ⟹ Verification Models
│    └── verilog_pg   ⟹ PG Verification Models
├── setup              ⟹ 环境及 EDA 配置
│   ├── build.py      ⟹ 环境创建脚本
│   ├── Makefile      ⟹ Top level Makefile
│   └── module_load   ⟹ EDA 工具配置
└── tools              ⟹ flow 相关脚本
```

```
    ├── dc      ⟹ Design Compiler
    ├── fm      ⟹ Formality
    └── pt      ⟹ PrimeTime
```

2.2 工具版本

对于本参考设计，我们使用以下列出的工具版本。如果选择不同版本的工具，可能会得到与此处所述不同的结果。注意，我们在“setup”目录中包含了一个“module_load”文件，请修改此文件以适合您的环境，建议使用如 推荐(或相近)版本的 EDA 工具。

```Bash
module load dc_shell/T-2022.03-SP5-1
module load formal/T-2022.03-SP5
module load lc_shell/S-2021.06-SP4
module load pt_shell/R-2020.09-SP5-4
```

2.3 技术库信息

参考设计中包含的技术库是“saed32”Synopsys 库，用于示例目的。

```Bash
saed32lvt_dlvl_tt0p85v25c_i0p85v.db   saed32lvt_ss0p7vn40c.db
saed32lvt_ulvl_tt0p85vn40c_i0p85v.db
saed32lvt_dlvl_tt0p85v25c_i1p05v.db   saed32lvt_ss0p95v125c.db
saed32lvt_ulvl_tt1p05v125c_i0p78v.db
saed32lvt_dlvl_tt0p85vn40c_i0p85v.db  saed32lvt_ss0p95v25c.db
saed32lvt_ulvl_tt1p05v125c_i0p85v.db
saed32lvt_dlvl_tt0p85vn40c_i1p05v.db  saed32lvt_ss0p95vn40c.db
saed32lvt_ulvl_tt1p05v125c_i1p05v.db
saed32lvt_dlvl_tt1p05v125c_i1p05v.db  saed32lvt_tt0p78v125c.db
saed32lvt_ulvl_tt1p05v25c_i0p78v.db
saed32lvt_dlvl_tt1p05v25c_i1p05v.db   saed32lvt_tt0p78v25c.db
```

```
saed32lvt_ulvl_tt1p05v25c_i0p85v.db
saed32lvt_dlvl_tt1p05vn40c_i1p05v.db
```

3. 任务运行

3.1 目录创建

安装目录下的 setup 目录下包含了环境创建脚本 build.py，用户可运行 build 脚本完成对应流程的环境搭建工作。

```Bash
##COMMAND: python3 ${DEMO_PATH}/setup/build.py

[n232-135-013]$ python3
/ic/data/usr/jiexi/demo_test/demo_case/setup/build.py
    COMMAND : mkdir -p
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1
    COMMAND : ln -sfn
/ic/data/usr/jiexi/demo_test/demo_case/project_setting/demo
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/datain
    COMMAND : cp -rf
/ic/data/usr/jiexi/demo_test/demo_case/setup/module_load
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/module_load
    COMMAND : cp -rf
/ic/data/usr/jiexi/demo_test/demo_case/setup/Makefile
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/Makefile
    COMMAND : cp -rf
/ic/data/usr/jiexi/demo_test/demo_case/tools/*
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/.
```

- 以上命令会自动生成工作目录，var(run_dir) = <root_dir>/<design_name>/<VERSION>_<BRANCH>:

  e.g /ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1

- 同时会自动进行以下操作:

- 建立所有 sub dir。
- link block 需要的输入文件到 datain。
- Copy Makefile/module_load 等文件到$vars(run_dir)工作目录，用户可根据项目修改 EDA 工具版本。

```
Bash
[n232-135-013]: $ cd
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1
total 32K
4.0K drwxr-xr-x 3 jiexi Nov  7 20:46 ..
0 Nov 7 20:46 datain ->
/ic/data/usr/jiexi/demo_test/demo_case/project_setting/demo
4.0K drwxr-xr-x 9 jiexi Nov  7 20:46 dc
4.0K drwxr-xr-x 6 jiexi Nov  7 20:46 fm
4.0K drwxr-xr-x 6 jiexi Nov  7 20:46 pt
4.0K -rw-r--r-- 1 jiexi Nov  7 20:46 module_load
4.0K -rw-r--r-- 1 jiexi Nov  7 20:46 Makefile
4.0K drwxr-xr-x 6 jiexi Nov  7 20:46 .
```

其中 build.py 的用法如下:

