IC Flow Platform 管理员手册

Product Name : IFP (IC Flow Platform)

Product Version : V1.4.3

Release Date : 2025.12.10

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)

前言

IC Flow Platform （下文中简称 IFP）是 ByteDance 开源的芯片设计流程平台，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。

IFP 是基于项目和用户组的一站式工作平台，能够适配各种不同的设计环境（DV/DFT/SYN/PD/K 库等），通过层次解耦的设计思路拆分方法学和自动化流程。提供集约化的数据管理和展示，大幅提高多项目并行、多用户协作的工作效率。在统一管理的前提下，通过丰富的 API 功能提供高自由度的拓展性，从而收敛日常工作中与主流程相关的辅助工作，满足不同用户的需求。

管理员请参考 《IC Flow Platform 管理员手册》配置项目组/用户组的默认设置和 API 功能。

用户请参考《IC Flow Platform 用户手册》在管理员默认配置基础上使用 IFP 进行日常工作。

1. 工具安装

1.1 环境依赖

1.1.1 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。

1.1.2 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 **python3.8.8** 以解决库依赖问题。

不同版本的 python 会有 python 库版本问题。

1.1.3 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/volclava 集群管理系统。

1.1.4 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。

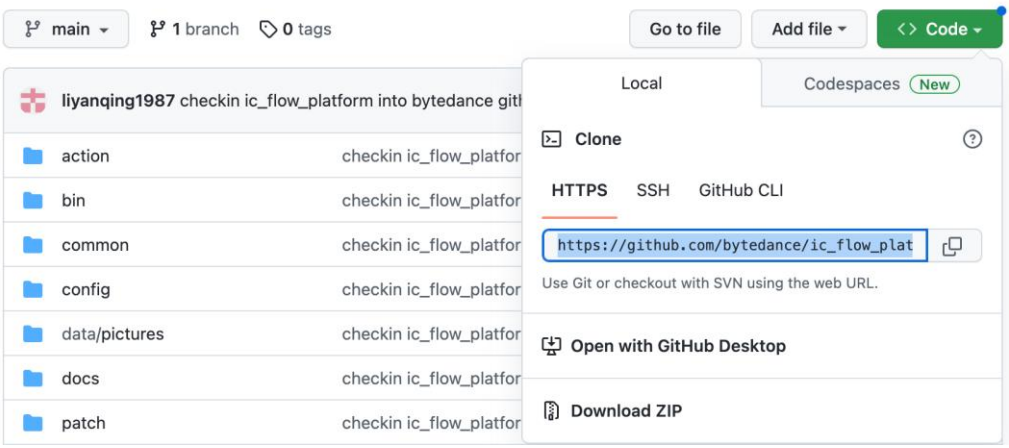
1.2 下载

IFP 的 github 路径位于 https://github.com/bytedance/ic_flow_platform，可通过如下两种方式下载代码。

（1）执行命令：

git clone https://github.com/bytedance/ic_flow_platform.git

（2）*Code -> Download ZIP* 的方式。



1.3 安装

（1）将安装包拷贝到安装目录，并给与合适的目录名。

（2）确认是否满足 *章节 1.1 环境依赖* 要求。

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

pip3 install -r requirements.txt

（4）在安装目录下，执行如下命令安装 IFP。

python3 install.py

2. 管理员配置

IFP 推荐的工作模式是：管理员为**不同的项目、用户组**配置 IFP 的**默认**系统设置、环境变量、流程和日常工作所需的辅助工具接口，而用户仅需输入所在的项目、用户组即可匹配到管理员的默认配置，在界面上简单的微调即可开始日常工作。建议管理员按照如下顺序配置：

（1）根据 *章节 2.1 系统配置* 完成系统命令、IFP 执行逻辑等默认设置

（2）根据 *章节 2.2 环境配置* 完成 EDA 版本、系统变量等默认配置

（3）根据 *章节 2.3 流程配置* 完成指定项目、指定用户组默认流程、相关属性、依赖关系的配置

（4）根据 *章节 2.4 API 配置* 完成日常工作中常用辅助功能的配置

（5）根据 *章节 2.5 Wrapper 配置* 开发 wrapper 功能自动生成 ifp.cfg.yaml（该文件用于记录和加载用户 IFP 相关的信息），从而大大简化用户配置流程。

2.1 系统配置

通过*${IFP_INSTALL_PATH}/config/config.py* 配置默认的系统设置。

2.1.1 命令与路径

system_log_path

（可选）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。例如：

```
system_log_path = '/ic/data/CAD/flows/ic_flow_platform/system.log'
```

lmstat_path

（可选）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。例如：

```
lmstat_path = "/ic/software/cad_tools/it/licenseMonitor/tools/lmstat"
```

in_process_check_server

（可选）用于配置任务进程中检测服务的服务器地址。配置后，可在任务执行过程中进行远程进程状态检查，若不配置，则不会进行进程状态校验。例如：

```
in_process_check_server = " http://10.0.0.1:8000"
```

send_result_command

（可选）Run Action 结束后执行的命令，一般用于给用户发送汇总结果。支持 RESULT 和 USER 两个变量，其中 RESULT 会被替换为 Task 当前的状态，USER 会被替换为当前用户名。例如：

```
send_result_command = "/ic/software/cad_tools/bin/send_lark -c RESULT -r USER"
```

xterm_command

用于指定 IFP 交互终端模式，可配置 xterm/gnome-terminal/konsole 等。例如：

xterm_command = "/bin/gnome-terminal --tab -- bash -c"

2.1.2 行为开关

mem_prediction

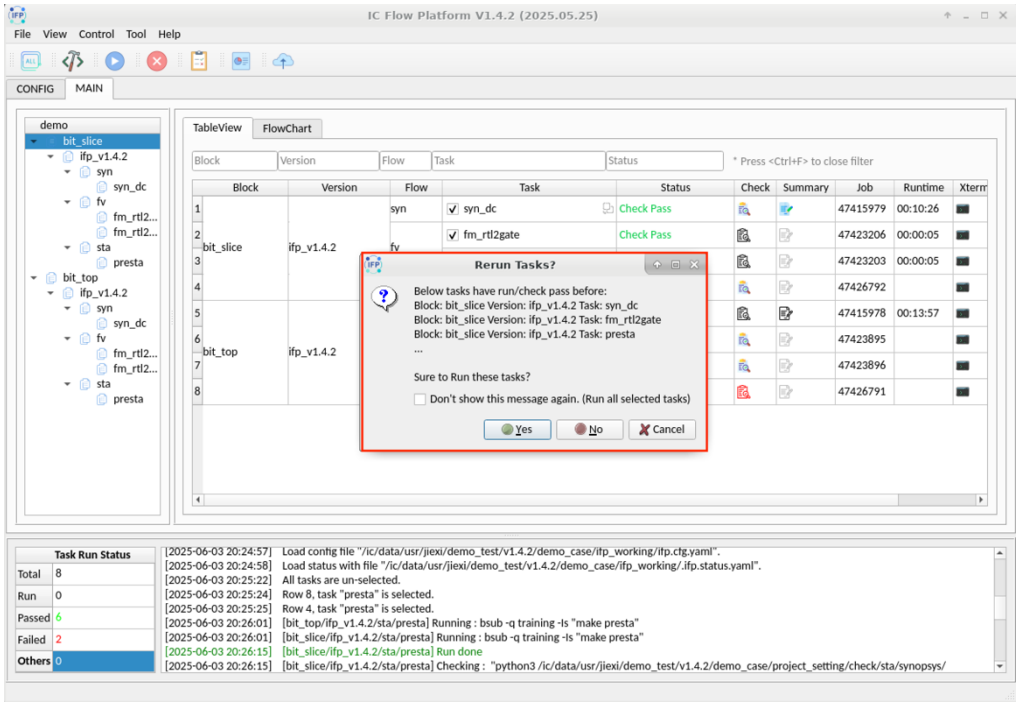
用于指定是否启用内存预测功能。开启后，在执行任务前将预测内存使用情况，并做出相应提示或限制。默认是 False。

