BQS Script Usage Model

**Jason Wang**

ABSTRACT

BQS scripts are series of scripts developed by Script group according with the requirements from the whole QA group. BQS scripts are the core implementation scripts for BQS platform.

In this document we will introduce the usage of core script and the some application tools coming with core script, such as: suite support, behavior check.

REVISION HISTORY

|  |  |  |
| --- | --- | --- |
| REVISION | RELEASE DATE | COMMENTS |
| V1.0 | 05/05/2014 | Initial draft – Jason Wang |
| V1.1 | 07/01/2014 | Delete default implement par trce option –Jason Wang |
| V1.2 | 08/15/2014 | Some APP tools update & type error fix –Jason Wang |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

CONTENTS

[1 Preface 6](#_Toc395885220)

[1.1 What are BQS scripts 6](#_Toc395885221)

[1.2 how to get BQS scripts 6](#_Toc395885222)

[1.2.1 BQS scripts on the web 6](#_Toc395885223)

[1.2.2 BQS scripts export 6](#_Toc395885224)

[1.3 BQS scripts history 6](#_Toc395885225)

[1.4 BQS scripts developer team 6](#_Toc395885226)

[2 Fundamental Concepts 8](#_Toc395885227)

[2.1 Scripts directory structure 8](#_Toc395885228)

[2.2 Lattice main script structure 8](#_Toc395885229)

[2.2.1 Main script total structure 8](#_Toc395885230)

[2.2.2 Design entry structure 9](#_Toc395885231)

[2.2.3 Implementation structure 10](#_Toc395885232)

[2.2.4 Simulation structure 12](#_Toc395885233)

[2.2.5 Check & report structure 13](#_Toc395885234)

[3 Basic usage 14](#_Toc395885235)

[3.1 Environment setting 14](#_Toc395885236)

[3.2 Script initialization 14](#_Toc395885237)

[3.3 Test case building 15](#_Toc395885238)

[3.4 Test suite building 15](#_Toc395885239)

[3.5 Typical Lattice main options 15](#_Toc395885240)

[3.5.1 Public options 16](#_Toc395885241)

[3.5.2 Diamond options 17](#_Toc395885242)

[3.5.3 Project options 18](#_Toc395885243)

[3.5.4 Implementation options 19](#_Toc395885244)

[3.5.5 Simulation options 19](#_Toc395885245)

[3.6 Lattice main options APP & Restrictions 20](#_Toc395885246)

[3.6.1 Public options 20](#_Toc395885247)

[3.6.2 Project options 21](#_Toc395885248)

[3.6.3 Implementation options 21](#_Toc395885249)

[3.6.4 Simulation options 22](#_Toc395885250)

[3.7 Launch file writing 22](#_Toc395885251)

[3.7.1 CMD line structure 22](#_Toc395885252)

[3.7.2 Demo CMD line writing 24](#_Toc395885253)

[3.8 Launch test suite 25](#_Toc395885254)

[3.9 Result check 25](#_Toc395885255)

[3.9.1 Where is the implementation result 25](#_Toc395885256)

[3.9.2 Some debug files 25](#_Toc395885257)

[3.10 Report check 26](#_Toc395885258)

[3.10.1 Where is test report 26](#_Toc395885259)

[3.10.2 Report result definition 26](#_Toc395885260)

[4 Test flow orientation 27](#_Toc395885261)

[4.1 General flow 27](#_Toc395885262)

[4.1.1 Object 27](#_Toc395885263)

[4.1.2 Quick start 27](#_Toc395885264)

[4.1.3 Major options 27](#_Toc395885265)

[4.1.4 Work flow 28](#_Toc395885266)

[4.1.5 General flow usage 29](#_Toc395885267)

[4.1.6 Examples 30](#_Toc395885268)

[4.2 Sweeping flow 30](#_Toc395885269)

[4.2.1 Object 30](#_Toc395885270)

[4.2.2 Quick start 30](#_Toc395885271)

[4.2.3 Major options 31](#_Toc395885272)

[4.2.4 Work flow 32](#_Toc395885273)

[4.2.5 Sweeping flow usage 34](#_Toc395885274)

[4.2.6 Examples 36](#_Toc395885275)

[4.3 Crossover flow 36](#_Toc395885276)

[4.3.1 Object 36](#_Toc395885277)

[4.3.2 Quick start 36](#_Toc395885278)

[4.3.3 Major options 37](#_Toc395885279)

[4.3.4 Work flow 38](#_Toc395885280)

[4.3.5 Crossover flow usage 38](#_Toc395885281)

[4.3.6 Examples 39](#_Toc395885282)

[4.4 Simulation flow 39](#_Toc395885283)

[4.4.1 Object 39](#_Toc395885284)

[4.4.2 Quick start 39](#_Toc395885285)

[4.4.3 Major options 40](#_Toc395885286)

[4.4.4 Work flow 40](#_Toc395885287)

[4.4.5 Simulation flow usage 41](#_Toc395885288)

[4.4.6 Examples 44](#_Toc395885289)