```
Bash
usage: build.py [-h] [-p PROJECT] [-f FLOW] [-d DESIGN]
 [-v VERSION] [-b BRANCH]
optional arguments:
  -h, --help         show this help message and exit
  -p PROJECT, --project PROJECT
              specify project, default is "demo".
  -f FLOW, --flow FLOW
              specify project, default is "[syn, fv, sta]".
  -d DESIGN, --design DESIGN
              specify design name.
  -v VERSION, --version VERSION
              specify run version, default is "IFP".
```

```
  -b BRANCH, --branch BRANCH
                specify run branch, default is "branch1".
  -yaml, --ifp_cfg_yaml
                Enable yaml mode, write ifg_cfg_yaml.
#####帮助信息
-p，project，指定项目名，默认“demo”。
-f，flow，指定 flow 名，默认 syn/fv/sta 目录都创建。
-d，design，指定 design 名称，默认“bit_slice”。
-v，version，指定版本信息，默认“IFP”。
-b，branch，指定 branch 名，默认“branch1”。
-yaml，ifp_cfg_yaml，仅用于生成 IFP 用户配置文件，并写入变量
${DEMO_PATH}。
```

您可以在“bit_slice”级别（当前默认）运行，以便快速了解如何运行工具。如果想要更复杂的例子，可以在“bit_top”级别运行（即指定-d bit_top）。

3.2 任务运行

此 demo case 基于 Makefile 系统的 target 管理模式，可以用“make help”命令查看支持的所有选项。

```Bash
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ make help
Description:
Examples:
 * The following command
     make  syn_dc
     make  debug_syn_dc
     make  clean_dc
     make  fm_rtl2gate
     make  fm_rtl2pg
     make  clean_fm
     make  presta
     make  clean_sta
```

Synthesis 流程：

跑完 syn 流程大约需要 10 分钟。

```
Bash
cd $var(run_dir)
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ source
module_load
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$
bsub -Is -q comet "make syn_dc"
```

Formal 流程：

formal 流程会基于综合流程 dc 的 netlist 结果进行比对，所以需要检查综合网表是否生成$<run_dir>/dc/results_bit_slice/bit_slice.vg。

```
Bash
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/$ source
module_load
/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/&
bsub -Is -q comet "make fm_rtl2gate"

####
#rtl2gate: function rtl vs synthesis netlist
#rtl2pg: rtl vs pr netlist
```

Sta 流程：

presta 流程会基于综合流程 dc 的 netlist 结果运行：

```
Bash
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ source
module_load
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$
bsub -Is -q comet "make presta"
```

3.3 结果检查

- Syn 的报告和输出文件分别生成在文件夹 logs、reports 和 results_bit_slice 目录下，用户可以 gvim 打开 review，如下：

```Bash
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree dc/logs
dc/logs
├── log.dc.bit_slice
└── snps_bit_slice.upf.log

[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree dc/reports/
dc/reports/
├── bit_coin.check_library.rpt
├── bit_slice_bam.area.rpt
├── bit_slice_bam.cmv.rpt
├── bit_slice_bam.iso.rpt
├── bit_slice_bam.ls.rpt
├── bit_slice_bam.report.scan_def
├── bit_slice_bam.ret.rpt
├── bit_slice.cmv.rpt
├── bit_slice_dft_conf.rpt
├── bit_slice_dft.dft_signal.rpt
├── bit_slice_dft_drc.rpt
├── bit_slice_dft.report.scan_def
├── bit_slice_post_dft.cmv.rpt
├── bit_slice.power.rpt
├── bit_slice_preview_dft_all.rpt
├── bit_slice_preview_dft.rpt
├── bit_slice.qor.rpt
├── bit_slice.report.scan_def
├── bit_slice_scan_conf.rpt
└── bit_slice.tim.rpt

[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree dc/results_bit_slice/
dc/results_bit_slice/
```