fullscreen_flag

用于指定启动 IFP GUI 界面是否为全屏模式，默认是 True。

rerun_flag

当任务已经 Pass，用户再次点击 Run Action 时，是否弹窗提醒，默认是 True。



ignore_fail

当前置任务 Fail 时，当前任务是否继续执行，默认是 False。

send_result

当任务结束时是否调用 *send_result_command* 发送通知，默认是 False。

auto_check

当任务 *Run* 结束后是否自动跑 *Check*。

auto_import_tasks

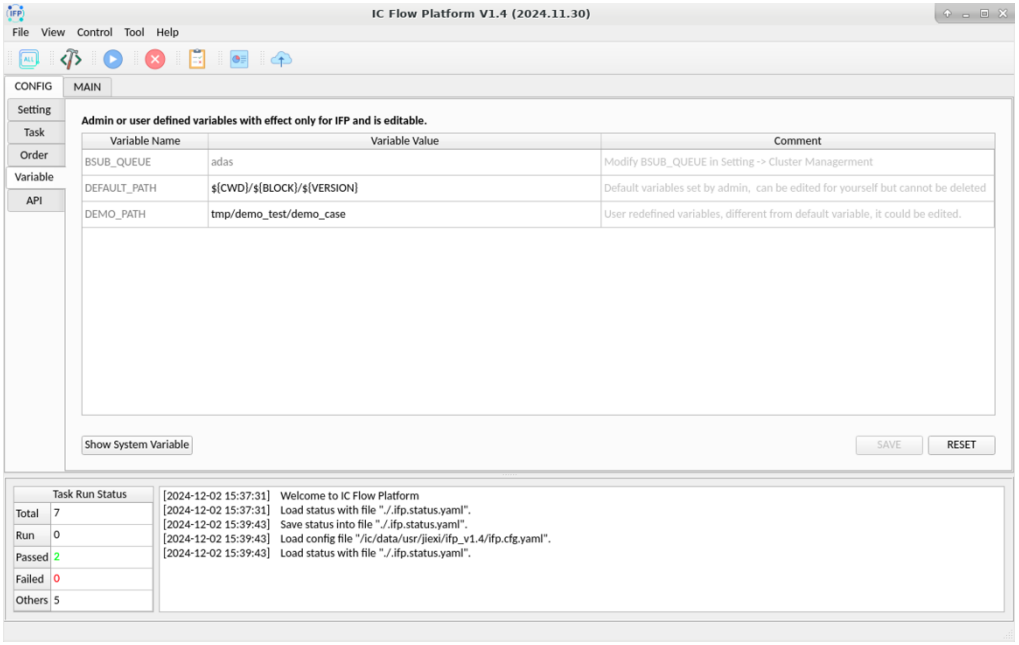
当用户在 *CONFIG-Task 界面* 新增 block/version 时是否会自动导入管理员配置的默认流程，默认是 True。

rerun_check_or_summarize_before_view

在 *MAIN 界面* 点击 *Check/Summarize 图标* 查看报告之前，是否重新执行 Check/Summarize 操作，默认是 True。

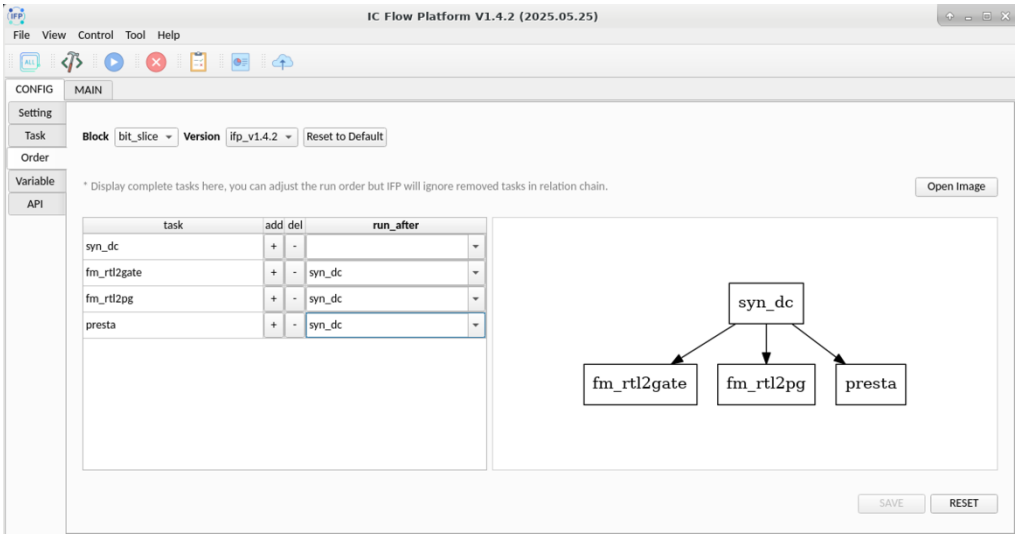
enable_variable_interface

是否启用 *Variable 界面*



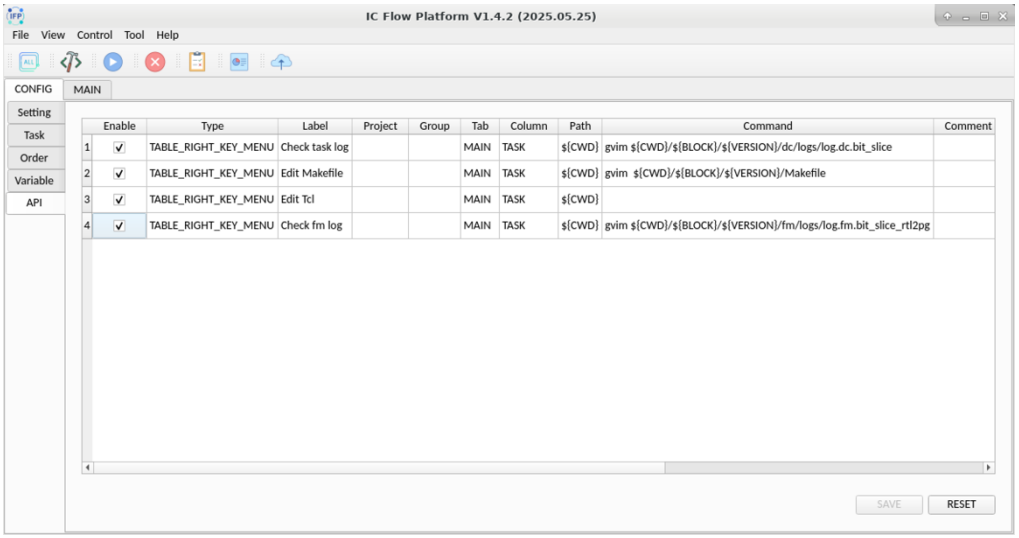
enable_order_interface

是否启用 *Order 界面*



enable_api_interface

是否启动 *API 界面*



makefile_mode

在任务提交时自动分析并选择所需的前置任务，实现任务间的自动依赖管理，默认是 False。

2.2 环境配置

通过*${IFP_INSTALL_PATH}/config/env.** 配置默认的环境设置。

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，一般用于指定 EDA 工具版本。例如:

```
# Set default INNOVUS setting.
  module load innovus/22.34-s061_1
# Set lsfMonitor path.
  export PATH=~/tools/ic_flow_platform/tools/lsfMonitor/monitor/bin:$PATH
```