[4.5 Command flow 46](#_Toc395885290)

[4.5.1 Object 46](#_Toc395885291)

[4.5.2 Quick start 46](#_Toc395885292)

[4.5.3 Major options 46](#_Toc395885293)

[4.5.4 Work flow 47](#_Toc395885294)

[4.5.5 CMD flow usage 50](#_Toc395885295)

[4.5.6 Examples 50](#_Toc395885296)

[5 Application tool usages 52](#_Toc395885297)

[5.1 suite support usage 52](#_Toc395885298)

[5.1.1 Object 52](#_Toc395885299)

[5.1.2 Quick start 52](#_Toc395885300)

[5.1.3 Detail usage 52](#_Toc395885301)

[5.2 Behavior check usage 55](#_Toc395885302)

[5.2.1 Object 55](#_Toc395885303)

[5.2.2 Quick start 55](#_Toc395885304)

[5.2.3 Detail Usage 55](#_Toc395885305)

[5.3 Scan report usage 61](#_Toc395885306)

[5.3.1 Object 61](#_Toc395885307)

[5.3.2 Quick Start 61](#_Toc395885308)

[5.3.3 Detail Usage 61](#_Toc395885309)

[5.4 Multi line usage 61](#_Toc395885310)

[6 Reference 63](#_Toc395885311)

[7 Demo case & suite 64](#_Toc395885312)

[7.1 Test case demo 64](#_Toc395885313)

[7.2 Test suite demo 64](#_Toc395885314)

[8 Appendices 65](#_Toc395885315)

[8.1 Lattice main script options 65](#_Toc395885316)

[8.2 Lattice main script flow chart 67](#_Toc395885317)

# 1 Preface

## 1.1 What are BQS scripts

BQS scripts are series of scripts including main scripts and application tools. BQS scripts support kinds of test flows to target all test requirements for the whole QA group with the standard test case or test suite format.

Currently BQS scripts support the following test flows:

1. [General (pushbutton) flow](#_4.1_General_flow)
2. [Sweeping flow](#_4.2_Sweeping_flow)
3. [Crossover flow](#_4.3_Crossover_flow)
4. [Simulation flow](#_4.4_Simulation_flow)
5. [Command flow](#_4.5_Command_flow)

BQS scripts also support some check, scan and reporting flow with the help of application tools. Currently the main scripts only support simply results check flow. For detail check, scan and reporting flow please use application tools.

## 1.2 how to get BQS scripts

Currently BQS scripts were controlled by SVN. You need to have an account to view and download the scripts. For account apply please contact Script group.

### 1.2.1 BQS scripts on the web

Type the following address in your web explorer:

<http://d25315/viewvc/auto/bqs_scripts/>

You will see the following folders:

1. Release/ --current stabile version
2. Tag/ --history stabile version
3. Trunk/ --the latest develop version

### 1.2.2 BQS scripts export

To export BQS scripts we need to install SVN client first in your test machine. To check if your machine have installed SVN client just type “SVN help” in the command line console, if the console export the help information, it means the SVN client was installed. Contact your machine admin for SVN install if there is no SVN client on your machine.

Typing the following command line to export BQS scripts:

***“svn export http://d25315/auto/bqs\_scripts/release”***

It is recommending to use the following command line to update the device support of the BQS scripts:

***“python release/tools/generate\_diamond\_xml/generate\_xml.py <diamond install path> <diamond bit> trunk/conf”***

Available value for diamond bit will be: nt, nt64, lin, lin64

## 1.3 BQS scripts history

The original scripts come from benchmark test flow. The initial version was developed by Steven Song. After years updating by former benchmark group, the scripts contain more functions than before. It can do much more types of test flow, such as General flow, command flow and more sweeping flow. It also has many tool scripts to scan, check and report the results.

## 1.4 BQS scripts developer team

BQS scripts are a huge project, there are many engineers join this project to definition, coding, testing. Here is a list of main participators:

1. Main participators

|  |  |  |
| --- | --- | --- |
| **Section** | **Name** | **Sub section** |
| definition | Ping Chen <Ping.Chen@latticesemi.com> |  |
|  | Shawn Yan <Shawn.Yan@latticesemi.com> |  |
|  | Jason Wang <Jason.Wang@latticesemi.com> |  |
|  | Yibin Sun <Sun.Yibin@latticesemi.com> |  |
| coding | Shawn Yan <Shawn.Yan@latticesemi.com> | Lattice main script |
|  | Yu Zhao <Yu.Zhao@latticesemi.com> | application tools |
| testing | Bohong Chen dimission | Check flow |
|  | Jeffrey Ye <Jeffrey.Ye@latticesemi.com> | Sweeping flow |
|  | Strdom Fang <Strdom.Fang@latticesemi.com> | General flow |
|  | Peter Zhou <Peter.Zhou@latticesemi.com> | Simulation flow |
|  | Jason Wang <Jason.Wang@latticesemi.com> | Simulation flow |