```
├── bit_slice_bam.ctl
├── bit_slice_bam.ddc
├── bit_slice_bam.scan_def
├── bit_slice_bam.test.ddc
├── bit_slice_bam.upf
├── bit_slice_bam.vg
├── bit_slice_compiled.ddc
├── bit_slice_compiled.script
├── bit_slice_compiled.sdc
├── bit_slice_compiled.sdf
├── bit_slice_compiled.spef
├── bit_slice_compiled.upf
├── bit_slice_compiled.vg
├── bit_slice.ctl
├── bit_slice.ddc
├── bit_slice_dft.ctl
├── bit_slice_dft.ddc
├── bit_slice_dft.internal_scan.spf
├── bit_slice_dft.scan_def
├── bit_slice_dft.script
├── bit_slice_dft.sdc
├── bit_slice_dft.sdf
├── bit_slice_dft.spef
├── bit_slice_dft.test.ddc
├── bit_slice_dft.upf
├── bit_slice_dft.vg
├── bit_slice.name_map
├── bit_slice_rc_compiled.tcl
├── bit_slice_rc_dft.tcl
├── bit_slice_rc.tcl
├── bit_slice.scan_def
├── bit_slice.script
├── bit_slice.sdc
├── bit_slice.sdf
├── bit_slice.spef
├── bit_slice.svf
├── bit_slice.test.ddc
```

```
├── bit_slice.upf
└── bit_slice.vg
```

可以 gvim 打开 qor.rpt 查看 design 运行的 timing 和 area 报告等。

- Formal 的相关文件分别生成在文件夹 reports、logs 和 fm_sess 目录下，如下：

```
Bash
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree fm/reports
fm/reports
└── bit_slice.fm.rtl2gate.rpt

[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree fm/fm_sess
fm/fm_sess
└── fm_sess_bit_slice_rtl2gate.fss

[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree fm/logs
fm/logs
└── log.fm.bit_slice_rtl2gate
```

- Sta 的相关文件分别生成在文件夹 logs 和 reports_bit_slice 目录下，如下：