IFP 启动时会 source 上述文件，用于在环境中配置管理员指定的工具。

2.3 流程配置

通过*${IFP_INSTALL_PATH}/config/default.yaml* 配置默认的流程设置。

2.3.1 项目与用户组

管理员可通过 default.yaml 的命名为不同的项目与用户组配置不同的默认设置。例如:

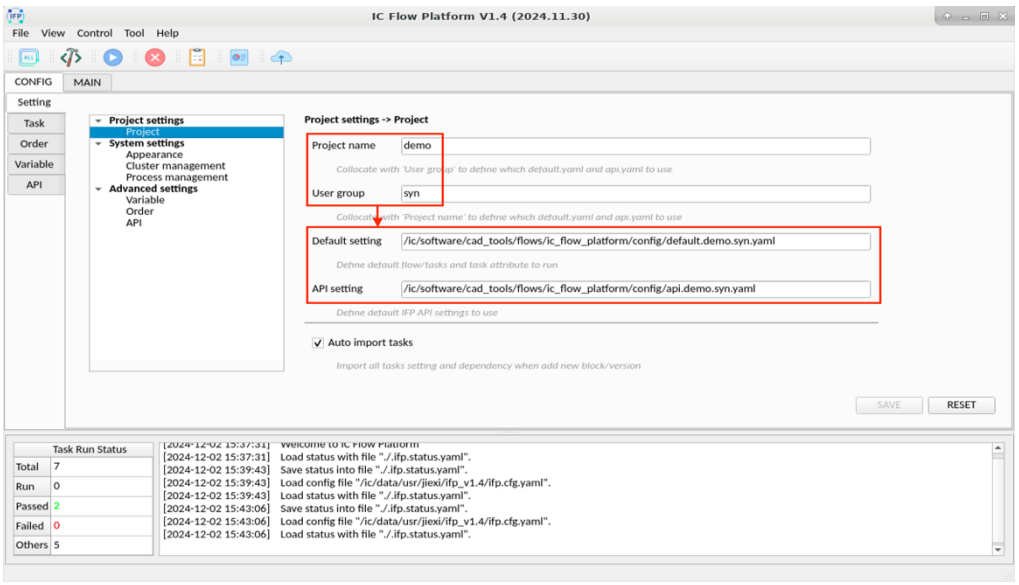
default.yaml ： 适用于所有项目和用户组的成员

default.syn.yaml ： 适用于所有项目和 syn 组的成员

default.projectA.yaml ： 适用于 projectA 和所有用户组的成员

default.projectA.syn.yaml：仅适用于 projectA 和 syn 组的成员

用户则通过 *CONFIG-Setting 界面* 中的 *Project name* 和 *User group* 来匹配管理员的默认设置:



2.3.2 变量

IFP 会自动识别和替换的变量类型包括：

（1）系统环境变量：如 bash 环境中用户 export 的变量

（2）管理员定义在 default.yaml 中的内置变量：

这些变量仅在 IFP 内部可调用，方便简化其他路径、命令的定义。例如：

```
VAR:
    BSUB_QUEUE: ai_syn
    DEFAULT_PATH: ${CWD}/${BLOCK}/${VERSION}/${FLOW}
    MAX_RUNNING_JOBS: 6
```

（3）用户自定义的内置变量

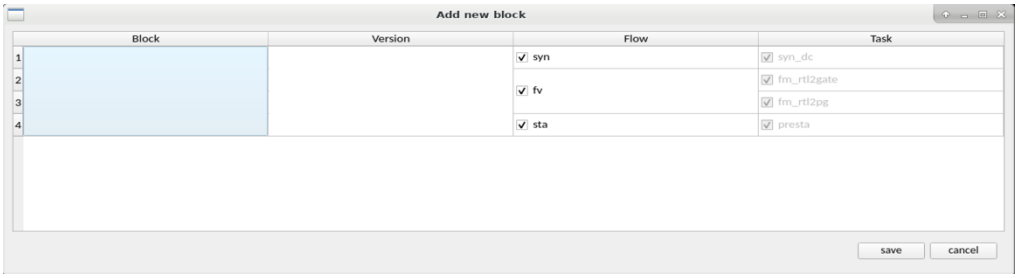
（4）IFP 专有变量：

IFP 将自动识别命令中的 *${IFP_INSTALL_PATH} / ${CWD} / ${BLOCK} / ${VERSION} / ${FLOW} / ${TASK} / ${USER}* 并替换为当前触发的相关对象属性。

2.3.3 工作流程

任务

为指定的 *Task* 定义固定的行为和属性，用户仅需勾选上 *auto_import_task* 即可自动导入管理员为指定项目和用户组配置的流程。



行为

每个 *Task* 包含九种属性：

（1）其中 *COMMON* 在特殊场景自动生效

（2）除此以外拥有五种 Action，分别对应 MAIN 界面五个 Button：

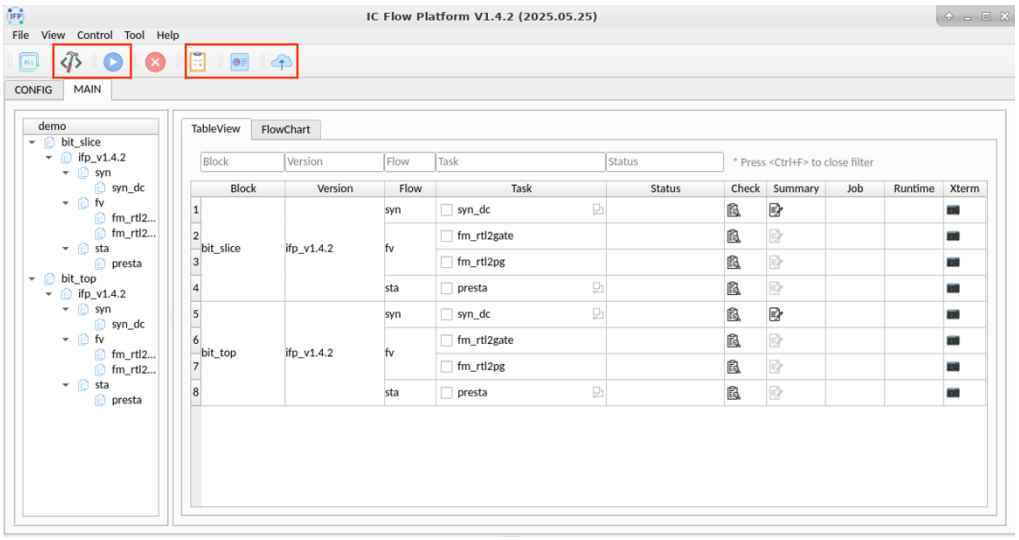
BUILD Action ： 创建该 Task 的运行环境（可选）

RUN Action ： 执行 EDA 的主体任务（必须）

CHECK Action ： 检查 Run 的结果（可选），若配置，任务 *Run* 结束后将自动跑 *Check*

SUMMARIZE Action ： 总结报告（可选）

RELEASE Action ： 发布数据（可选）



（3）*RUN_MODE* 属性，用于选择默认的运行模式

（4）*RUN_AFTER* 属性，用于定义 TASK 之间的执行顺序

（5）*DEPENDENCY* 属性，用于定义 TASK 执行的前置依赖

属性

（1）*COMMON Action* 包含 *XTERM_COMMAND（用户点击 Xterm 图标时在交互终端执行的命令）* 属性。

（2）除此以外的每个 Action 包含 *PATH（执行路径）/ COMMAND（执行命令）/ RUN_METHOD（执行方式）/ LOG（日志路径，支持配置多个日志文件，仅适用于 RUN）/ VIEWER（查看报告的工具，仅适用于 CHECK/SUMMARIZE）/ REPORT_FILE（报告路径，仅适用于 CHECK/SUMMARIZE）* 属性。