# 2 Fundamental Concepts

## 2.1 Scripts directory structure

BQS scripts are the collection of main scripts and application tools. The structure of BQS scripts will like the following:

./BQS\_scripts --top folder name (sometimes use trunk)

./bin --main scripts folder

Run\_lattice.py --main script for Lattice

./xlib --public library for all vendor

… --will add more main scripts in the future

./conf --configure information located here

./sim --default setting for simulation

./synthesis --default synthesis setting

Default.ini --default configure for main script

DiamondDevFile.xml --default Lattice device list

./tools --application tools located here

./autoLSE --tools for LSE run

./scan\_report --tools for report scan

… --some other tools here

./docs

./ BQS\_script\_usage\_model.docx --documents for scripts usage

…

CHANGES --revision change log file

## 2.2 Lattice main script structure

### 2.2.1 Main script total structure

Lattice main script is test case oriented. Script will run a design case step by step: get design entry, run implement flow, run simulation flow and finally run check and report flow.

Here is a top level flow chart for script work flow.



1. Script work flow

### 2.2.2 Design entry structure

Lattice main script will scan the test case to find the design entry. With the different information in test case, script will pass down into different implementation flow. Here are the scan rules for making a decision.

1. Scan case folder to search a file with name like xx.info ([qa]:cmd\_file=xx can be used as item 2);
2. Try to search file named like run + + (".sh", ".csh", ".bat", ".cmd", ".py", ".pl"), if found, run cmd\_file flow (will execute check flow ONLY –check-conf xx is specified in command line);
3. Scan xx.info file for LDF file information.
4. Scan xx.info file for key word “cmd\_flow”

Here is a flow chart for script running in this process:



1. Entry work flow

\*NOTE:

1. TCL work flow:

Test case have a project file (which also can be rebuild from RTL/EDF files), script will run like Diamond GUI flow (open project, set project, run implement, run simulation…).

1. CMD work flow:

Test case doesn’t have a project file, script will run in command line flow (use input files to run foundry commands defined in “cmd\_flow” section, such as: map, par).

### 2.2.3 Implementation structure

Implement flow will run the test case according the flow steps defined by info file or external options. Since we have two different design entries, so implement will be a little difference from each other.

#### 2.2.3.1 Implement flow for TCL flow

TCL flow will support the following option:

1. Device-kit: user can change project device info
2. No-scuba: user can disable update scuba command line in project files
3. Strategy update: user can update strategy setting
4. Run process: user can specific a run process (run map, run par, run par trce … )
5. Sweeping: user can run sweeping flow
6. Multi-seed: user can run multi-seed flow

\*NOTE: for total support options see [chapter 8.1](#_8.1_Lattice_main)

Here is a flow chart for TCL flow:



1. TCL work flow

\*NOTE: Implement prepare: will be creating work path, design information collection.

#### 2.2.3.2 Implement flow for CMD flow

CMD flow is for standalone command line support, with this flow support, user can run PAR command line with MAP NCD file and PRF file. CMD flow will support the following options and flow.

1. Run process: user can specific a run process (run map, run par, run par trce … )
2. Run process(engine) with specific command line
3. Sweeping: user can run sweeping flow(not support now)
4. Multi-seed: user can run multi-seed flow(not support now)

Here is a flow chart for CMD flow.



1. CMD work flow

\*Currently we don’t support sweeping flow and multi-seed flow.

\*Sanity check uses the following rules:

1) User should list all input files for the run command(s). For example: When a user want to run MAP command, the device information and NGD file were required in the info file, if there are one more flow specified by user, such as: PAR, one more LPF file should be list in the info file.

2) The default command line will be used while there is no specific command line find in info file.

3) User should list all run flows step by step, i.e. user input files are map ncd and prf files and user want to run par and par trace, user should list the following lines in “[cmd\_flow]”:

a) run\_par = 1

b) run\_trce = 1

\*for actually options please refer to command flow ([chapter 4.5](#_4.5_Command_flow)).

### 2.2.4 Simulation structure

Simulation flow is for those test cases need to run simulation. We can find the simulation information from the info file.

Currently simulation flow support simulation for all of the following styles with both ModelSIM & Active-HDL.

1. RTL simulation flow
2. POST map simulation flow
3. POST par simulation flow

For every of these three flows, BQS script support:

1. Source file simulation flow
2. Macro(.do) file simulation flow

BQS scripts will treat these options as add on option, this means script will run what the script get form arguments:

1. Get run RTL simulation: RTL simulation will be run
2. Get run RTL & MAP simulation: RTL & MAP simulation will be run
3. Get run RTL & PAR simulation: RTL & PAR simulation will be run
4. Get run RTL, MAP & PAR simulation: RTL, MAP & PAR simulation will be run.