```
Bash
[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree pt/logs
```

```
pt/logs
└── log.pt.bit_slice

[/ic/data/usr/jiexi/demo_test/bit_slice/IFP_branch1/]$ tree
pt/reports_bit_slice
pt/reports_bit_slice
├── bit_slice_global_cons.rpt
├── bit_slice_global_timing.rpt
├── bit_slice_power_average.rpt
├── bit_slice_qor.summary.rpt
└── bit_slice_timing.rpt
```

附四、auto_check 检查机制及配置方法

1. 检查机制介绍

IFP 集成了一套自动化 checklist 检查机制，用户仅需要提供一份 Excel 格式的描述，系统工具可以将之转换为对应的 checklist 检查脚本，在 IFP 做 check 检查的时候能自动调用这些 checklist 脚本做任务结果检查。

这种 checklist 机制极大地简化了用户的 checklist 检查操作，减少了手动编写和维护检查脚本的工作量，并且降低了复杂 checklist 检查的实现难度。 用户可以更专注于检查内容的定义和调整，而无需关注底层的脚本实现细节。

如下是一份 Excel 格式的 syn 流程的 checklist 配置（此 Excel 格式 的 checklist 配置文件可以按照自己的需求修改）。

TASK	DESCRIPTION	FILE	MESSAGE	TYPE
syn_dc	Check "Error" message on log file.	log/syn_dc.log	^Error	check_error_message
	Check "Error" message on log file.	log/00_analyze.log log/01_elaborate.log log/02_link_design.log log/03_read_sdc.log log/04_compile1.log log/05_compile2.log log/06_final.log log/07_check_dont_touch_cell.log log/08_dump_func_sdc.log	^Error	check_error_message
	Check "Warning" message on log file.	log/syn_dc.log	^Warning	check_warning_message
	Get RTL version.	log/syn_dc.log	vars\(RTL_VERSION\)	check_warning_message
	check timing rpt	report/<BLOCK>.final.check_timing.rpt		review_file
	check_dont_use cell report	report/<BLOCK>.dont_use_lib_cell.rpt		review_file

针对 Excel 表格中的 TASK/DESCRIPTION/FILE/MESSAGE/TYPE 项，如下是 checklist 格式说明。

NOTICE	
TASK	Specify task name. For example, syn_dc/initopt/finalopt.
DESCRIPTION	Description for the qualify item. Please use "" instead of '' on the message.
FILE	Specify file(s) you want to check.
MESSAGE	Specify message(s) you want to check. Ignore it if the TYPE is "check_file_exdist" or "review_file".
TYPE	What kind of type you want to check, it supports below types:
	check_error_message : Check error message(s) on specified file. Fail if exist, pass if not exist.
	check_warning_message : Check warning message(s) on specified file.
	check_expected_message : Check specified message(s) on specified file. Pass if exist, fail if not exist.
	check_file_exist : Check file(s) exist or not. Pass if exist, fail if not exist.
	review_file : Review specified file(s).
PS	* Support variables <BLOCK> and <TASK>.
	* Only three results : PASSED / FAILED / REVIEW

Excel 包含五部分内容：

1. **TASK** ： 指定流程步骤，比如 syn 的 TASK 有 syn_dc/syn_dcg；一个 TASK 可以

有多行 checklist 项，此时应该把对应的 TASK 单元格合并；如果 TASK 前面有“#”字符，标明这个 TASK 行是注释行，暂不开放，脚本会忽略它。

2. **DESCRIPTION** ： checklist 项描述。DESCRIPTION 说明本 checklist 项检查什么，需要给与清晰明确的说明。

3. **FILE** ： checklist 检查文件。每个 checklist 项可以同时检查多个文件，文件间用空格隔开。

4. **MESSAGE** ： checklist 检查信息。从前面 FILE 指定的文件中检索指定的信息，可以同时检索多条信息，每条信息用双引号“”括起来，多条 MESSAGE 之间用空格隔开。

5. **TYPE** ： 检查类型，当前主要包含以下五种：

- check_error_message：检查指定文件中是否存在指定错误信息。存在则失败，不存在则成功。
- check_warning_message： 检查指定文件中是否存在指定的警告信息。文件存在则抓取相关信息来 review。
- check_expected_message： 检查指定文件中是否存在指定的信息。存在则成功，不存在则失败。
- check_file_exist： 检查指定文件是否存在。存在则成功，不存在则失败。
- review_file：查看指定文件内容。文件存在则 review 其内容。

如上几种检查类型可以涵盖 大多数的检查需求，其它类型的检查需求则需要借助辅助工具做定制化处理。

针对“检查错误”这个常规项，下面是一行示例信息。

- DESCRIPTION : Check “Error” message on log file.
- FILE : log/syn_dc.log (需要检查的文件)
- MESSAGE : “^Error” ”no such element in array” (需要检查的错误信息，可以检查多条)
- TYPE : check_error_message (检查类型：检查指定的错误信息是否存在，如果存在，检查结果为 FAIL)

备注：

- Excel 默认支持两个变量<BLOCK>（模块名）和<TASK>（任务名），这些变量在 checklist 脚本执行的时候会直接替换成相应的值。

- 所有的检查只有 3 种可能的结果，PASSED / FAILED / REVIEW。

2. checklist 配置方法

用户定义 Excel 格式的 checklist 项，以如下 syn_dc_checklist.xls 为例（**Excel 请在 Linux 中采用英文模式编辑，不要引入全角字符**）。

TASK	DESCRIPTION	FILE	MESSAGE	TYPE
syn_dc	Check "Error" message on log file.	log/syn_dc.log	^Error	check_error_message
	Check "Error" message on log file.	log/00_analyze.log log/01_elaborate.log log/02_link_design.log log/03_read_sdc.log log/04_compile1.log log/05_compile2.log log/06_final.log log/07_check_dont_touch_cell.log log/08_dump_func_sdc.log	^Error	check_error_message
	Check "Warning" message on log file.	log/syn_dc.log	^Warning	check_warning_message
	Get RTL version.	log/syn_dc.log	vars\(RTL_VERSION\)	check_warning_message
	check timing rpt	report/<BLOCK>.final.check_timing.rpt		review_file
	check_dont_use cell report	report/<BLOCK>.dont_use_lib_cell.rpt		review_file

3. Checklist Excel -> Checklist scripts 转换方式

通过调用 switch 脚本 <IFP_INSTALL_PATH>/action/check/scripts/gen_checklist_scripts.py 将 checklist Excel 转换为 checklist script。生成的（python）checklist script 就可以用到 flow task 的运行目录，用于结果检查。