（3）*RUN_MODE* 包含运行模式选择属性

（4）*RUN_AFTER* 包含 *TASK* 属性 （字符串）

（5）*DEPENDENCY* 包含 *FILE* 和 *LICENSE* 属性（列表）

在设定时可调用 *章节 2.3.2 变量* 中定义的各种变量：

```
VAR:
    BSUB_QUEUE: ai_syn
    SYN_MODE: RUN
TASK:
    syn_dc:
        COMMON:
            XTERM_COMMAND: echo “Xterm mode for IFP”
        BUILD:
            PATH: ${DEFAULT_PATH}
            COMMAND: mkdir ${DEFAULT_PATH}
        RUN:
            PATH: ${DEFAULT_PATH}
            COMMAND: make syn_dc run_local=1
            RUN_METHOD: bsub -q ${BSUB_QUEUE} -Is
            LOG: ${DEFAULT_PATH}/logs/syn_dc.log, ${DEFAULT_PATH}/logs/read_lib.log
        RUN.DBG:
            PATH: ${DEFAULT_PATH}
            COMMAND: make restore_dc
            RUN_METHOD: bsub -q ${BSUB_QUEUE} -Is
            LOG: ${DEFAULT_PATH}/logs/restore_dc.log
        IN_PROCESS_CHECK:
            ENABLE: True
            INTERVAL: 1800
            COMMAND: ${check_script}
            PATH: ${CWD}
            NOTIFICATION: ${notification}
        CHECK:
            PATH: ${DEFAULT_PATH}/syn_dc
            COMMAND: python3 ${IFP_INSTALL_PATH}/action/check/syn/synopsys/syn_synopsys.syn_dc.py -b ${BLOCK}
                     -t ${TASK}
            VIEWER: python3 ${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report.py
            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: ${DEFAULT_PATH}/syn_dc/syn_qor.xlsx
    RUN_MODE：${ SYN_MODE }
    RUN_AFTER：
        TASK：init&pre-check
    DEPENDENCY:
      FILE:
        - ${DEFAULT_PATH}/syn_dc/Netlist
      LICENSE:
        - ${Feature_name} 50
```

IN_PROCESS_CHECK 用于为任务定制运行过程中检查机制。其中属性介绍如下：

ENABLE: True # 默认为 True，用户不启用，可置为 False

INTERVAL: 300 # 用于指定检查间隔（单位为 s），如果置空，那该项检查仅执行一次

COMMAND: ${CEHCK_SCERIPT} # 执行的 Check 脚本，py/sh 脚本均可

NOTIFICATION: {notification} # 添加提醒操作。

RUN_MODE 支持配置多种运行模式：不同运行模式可以关联不同的执行路径/执行命令/执行方式/日志路径。

例如上述的 RUN 和 RUN.DBG，管理员根据需求预先配置好可供用户选择的运行环境模式，然后可以在 VAR 中通过变量的方式来统一调整所有 TASK 的 RUN_MODE（用户也可在 GUI 界面中选择合适模式），系统将根据选定模式执行对应命令。

RUN_METHOD 支持多种模式：

（1）当前终端本地执行：不填写任何内容

（2）通过集群管理软件提交：bsub -q normal -n 5 (IFP 会自动加上-I 参数)

（3）通过交互式终端提交：xterm -e 或 terminator -e

（4）从交互式终端中通过集群管理软件提交：xterm -e bsub -q normal -n 5

注意：如果该 COMMAND 需要终端让用户输入命令进行交互，请务必使用（3）（4）方式，否则会造成找不到交互式终端而结束任务

注意：IFP 通过 COMMAND 的退出码判断 Action 是 Pass/Fail，务必准确使用退出码!

LOG 支持配置多个日志文件：

（1）可直接在界面配置

（2）通过逗号分隔多个日志路径：LOG: ${DEFAULT_PATH}/logs/syn_dc.log, ${DEFAULT_PATH}/logs/read_lib.log

（3）通过列表形式配置多个日志文件：LOG: [‘${DEFAULT_PATH}/logs/syn_dc.log’, ‘${DEFAULT_PATH}/logs/read_lib.log’]