Here is a flow chart for simulation flow:



1. Simulation flow

### 2.2.5 Check & report structure

Check report flow is used to check the test case implement status and report the implement info. We use “xx.conf” file to specify the check and report information for script use. And here is top level flow chart:



1. Check & report flow

# 3 Basic usage

## 3.1 Environment setting

To make sure the test suite run correctly we should check the following information:

1. Check the following SW installed on your machine:
2. SW list

|  |  |  |  |
| --- | --- | --- | --- |
| NO. | SW Name | Version | Comments |
| 1 | Python | V2.7 or higher |  |
| 2 | Diamond | NA | Install version depends on your test. |
| 3 | ModelSIM | 6.6d or higher | Ignore if you don’t run simulation with ModelSIM |
| 4 | QuestaSIM | 10.2 or higher | Ignore if you don’t run simulation with QuestaSIM |

1. Check the following license setting on your machine:
2. License setting

|  |  |  |  |
| --- | --- | --- | --- |
| NO. | SW Name | License | Comments |
| 1 | Python | NA | No license need |
| 2 | Diamond | 1700@linux2 ; 1700@d27 |  |
| 3 | Active-HDL | 27000@lsh-prince | Ignore if you don’t run simulation with Active-HDL |
| 4 | ModelSIM | 1717@ldc-wanlic01 | Ignore if you don’t run simulation with ModelSIM |
| 5 | QuestaSIM | 1717@ldc-wanlic01 | Ignore if you don’t run simulation with QuestaSIM |

## 3.2 Script initialization

Before launch script we should change or confirm some script setting in default.ini file (see [chapter 2.1](#_2.1_Scripts_directory) for file location):

**Function**: these values in this file will be the default value for script run. So if you finish these default values setting in this file you may ignore relevant options in command line to shorter the command line.

**QA section**:

1. default.ini qa section value

|  |  |  |  |
| --- | --- | --- | --- |
| NO. | Option name | Default value | Comments |
| 1 | diamond | c:/lscc/diamond/3.2\_x64 | Diamond install path for windows |
| 2 | diamond\_lin | NA | Diamond install path for Linux |
| 3 | diamond\_sol | NA | Diamond install path for Solaris |
| 4 | diamond\_fe\* | NA | Only for Crossover flow use |
| 5 | diamond\_be\* | NA | Only for Crossover flow use |
| 6 | modelsim\_path | C:\modeltech\_6.6d\win32 | ModelSIM install path |
| 7 | questasim\_path | NA | QuestaSIM install path |

\*Note: diamond\_fe, diamond\_be options were commented by semicolon by default, please only add these two options when crossover flow need.

**Environment section:**

1. default.ini environment section value

|  |  |  |  |
| --- | --- | --- | --- |
| NO. | Option name | Default value | Comments |
| 1 | LM\_LICENSE\_FILE | 27000@lsh-prince | Active-HDL license |
| 2 |  | 1717@ldc-wanlic01 | ModelSIM, QuestaSIM license |
| 3 |  | 1700@d27 | Diamond license |
| 4 |  | 1700@linux2 | Diamond license |
| 5 | TW\_PAP\_VERBOSE | 1 | Add PAP result in trace report |
| 6 | LSC\_RECORD\_CPUMEM | 1 | Record the peak memory usage |

## 3.3 Test case building

Since the script was developed based on some basic rules, you have to give the script enough information before getting the right results as you expected. So a specific case format should be followed to make the script works correctly.

Currently we support 2 different case styles.

1. BQS case format

BQS case format is the standard case format we defined for BQS scripts. The case structure for this kind of case like the following:

./case1

./models --folder to place your models files

./others --folder to place NGO NGD and some other files which will be include in your Diamond project

./par --folder to place project file

./sim --folder to place files for simulation

./source --folder to place source files

Bqs.info --information for script run

Bqs.conf --information for script check and report use

For detail case building refers to: BQS\_spec\_case\_build.docx in [chapter 6](#_6_Reference)

1. QAS case format

QAS case was defined for front end QAS system use. Initially our BQS scripts do not support this kind of case, as the requirement of test case reuse BQS scripts also support this kind of case in the later version now. The case style like the following:

./case2

\_qas.info --QAS key case information record file

Gold.txt --golden file

Case2.v --some source file

Case2\_tb.v --some test bench file

… --some other file

For detail case building refers to: Front end QAS case usage model.

## 3.4 Test suite building

On user perspective, they would like to put all test cases together to construct a test suite and test all his or her items suite by suite. BQS scripts also support suite test with the suite support application tool.

For test suite building refers to: BQS\_spec\_suite\_build.docx in [chapter 6](#_6_Reference)

For suite support application tool usage refers to: [chapter 5.1](#_5.1_suite_support)

## 3.5 Typical Lattice main options

In this section we will list some typical application options for Lattice main script including the default and available values, detail explanation for these options. For total options please refers to: [chapter 8.1](#_8.1_Lattice_main)

Every option will have one of the following option types:

1. Store true: if option present, the behavior for this option will be active.
2. Store False: if option present, the behavior for this option will not be active.
3. Store string: Set the value for the option.
4. Store choice: the value for this option must in some available list.

### 3.5.1 Public options

Public options are basic options for script run, including showing help and debug information, where to find test suite, where to run test suite and where to place test report.