```
SQL
[n212-206-218]:
<IFP_INSTALL_PATH>/action/check/scripts/gen_checklist_scripts.py -
i syn_dc_checklist.xls -f syn -v synopsys
>>> Generating checklist script for syn task "syn_dc" ...
/ic/data/usr/jiexi/cad_tools/flows/ic_flow_platform/action/check/s
yn/synopsys/syn_synopsys.syn_dc.py
```

其中 gen_checklist_scripts.py 脚本用法如下：

```
Bash
[liyanqing.1987@cmp207 ~]$ gen_checklist_script.py -h
usage: gen_checklist_scripts [-h] -i INPUT -f FLOW -v VENDOR [-o
```

```
OUTDIR]

optional arguments:
  -h, --help         show this help message and exit
  -i INPUT, --input INPUT
                Specify the input excel file.
  -f FLOW, --flow FLOW  Specify the checklist flow.
  -v VENDOR, --vendor VENDOR
                Specify the checklist vendor.
  -o OUTDIR, --outdir OUTDIR
                Specify the checklist script output directory,
default is current directory.
```

-i INPUT，必选项，指定输入 Excel 文件。

-f FLOW，必选项，指定流程名称，一般为 dft/syn/fv/sta/pr/cts/...等。

-v VENDOR，必选项，指定 vendor 名称，用于标识 EDA 工具所属的 vendor。（同一个 flow，不同 vendor 可能都有竞品工具，其 checklist 项是不一样的）

-o OUTDIR，可选项，指定 checklist scripts 的输出路径，默认为当前路径。

4. 执行 checklist script

用户拿到 checklist script，将其配置到 IFP 对应的 task 属性属性中。

备注：如上配置中包含的脚本使用方法介绍如下：

- view_checklist_report 的路径和使用方法如下：

  路径：${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report

  帮助信息：

```
Bash
[n232-134-195]$ view_checklist_report -h
usage: view_checklist_report [-h] [-i INPUT]

optional arguments:
  -h, --help          show this help message and exit
  -i INPUT, --input INPUT
                Specify the checklist report, default is
"./file_check.rpt" or "file_check/file_check.rpt".
```

-i INPUT，必选项，指定输入 checklist report。

- syn_synopsys.syn_dc.py 使用方法如下：

```Bash
[n232-134-195]: syn_synopsys.syn_dc.py -h
usage: syn_synopsys.syn_dc.py_b [-h] [-b BLOCK] [-t TASK] [-c
CORNER]

optional arguments:
  -h, --help          show this help message and exit
  -b BLOCK, --block BLOCK
                Specify block name.
  -t TASK, --task TASK  Specify task name.
  -c CORNER, --corner CORNER
                Specify corner name
```

-b BLOCK，指定模块名。

-t TASK，指定任务名。

-c CORNER，指定 corner 名称

5. 汇总并展示 checklist 报告

task 运行完成后，点击 check 按钮，自动执行脚本并在 checklist script 的执行目录下，会生成所有 checklist 项的检查日志，以及最终的检查报告。 点击对应 task check 列的图标，查看具体 checklist 检查项。

如果 Result 是 FAILED/REVIEW，那么用户可以直接点击对应行的“view”按钮，就会弹出 xtem 方便用户直接查看具体的 checklist 日志（或者要 review 的文件内容）。

checklist 检查汇总报告的格式为：**Result　Description　Detail**

- Result 列，PASSED 结果为绿色，FAILED 结果为红色，REVIEW 结果为橙色，从颜色即可很容易分辨出最终结果（有误失败项）。
- Description 列，是 checklist 项描述，所以这个描述一定要尽量客观详细。
- Detail 列，如果 Result 是 FAILED/REVIEW，那么用户一般需要检查一下 错误原因/review 信息，此处可以看到一个“review”按钮，点击 review 按钮，可以看到具体的 checklist 日志（或者要 review 的文件内容）。