RUN_AFTER 支持多种场景：

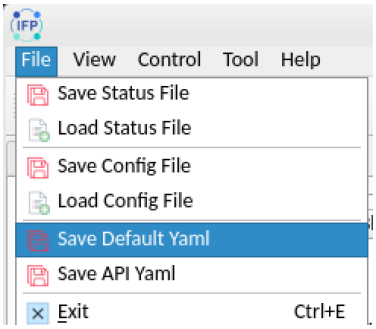
（1）前置任务都完成后才可以启动当前任务： *A&B*

（2）前置任务中有一个完成后即可启动当前任务： *A|B*

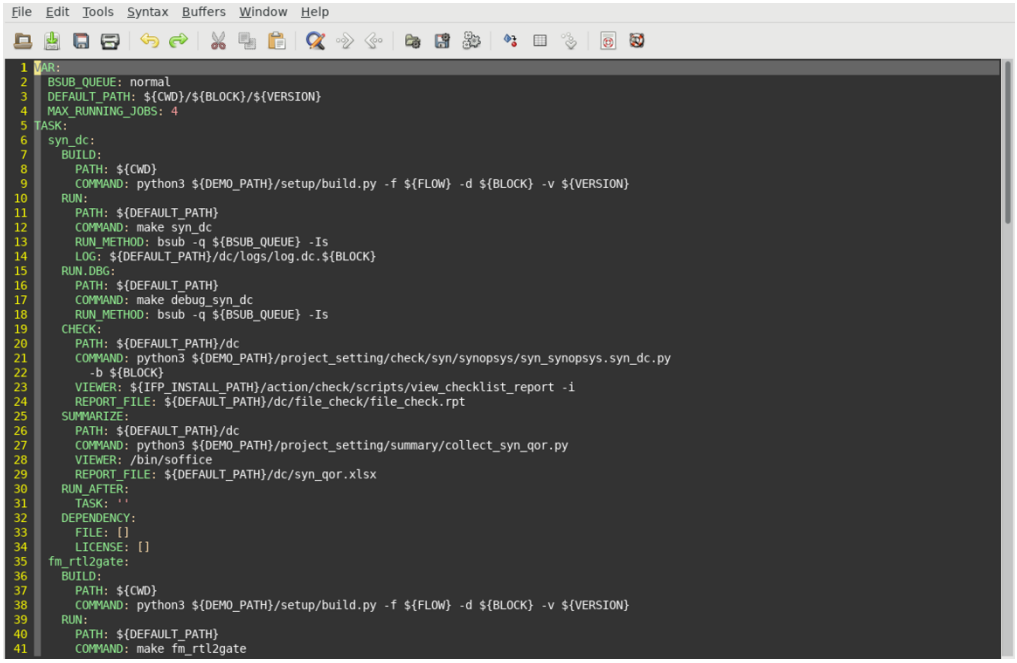
（3）前置任务完成后可多次启动当前任务： *A,B*

2.3.4 通过界面配置

管理员可通过菜单栏 *File -> Save Default Yaml* 将当前所有任务的配置导出为 default.yaml，需确保 block 的唯一性。



如下导出为 default.syn.yaml，格式如下。



2.4 API 配置

通过*${IFP_INSTALL_PATH}/config/api.yaml* 配置默认的 API 设置。

2.4.1 项目与用户组

同 *章节 2.3.1 项目与用户组*

2.4.2 定制化接口配置

PRE_CFG

IFP 在启动时将自动加载当前路径下的 ifp.cfg.yaml，其中包含了用户所处项目、用户组、定义的 IFP 变量、负责的模块和加载的默认设置等。而该 API 适用于在 IFP 加载配置前自动判断以上信息，从而快速产生 ifp.cfg.yaml，用户无需任何设置即可开始日常工作。由于执行该 API 时还不清楚用户所处项目和用户组，所以该 API 一般配置于 api.yaml 中。例如：

配置方式：

```
API:
    PRE_CFG:
        - LABEL: 'pre_cfg'
          PATH: ${CWD}
          ENABLE：True
          COMMAND:  python3 ${IFP_INSTALL_PATH}/tools/ifp_pre_cfg.py
          COMMENT：
```

属性包括：

LABEL ： 唯一标签（必须）

PATH ： 命令执行路径（必须）

ENABLE ： 是否启动该 API（可选，默认为 True）

COMMAND ： 执行的命令（必须）

COMMENT ： 注释（可选）

PRE_IFP

该 API 在执行完 PRE_CFG 和加载完 ifp.cfg.yaml 后，启动图形界面前触发。因为已经加载 ifp.cfg.yaml 中用户所在项目和用户组，所以该 API 一般定义在 api.${project}.${group}.yaml 中，触发一些图形界面启动前置任务。

配置方式：

```
API:
    PRE_IFP:
        - LABEL: "PRE_IFP function for demo case"
          PROJECT:
          GROUP:
```

PATH: ${CWD}

ENABLE: True

COMMAND: echo -e 'You can execute some script after load ifp.cfg.yaml and before launch GUI '

COMMENT : 'For demo case'

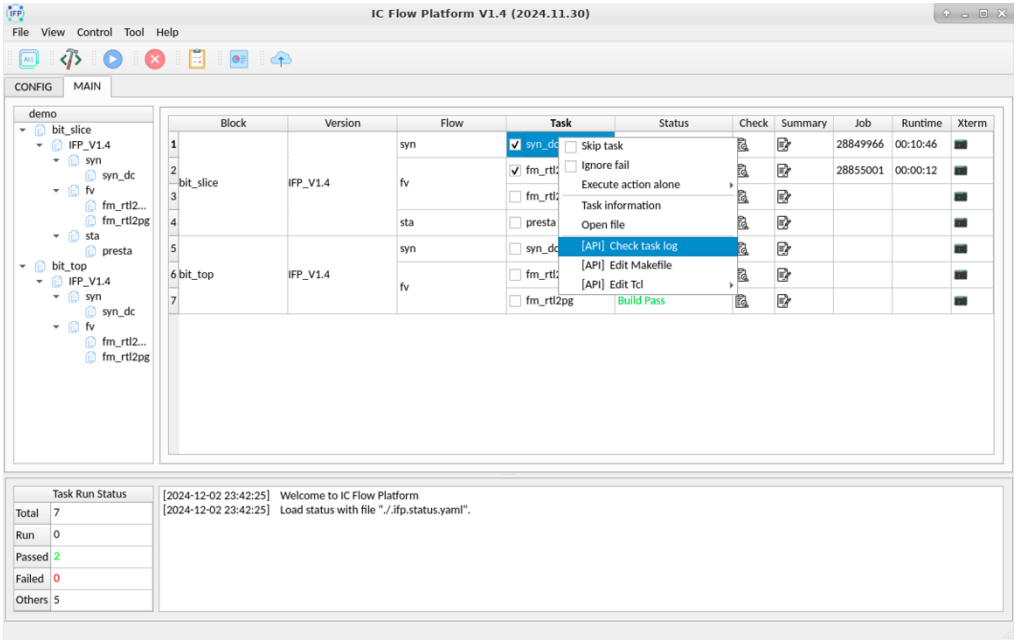
除了 PRE_CFG 中提到的属性以外，PRE_IFP 具有一些特有的属性：

PROJECT ： 生效的项目（可选）

GROUP ： 生效的用户组（可选）

TABLE_RIGHT_KEY_MENU

该 API 为 *CONFIG-Task 界面* 和 *MAIN 界面* 表格右键菜单绑定的功能，一般用于汇总与 Block / Version / Flow / Task 相关的日常辅助功能。例如：



配置方式：

API:

TABLE_RIGHT_KEY_MENU:

- LABEL: "Check task log "

PROJECT:

GROUP:

TAB: MAIN

COLUMN: TASK

PATH: ${CWD}

ENABLE: True

BLOCK_NAME:

VERSION_NAME:

FLOW_NAME:

TASK_NAME:

COMMAND: gvim ${CWD}/${BLOCK}/${VERSION}/dc/logs/log.dc.bit_slice

COMMENT : 'Review log'

API-2:

除了 PRE_CFG 和 PRE_IFP 中提到的属性以外，*TABLE_RIGHT_KEY_MENU* 具有一些特有的属性：

TAB ： 生效的界面，仅限 CONFIG/MAIN（可选）

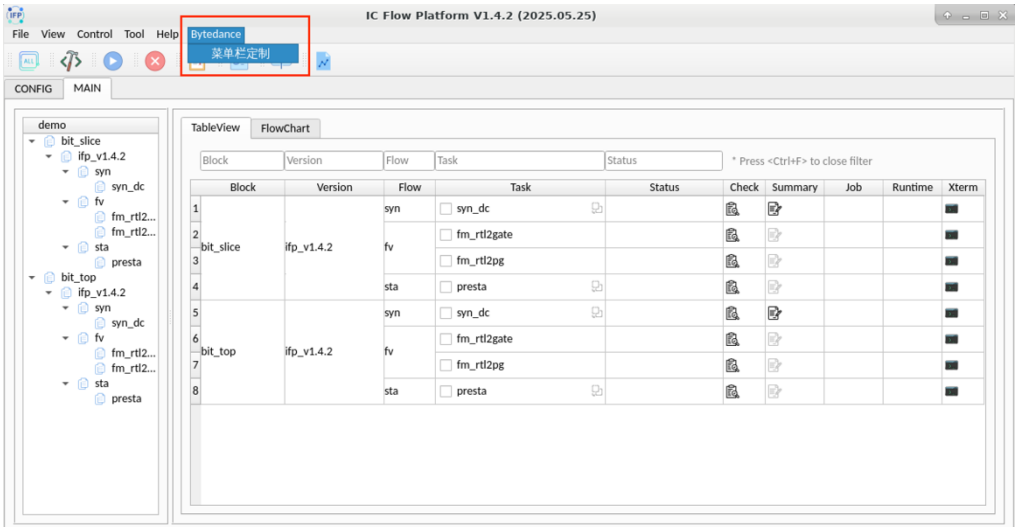
COLUMN ： 生效的列，仅限 BLOCK/VERSION/FLOW/ TASK （可选）