#### 3.5.1.1 –help (print help info)

|  |  |
| --- | --- |
| Option name | --help |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

This option is used to print the help information for Lattice main script.

#### 3.5.1.2 –debug (print debug info)

|  |  |
| --- | --- |
| Option name | --debug |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

This option is used to print the debug information while script running.

#### 3.5.1.3 –dry-run (only generate project)

|  |  |
| --- | --- |
| Option name | --dry-run |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

This option is used to generate LDF project in the job path and skip the following flows.

#### 3.5.1.4 –file (user script configure file)

|  |  |
| --- | --- |
| Option name | --file, -F |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Specify user configuration file, the file format is same as default.ini ([chapter 3.2](#_3.2_Script_initialization)) file.

#### 3.5.1.5 –conf (user configure directory)

|  |  |
| --- | --- |
| Option name | --conf |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Specify user configuration directory to replace script default conf directory.

#### 3.5.1.6 –top-dir (top source path name)

|  |  |
| --- | --- |
| Option name | --top-dir |
| Option type | Store string |
| Option default value | Script current work directory |
| Option available value | NA |

Specify top source path name.

#### 3.5.1.7 –job-dir (top job path name)

|  |  |
| --- | --- |
| Option name | --job-dir |
| Option type | Store string |
| Option default value | The values for –top-dir |
| Option available value | Any valid directory on local machine |

Specify top job path name.

#### 3.5.1.8 –design (test design name)

|  |  |
| --- | --- |
| Option name | --design |
| Option type | Store string |
| Option default value | NA |
| Option available value | Any string |

Specify test design name.

#### 3.5.1.9 –tag (job tag name)

|  |  |
| --- | --- |
| Option name | --tag |
| Option type | Store string |
| Option default value | \_scratch |
| Option available value | Any string |

Specify top job path name.

#### 3.5.1.10 –check-rpt (report file specify)

|  |  |
| --- | --- |
| Option name | --check-rpt |
| Option type | Store string |
| Option default value | Check\_by\_flow.csv |
| Option available value | Any string |

Specify report file.

### 3.5.2 Diamond options

Diamond options were used to specify the diamond install path and version (32bit or 64bit), actually we have specify these information in default.ini ([chapter 3.2](#_3.2_Script_initialization)) already. The options here have a higher priority, so if we specify some values here, it will overwrite the values in default.ini file.

#### 3.5.2.1 –x86 (Diamond version)

|  |  |
| --- | --- |
| Option name | -X, --x86 |
| Option type | Store true |
| Option default value | If this option not present, the diamond will be considered as 64bit |
| Option available value | NA |

Specify Diamond version, if this option present, the diamond will be considered as 32ibt.

#### 3.5.2.2 –diamond (Diamond install path for Windows)

|  |  |
| --- | --- |
| Option name | --diamond |
| Option type | Store string |
| Option default value | Values in default.ini file |
| Option available value | Any valid Diamond install path for Windows platform |

Specify Diamond install path on Windows platform.

#### 3.5.2.3 –diamond\_lin (Diamond install path for Linux)

|  |  |
| --- | --- |
| Option name | --diamond-lin |
| Option type | Store string |
| Option default value | Values in default.ini file |
| Option available value | Any valid Diamond install path for Linux platform |

Specify Diamond install path on Linux platform.

### 3.5.3 Project options

Project options are for project setting before project run.

#### 3.5.3.1 –run-scuba (update model files)

|  |  |
| --- | --- |
| Option name | --run-scuba |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

If this option present, script will try to update the files in model folder if scuba command line can be find.

#### 3.5.3.2 –devkit (update device)

|  |  |
| --- | --- |
| Option name | --device |
| Option type | Store string |
| Option default value | NO |
| Option available value | Any available device supported by Diamond |

Update the project device before run implementation if this option present.

#### 3.5.3.3 --synthesis (synthesis tools choice)

|  |  |
| --- | --- |
| Option name | --synthesis |
| Option type | Store choice |
| Option default value | synplify |
| Option available value | synplify, lse, precision |

Update the project device before run implementation if this option present.

#### 3.5.3.4 –use-sdc (use SDC file for synthesis)

|  |  |
| --- | --- |
| Option name | --use-sdc |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Use SDC file for synthesis flow if this option present.

#### 3.5.3.5 –strategy (Diamond strategy choice)

|  |  |
| --- | --- |
| Option name | --strategy |
| Option type | Store choice |
| Option default value | Strategy1 |
| Option available value | area, timing, quick |

Diamond strategy choice, if the option not present the default diamond strategy will be “Strategy1”, if this option present the value must in “area”, “timing”, “quick”.

#### 3.5.3.6 –goal (synthesis optimization goal)

|  |  |
| --- | --- |
| Option name | --goal |
| Option type | Store choice |
| Option default value | NA |
| Option available value | area, timing, balanced |

Synthesis strategy choice, if this option present the value must in “area”, “timing”, “balanced”.