BLOCK_NAME ： 生效的 Block 名字，其他 **_NAME* 功能类似（可选）

API-2 ： 二级菜单，属性与一级菜单相同，但此时一级菜单的相关命令被禁用（可选）

MENU_BAR

该 API 为菜单栏绑定的功能，支持用户定制个性化菜单栏操作界面，可集成常用功能入口，提升操作效率与使用体验。例如：



配置方式：

API:

MENU_BAR:

- LABEL: "菜单栏定制"

MENU_BAR_GROUP: 'Bytedance'

PATH: ${CWD}

COMMAND: firefox https://localhost:8000

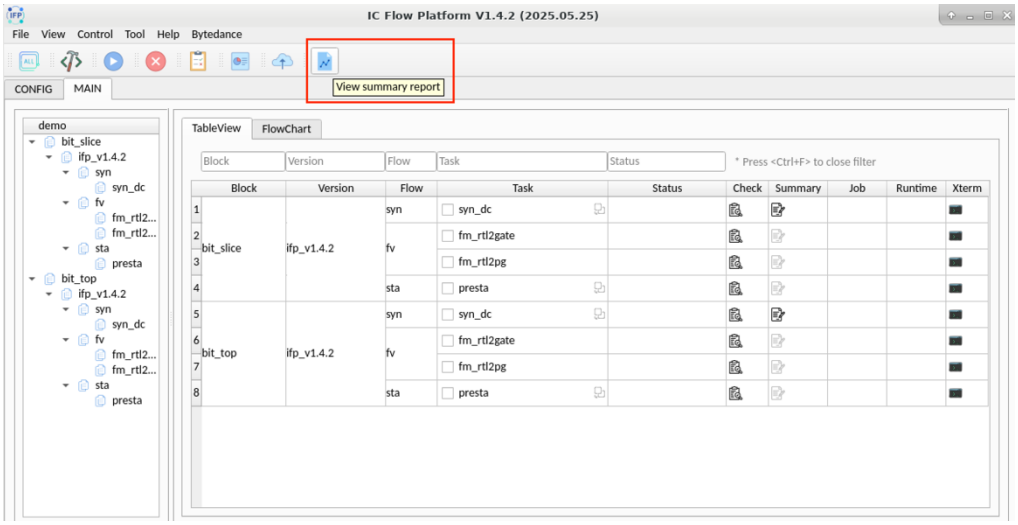
除了上述中提到的属性以外，MENU_BAR 具有一些特有的属性：

MENU_BAR_GROUP ： 菜单栏分组

TOOL_BAR

该 API 为工具栏绑定的功能，提供常用功能的快捷操作按钮区域，支持用户快速执行常见任务，如查看报告、导出等操

作。



配置方式：

API:

TOOL_BAR:

- LABEL: "View summary report"

ENABLE: True

PATH: ${CWD}

COMMAND: firefox backup/html/index.html

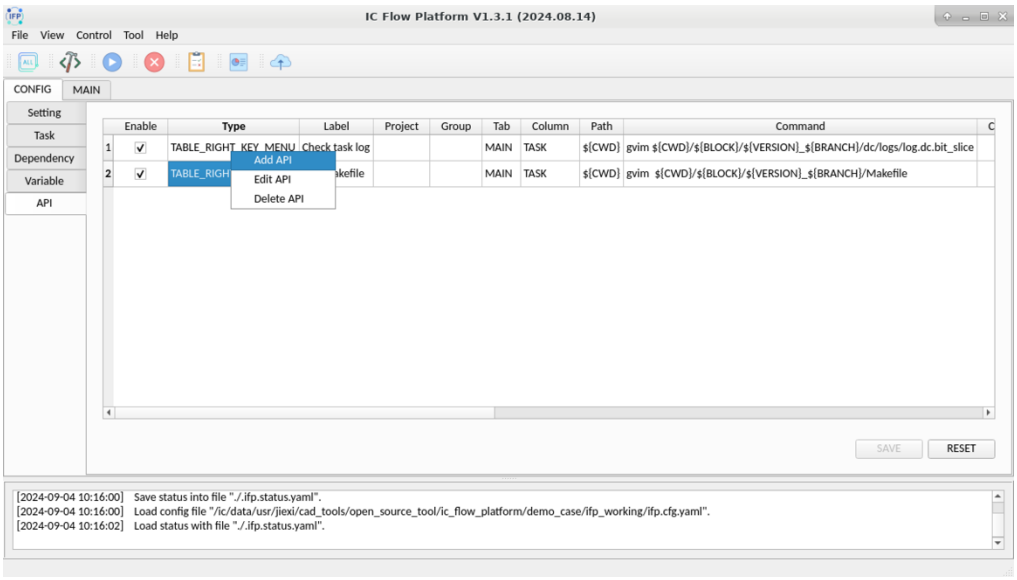
ICON: ${IFP_INSTALL_PATH}/data/pictures/office/report.png

除了上述中提到的属性以外，TOOL_BAR 具有一些特有的属性：

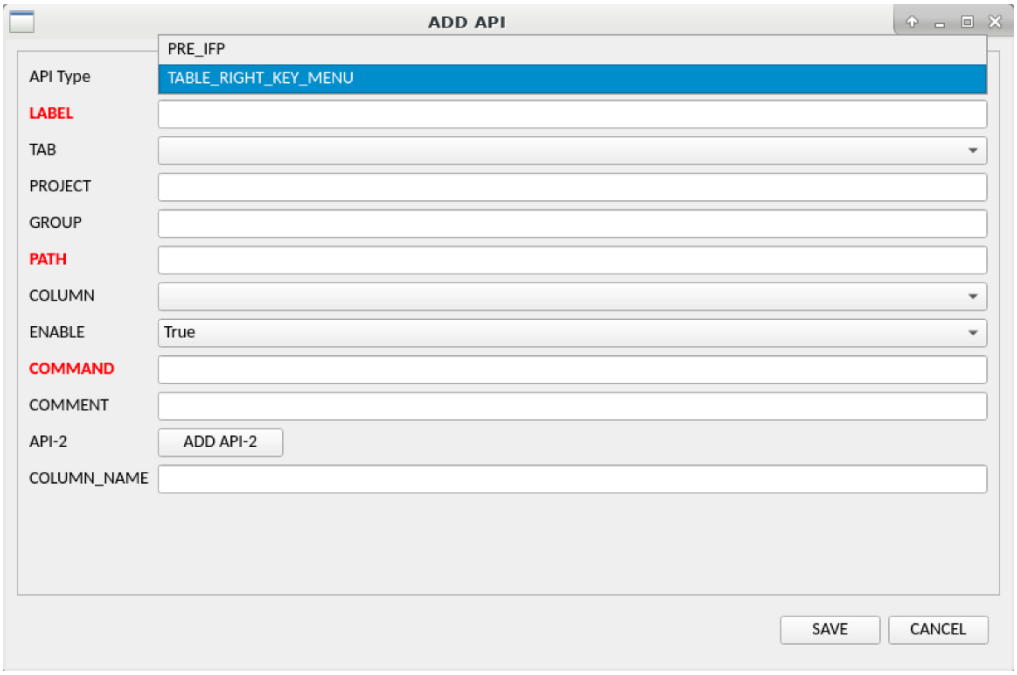
ICON ： 工具栏图标

2.4.3 通过界面配置

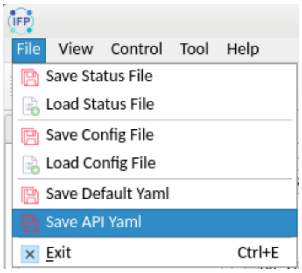
CONFIG-API 界面主要用于管理系统中与 API 相关的配置，管理员可以通过图形化界面对 API 的各项参数进行配置。通过右键功能完成新增、编辑和删除 API 设置，能够更高效地管理定制化接口。