#### 3.5.3.7–--run-ignore-tcl (DO NOT run tcl files which are specified in info file)

|  |  |
| --- | --- |
| Option name | --run-ignore-tcl |
| Option type | Boolean |
| Option default value | None |

Tcl files are defined in info file, their section name is qa, keys are inner\_tcl\_file and outer\_tcl\_file.

If run\_ignore\_tcl is False:

1. About inner\_tcl\_file, the scripts will merge the tcl flow lines into a standard open/save/close project tcl file and execute it.
2. About outer\_tcl\_file, the scripts will run it directly with pnmainc command.
3. Currently the inner\_tcl\_file and outer\_tcl\_file only support Radiant flow.

Tcl file key-word arguments:

1. design\_path
2. radiant\_path
3. devkit
4. family

Example:

Info file:

*[qa]*

*rdf\_file = ./testdesign/par/test.rdf*

*inner\_tcl\_file = ./testdesign/models/sbx1.tcl*

*outer\_tcl\_file = ./testdesign/models/sbx2.tcl*

tcl file:

*sbp\_design new -name a -path {%(design\_path)s/testdesign/models/sbp\_case/sbp\_case.sbx} -device {%(devkit)s} -language Verilog*

*sbp\_design config\_ip -vlnv {latticesemi.com:module:adc:1.0.0} -meta\_loc {%(radiant\_path)s/ip/%(family)s/adc} -cfg {%(design\_path)s/testdesign/models/sbp\_case/lib/latticesemi.com/module/adc/1.0.0/adc.cfg}*

*sbp\_add\_component -vlnv {latticesemi.com:module:adc:1.0.0} -name {adc\_inst}*

*sbp\_add\_port -direction in pi*

*sbp\_add\_port -direction out po*

*sbp\_add\_port -direction inout pio*

*sbp\_add\_port -from 1 -to 1 -direction in pb*

*sbp\_add\_interface\_port -vlnv {amba.com:AMBA3:AHBLite:r1p0} -direction master {interPort}*

*sbp\_design drc*

*sbp\_design save*

*prj\_add\_source {%(design\_path)s/testdesign/models/sbp\_case/sbp\_case.sbx}*

*sbp\_design generate*

*sbp\_design close*

### 3.5.4 Implementation options

All implementation options are for project implementation flow use, including pushbutton options, sweeping options.

#### 3.5.4.1 –run-translate (run project translate flow)

|  |  |
| --- | --- |
| Option name | --run-translate |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run translate flow for the given test case

#### 3.5.4.2 –run-map (run project map flow)

|  |  |
| --- | --- |
| Option name | --run-map |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run map flow for the given test case

#### 3.5.4.3 –run-par (run project par flow)

|  |  |
| --- | --- |
| Option name | --run-par |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run par flow for the given test case

#### 3.5.4.4 –run-par-trce (run project par trce flow)

|  |  |
| --- | --- |
| Option name | --run-par-trce |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run par and trace flow for the given test case

#### 3.5.4.5 –fmax-sweep (run fmax sweeping flow)

|  |  |
| --- | --- |
| Option name | --fmax-sweep |
| Option type | Store string |
| Option default value | NA |
| Option available value | <from> <to> <step> |

Specify fmax sweeping range, e.g. 50 300 10, means search form 50MHz to 300MHz, step is 10MHz.

#### 3.5.4.6 –seed-sweep (run seed sweeping flow)

|  |  |
| --- | --- |
| Option name | --seed-sweep |
| Option type | Store string |
| Option default value | NA |
| Option available value | <from> <to> <step> |

Specify seed sweeping range, e.g. 1 10 1, means search form seed1 to seed10, step is 1.

### 3.5.5 Simulation options

All simulation options are for project simulation flow use, including simulation tools setting and simulation flow control.

#### 3.5.5.1 –modelsim-path (modelsim path)

|  |  |
| --- | --- |
| Option name | --modelsim-path |
| Option type | Store string |
| Option default value | Values in default.ini file |
| Option available value | Any available ModelSIM install path |

Specify the ModelSIM install path, this value will overwrite the value set in default.ini file.

#### 3.5.5.2 –run-modelsim (simulation with modelsim)

|  |  |
| --- | --- |
| Option name | --run-modelsim |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Use ModelSIM run simulation when this option present in command line. By default the simulation tools is Active-HDL.

#### 3.5.5.3 –sim-rtl (run RTL simulation)

|  |  |
| --- | --- |
| Option name | --sim-rtl |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run project source RTL simulation.

#### 3.5.5.4 –sim-map-vlg (run post MAP Verilog simulation)

|  |  |
| --- | --- |
| Option name | --sim-map-vlg |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run post MAP Verilog simulation.

#### 3.5.5.5 –sim-par-vlg (run post PAR Verilog simulation)

|  |  |
| --- | --- |
| Option name | --sim-par-vlg |
| Option type | Store true |
| Option default value | NA |
| Option available value | NA |

Run post PAR Verilog simulation.

## 3.6 Lattice main options APP & Restrictions

### 3.6.1 Public options

#### 3.6.1.1 Design source and implementation location

A typical test suite will have the following structure:

############################folder structure############################

./test\_item --test item folder

#Position A

./suite1 --suite 1 folder

./suite2 --suite 2 folder

#Position B

./case1 --case 1 folder

./case2 --case 2 folder

#Position C

./model --models for this case

./others --NGO, NGD files

./par --project files

./sim --simulation files

./case3 --case 3 folder

./Vlib --simulation library for Modelsim

./suite3 --suite 3 folder

################################End################################

Lattice main script use “–top-dir”, “--job-dir”, “—design” and “—tag” to locate the source folder and the implementation folder positions.

A: “—top-dir” and “--design” are used to located the source path and in different locations (position A, B, C showing in the above structure) some options can be omitted.

1. BQS scripts folder located in Position A:

--top-dir must have located to which test suite

--design must have located to which design

1. BQS scripts folder located in Position B:

--top-dir can be omitted use current test suite directory

--design must have located to which design

1. BQS scripts folder located in Position C:

--top-dir can be omitted use current test suite directory

--design can be omitted use current design directory

B: “—job-dir” and “--tag” are used to located the implementation directory.

The values for these options will be:

1. Default values

|  |  |  |
| --- | --- | --- |
| Option name | Default value | Comments |
| --job-dir | Values of –top-dir |  |
| --tag | \_scratch |  |

In order to make the source case clean after your test flow, the script will create a folder which is named as $job-dir/$design/$tag to save all the results in it.

### 3.6.2 Project options

#### 3.6.2.1 –empty-lpf option

“--empty-lpf” option was focus on sweeping flows ([chapter 4.2](#_4.2_Sweeping_flow_1)), this option is used to clean up the LPF file before project run.

#### 3.6.2.2 –use-ori-clks option

“–use-ori-clks” option was focus on sweeping flows ([chapter 4.2](#_4.2_Sweeping_flow_1)).

The default sweeping flow will run MAP flow to get all clocks from the MAP report file and add these constraints for these clocks in the LPF file to run sweeping flow.

If this option used, it will parse the clocks from the original LPF file rather than from the MAP report file.

### 3.6.3 Implementation options

#### 3.6.3.1 Group default option

Start condition:

1. There is no group option find in command line.
2. There is no sweeping range in design information file (xxx.info, [chapter 3.3](#_3.3_Test_case))

Behavior:

Script will run PAR and trace for this design just like have option “—run-par-trce”.

#### 3.6.3.2 Push button flow

Just like the GUI operation, the script will run the processed which are defined in the command line like “--run-xxx”. These options can be found in the Appendix or the scripts short help message.

All general implementation options are add on options for example:

“—run-map” --script will run till MAP

“—run-map, --run-map-trce” --script will run till MAP and TRCE

“—run-map, --run-par, --run-export-vlg” --script will run till PAR and export post PAR back annotation file.

Option [--till-map] will stop after map flow done. The test flow only generates the resource utilization results.

All [--run-=xxx] options can be used in the other sweeping flows.

Pushbutton flow will be launched if set [--pushbutton] or no fmax sweeping range or no seed sweeping range found for this design.

#### 3.6.3.3 Fmax sweeping flow

If the design local info file has the key value pairs like ”fmax\_range =50, 300, 10” in the [qa] section, and no [--pushbutton] in the command line, the script will launch fmax sweeping flow by default. You can also define the fmax sweeping range by [--fmax-sweep=50 300 10] in the command line to run fmax sweeping flow.

#### 3.6.3.4 Seed sweeping flow

You must specify [--seed-lpf=xxx”] in design local info file to make the seed sweeping flow as you expected. Otherwise the input preference file (LPF) will only have default settings. Option [--seed-sweep=1 19 1] in command line will guide the script to run seed sweeping flow.

### 3.6.4 Simulation options

#### 3.6.4.1 Simulation add on options

All simulation options are add on options, this means script will run every option encountered.

Option [--sim-all] will run all simulation types.

If --sim-rtl find, will put results in $job\_dir/$design/$tag/sim\_rtl

If --sim-map-vlg find, will put results in $job\_dir/$design/$tag/sim\_map\_vlg

If --sim-map-vhd find, will put results in $job\_dir/$design/$tag/sim\_map\_vhd

If --sim-par-vlg find, will put results in $job\_dir/$design/$tag/sim\_par\_vlg

If --sim-par-vhd find, will put results in $job\_dir/$design/$tag/sim\_par\_vhd

## 3.7 Launch file writing

Launch files usually will be simply text files (in windows the file suffix name should be “.bat” , while in Linux it can be any suffix with the permit of execution) and can be edit by any text editor (Note pad, Crimson Editor…).