如下为编辑界面，各项内容填写请参考 *章节 2.4.2 定制化接口配置* 内容。



配置完成后，管理员可在菜单栏 *File -> Save API Yaml* 将当前配置导出为 api.yaml。



2.5 Wrapper 配置

通过写 wrapper 的方式自动配置 IFP 的用户输入文件 ifp.cfg.yam。请参考*${IFP_INSTALL_PATH}/tools/ifp_demo_wrapper.py*

ifp.cfg.yaml 是 IFP 的配置文件，用于集中管理和加载所有与 IFP 相关的配置信息，包含了模块名称、版本信息、命令执行、流程调度，以及环境设置和用户自定义参数等，能够帮助用户快速完成 IFP 的初始化配置，具体优势如下：

（1）用户无需手动填写繁琐的配置项；

（2）保证配置格式统一、内容完整；

（3）支持根据模块/流程自动适配参数；

（4）便于系统后续扩展和维护。

生成配置文件 ： *${IFP_INSTALL_PATH}/tools/ifp_demo_wrapper.py -b bit_slice -v v1.4.2*

文件内容如下：

```
VAR:
  BSUB_QUEUE: normal
  MAX_RUNNING_JOBS: '6'
BLOCK:
  bit_slice:
    v1.4.2:
      syn:
        syn_dc: {}
      fv:
        fm_rtl2gate: {}
        fm_rtl2pg: {}
      sta:
        presta: {}
GROUP: syn
PROJECT: demo
API_YAML: ${IFP_INSTALL_PATH}/config/api.demo.syn.yaml
DEFAULT_YAML:   ${IFP_INSTALL_PATH}/config/default.demo.syn.yaml
```

3. 辅助工具

3.1 lsfMonitor

安装路径位于 *${IFP_INSTALL_PATH}/tools/lsfMonitor*。

使用说明可参考 https://github.com/liyanqing1987/lsfMonitor

3.2 自动化 CHECKLIST 检查机制配置

3.2.1 检查机制介绍

IFP 内嵌了一套自动化 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 进行 check 检查时可调用这些 checklist 脚本自动做任务结果检查。

该 checklist 机制旨在简化用户的 checklist 检查操作，减少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。

如下是针对 synthesis 流程整理的 Excel 格式 checklist 配置（该配置文件均可按照项目需求修改）。

TASK	DESCRIPTION	FILE	MESSAGE	WAIVE_MESSAGE	TYPE
initopt	Check "Error" message on log file.	run.log	^Error No such element in array	user define	check_error_message
	Check "Warning" message on log file.	run.log	^Warning	[MESG-11] Unusable isolation cell	check_warning_message
	Get genus version.	genus.cmd	Cadence Genus(TM) Synthesis Solution, Version		check_warning_message
	Make sure genus exit normally.	run.log	^Normal exit		check_expected_message
	Make sure final output files exist.	<BLOCK>.v <BLOCK>.sdc.gz			check_file_exist
	Review VT cell ratio.	<BLOCK>.<TASK>.vt_percentage.rpt			review_file
advopt	Check "Error" message on log file.	run.log	^Error		check_error_message
	Review <BLOCK>.final_qor.rpt	<BLOCK>.final_qor.rpt			review_file
	Check No empty modules in design.	<BLOCK>.check_design.rpt	No empty modules		check_expected_message
finalopt	Find uniquify naming rule during finalopt.	run.log	update names		check_expected_message
	Review <BLOCK>.scandef.rpt	<BLOCK>.scandef.rpt			review_file
	Check "Error" message on <BLOCK>.check_load_constraints.chk	<BLOCK>.check_load_constraints.chk	^Error		check_error_message

Excel 包含 TASK / DESCRIPTION / FILE / MESSAGE / WAIVE_MESSAGE / TYPE 六项内容，说明如下。

NOTICE	
TASK	Specify task name. For example, 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".
WAIVE_MESSAGE	Specify message(s) you want to waive. 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

（1）**TASK：**指定流程步骤。比如 syn 的 TASK 有 initopt/finalopt，一个 TASK 可定义多行 checklist 项（将对应的 TASK

单元格合并即可)；如果 TASK 前面带有**“#”**字符，则表明该 TASK 行是注释行，暂不开放，脚本会忽略它。

（2）**DESCRIPTION：**checklist 项描述。说明此项 checklist 要检查的内容，务必描述清晰。

（3）**FILE：**checklist 检查文件。每个 checklist 项可同时检查多个文件，文件之间用**换行**隔开。

（4）**MESSAGE：**checklist 检查信息。从 FILE 指定文件中检索指定的信息，支持同时检索多条信息，多条 MESSAGE 之间用**换行**隔开。

（4）**WAIVE_MESSAGE：**指定 waive 信息。从指定文件中检索到的信息进行 waive 操作，支持同时 waive 多条信息，多条 WAIVE MESSAGE 之间用**换行**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。

管理员需要注意的是：Excel 默认支持两个变量 *${BLOCK} / ${TASK}*，并支持执行 checklist 脚本时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。

3.2.2 CHECKLIST 配置方法

Demo Excel 位于 *${IFP_INSTALL_PATH}/action/check/demo_excel/syn_genus_checklist.xls。*

转换脚本路径位于 *${IFP_INSTALL_PATH}/action/check/scripts/gen_checklist_scripts.py。*

报告查看脚本位于 *${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report.py。*

配置 checklist 检查项

配置 checklist 项时，必须采用英文模式编辑，避免引入全角字符。如下 syn_dc_checklist.xls 为例。

TASK	DESCRIPTION	FILE	MESSAGE	WAIVE_MESSAGE	TYPE
syn_dc	Check “Error” message on log file.	log/syn_dc.log	^Error ^ERROR		check_error_message
	Check “Warning” message on log file.	log/syn_dcg.log	^Warning	(LINT-1) Inconsistent Data for Layer 17	check_warning_message
	Get RTL version.	log/syn_dcg.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

checklist 脚本转换

调用 switch 脚本*$<IFP_INSTALL_PATH>/action/check/scripts/gen_checklist_scripts.py* 将 checklist Excel 转换为 checklist script。其中 gen_checklist_scripts.py 脚本用法如下:

-i INPUT, --input INPUT 指定输入 Excel 文件（必须）。

-f FLOW, --flow FLOW 指定流程名称（必须）。

-v VENDOR, --vendor VENDOR 指定 vendor 名称（必须），用于标识 EDA 工具所属的 vendor 。

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

示例如下：

python3 <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" ...

/data/usr/cad_tools/flows/ic_flow_platform/action/check/syn/synopsys/syn_synopsys.syn_dc.py

配置 checklist 脚本

syn_synopsys.syn_dc.py 使用方法如下：

-b BLOCK, --block BLOCK 指定 block 名称（必须）。

-t TASK, --task TASK 指定 task 名称（必须）。

执行检查脚本会在目录下生成所有 checklist 项的检查日志和检查报告。管理员可将 checklist script 以及 view report 配置到 default.yaml，配置方式详见 *章节 2.2.3 属性*。

```
CHECK:
    PATH: ${DEFAULT_PATH}/syn_dc
    COMMAND: python3 ${IFP_INSTALL_PATH}/action/check/syn/synopsys/syn_synopsys.syn_dc.py -b ${BLOCK} -t ${TASK}
    VIEWER: python3 ${IFP_INSTALL_PATH}/action/check/scripts/view_checklist_report.py
    REPORT_FILE: ${DEFAULT_PATH}/syn_dc/file_check/file_check.rpt
```

附录

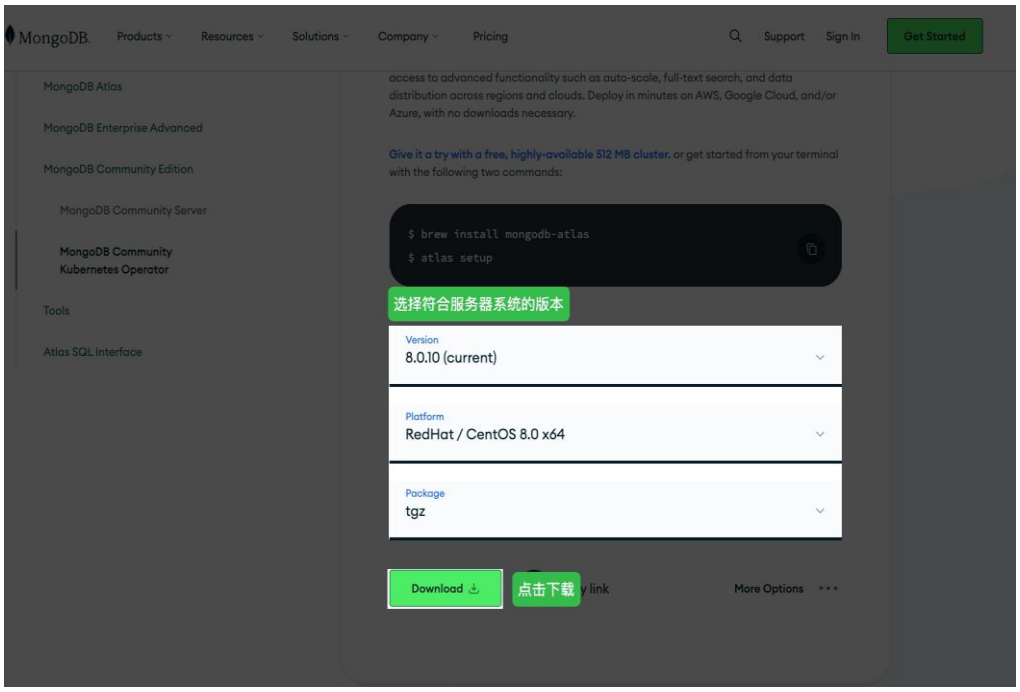
附录一 MongoDB 离线部署方式

1. 下载 MongoDB

首先需要选择一台服务器，用来部署 MongoDB 数据库。

根据数据库服务器的系统，在 MongoDB 官网下载对应系统版本的 MongoDB tgz 安装包。

网址为：https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel8-8.0.10.tgz



2. 解压安装包

（1）将 MongoDB 的安装包复制到安装路径：

mkdir <mongodb 安装路径>

cp <mongodb-linux-> <mongodb 安装路径>/*

cd <mongodb 安装路径>

（2）解压安装包，得到 mongodb-linux-*路径

tar -zxvf <mongo-linux->*

3. 启动 MongoDB 数据库

3.1 加载环境变量

需要将 MongoDB 加入到环境变量中：

export PATH=<mongodb 安装路径>/<mongodb-linux->/bin:$PATH*

export LD_LIBRARY_PATH=<mongodb 安装路径>/<mongodb-linux->/lib:$LD_LIBRARY_PATH*

3.2 创建数据库储存路径

创建一个空目录，作为 MongoDB 的数据保存路径

mkdir <mongodb 数据保存路径>

3.3 启动数据库

使用如下命令启动数据库:

*mongod *

*--bind_ip 0.0.0.0 *

*--port <当前机器的空闲端口> *

*--dbpath <mongdb 数据保存路径> *

--logpath <mongodb 日志存放位置>

4. 在 IFP 中配置 MongoDB

（1）当 MongoDB 成功启动时，可以在 IFP 中配置 MongoDB，用来存放 IFP 的全部用户使用 IFP 的记录

（2）配置方式，打开 IFP 安装路径下的 config/config.py 文件，在最后添加:

mongo_db = 'mongodb://<部署 mongodb 的机器>:<mongodb 使用的端口>/'

5. 将 MongoDB 部署为服务

在 Terminal 启动的 MongoDB 服务会随着 Terminal 的关闭而关闭。

为了避免这种情况的发生，可以把 MongoDB 部署为系统服务。让 MongoDB 随着服务器开机启动而启动。

5.1 编辑 env 文件

准备一个 mongodb.env 的文本文件，内容如下:

PATH=<mongodb 安装路径>/<mongodb-linux->/bin*

LD_LIBRARY_PATH=<mongodb 安装路径>/<mongodb-linux->/lib*

（1）编辑 service 文件：准备一个 mongodb.service 文件

service 文件模版

[Unit]

Description=SGlang DeepSeek Service

After=network-online.target

[Service]

ExecStart=/bin/bash -c '<启动命令>' # 启动命令填写在启动 mongodb 的命令， 参考 3.3

User=root

Group=root

Restart=always

RestartSec=3

EnvironmentFile=<env 文件> # env 文件填写编辑好的 mongo.env 文件的路径

StandardOutput=append:<log 文件> # log 文件填写 log 的保存路径

StandardError=inherit

[Install]

WantedBy=default.target

service 文件示例：（如下为真实使用的文件）

```
[Unit]
Description=MongoDB Service
After=network-online.target

[Service]
ExecStart=/bin/bash -c 'mongod --bind_ip 0.0.0.0 --port 27017 --dbpath
/ic/data/CAD/flows/user/zhangjingwen.silvia/mongodb/data --logpath /ic/data/CAD/flows/user/zhangjingwen.silvia/mongodb/logs'
User=zhangjingwen.silvia
Group=ic_work_group
Restart=always
EnvironmentFile=/ic/data/CAD/flows/user/zhangjingwen.silvia/mongodb/mongodb.env
RestartSec=3
StandardError=inherit

[Install]
WantedBy=default.target
```

（2）启动 service 服务

a. 移动 service 文件：把 service 文件放到/usr/lib/systemd/system/路径下

```
cp -rf mongodb.service /usr/lib/systemd/system/
```

b. reload service 文件

```
systemctl daemon-reload
```

c. enable 服务：enable service，使该服务可以随着系统启动而自动启动

```
systemctl enable mongodb.service
```

d. 启动服务

```
systemctl start mongodb.service
```