### 3.7.1 CMD line structure

There are five groups of options in our script:

1. Public options --script usage
2. Diamond options --Diamond install path
3. Project options --project information (device, strategy…)
4. Implementation options --how to run project
5. Simulation options --how to run simulation

Different option group are for different work flows in the whole Lattice main script run, see following picture:



1. Work flow and option groups association

So a typical Lattice main script command line will like the following:

“””

***Python < BQS\_script/bin/run\_lattice.py > < Public options > < Diamond options > < Project options > < Implementation options > < Simulation options >***

“””

1. Python: execute program which used to run the following argument string.
2. < BQS\_script/bin/run\_lattice.py >: script file, the script entry.
3. < Public options >: specify the public test suite information.
4. –top-dir: specify the test suite path
5. –job-dir: specify the work path(which place to run the test suite), default value is: the same value like --top-dir option
6. –design: which design should be run

\*Note: for all options in this group, see options description section

1. < Diamond options >: Diamond installs options.
2. –X, --x86: with this option means use 32bit Diamond.
3. –diamond: Diamond install path.

\*Note: for all options for this group, see options description section

1. < Project options >:
2. –devkit: allow you to setting the project device, it will overwrite the original device.
3. –synthesis: allow you to choice the synthesis tools (synplify or LSE).
4. < Implementation options >: options for how to run the project.

\*Note: for all options for this group, see options description section

1. < Simulation options >: options for how to run simulation.
2. --modelsim-path: specify Modelsim install path
3. --sim-rtl: run rtl simulation
4. --sim-map-vhd: run MapVHDL simulation

\*Note: for all options for this group, see options description section

\*Note:If scritp cannot find any option for <implement options> and <simulation options>, script will add one more implement option(--run-par-trace) automatically, this is for your convenience. It means if no implement and simulation options find in your command line, script will run PAR & Trace.

### 3.7.2 Demo CMD line writing

The test suite structure like the following:

Assume the test item folder was located in this directory: “c:/test/”.

############################folder structure############################

./test\_item --test item folder

./BQS\_scripts --BQS scripts folder

./suite1 --suite 1 folder

./suite2 --suite 2 folder

./case1 --case 1 folder

./case2 --case 2 folder

./model --models for this case

./others --NGO, NGD files

./par --project files

./sim --simulation files

./case3 --case 3 folder

./Vlib --simulation library for Modelsim

./suite3 --suite 3 folder

Launct.bat --launch file we need to write

################################ End################################

The test requirement for case1 will like the following:

1. My diamond install path is “c:/lscc/diamond/3.2\_x64” (same as default)
2. My diamond version is 64bit(same as default)
3. Place our result in default folder(\_scratch folder)
4. I want to run PAR and trace for this design
5. I want to run post MAP Verilog simulation with ModelSIM

Requirements to options:

1. Python
2. BQS\_script/bin/run\_lattice.py
3. < Public options >:

–top-dir=c:/test/test\_item/suite2

–job-dir= c:/test/test\_item/suite2 --same with –top-dir setting, omit it.

–design=case1

1. < Diamond options >:

!<-X, --x86> --we are using 64bit Diamond, omit it.

--diamond-path=<Install path> -- we have set this value ([chapter 3.2](#_3.2_Script_initialization)), omit it.

1. < Project options >:

Use default values --omit these options

1. < Implementation options >:
2. —run-par-trce< Simulation options >: --see section 4.2.3.3 for more details

--modelsim-path=<install path> --We have set this value ([chapter 3.2](#_3.2_Script_initialization)), omit it.

--run-modelsim --we need to use ModelSIM run simulation.

--sim-map-vlg --we need run post MAP Verilog simulation.

Note: “!<xx>” means this option not present.

So the finally command line for “case1” will be:

“””***python BQS\_script/bin/run\_lattice.py –top-dir=c:/test/test\_item/suite2 –design=case1 --run-par-trce --run-modelsim –sim-map-vlg***”””

## 3.8 Launch test suite

After we construct our test suite file, launch test suite will be an easy job, just double click the launch file on Windows or use command line (typically, “source <launch file name>”) on Linux.

## 3.9 Result check

This section is for general detail result check and debug.

### 3.9.1 Where is the implementation result

Implementation result location will depends on your command line options, the key options are list here:

1. –top-dir
2. –job-dir --default value is same as –top-dir option
3. --design
4. –tag --default value is “\_scratch”

The result path will be $job-dir/$design/$tag.

### 3.9.2 Some debug files

There are some files you may be interesting:

1. General debug files

|  |  |  |  |
| --- | --- | --- | --- |
| NO. | File name | Location | Function |
| 1 | local\_debug.ini | <tag> | All info script collected for this case |
| 2 | runtime\_console.log | <tag> | Console output info for this case |
| 3 | synthesis\_flow.tcl | <tag> | Project setting and synthesis flowTCL file generate by script |
| 4 | synthesis\_flow.time | <tag> | Command line for launch TCL file |
| 5 | synthesis\_flow.log | <tag> | Log info for synthesis flow run |
| 6 | foundry\_flow.tcl | <tag> | Project implementation TCL file |
| 7 | foundry\_flow.time | <tag> | Command line for Project implementation |
| 8 | foundry\_flow.log | <tag> | Log info for Project implementation |

\*NOTE:

1. <case>: real case name
2. <tag>: test case output directory ($job-dir/$design/$tag)

## 3.10 Report check

This section is for general report checking.

### 3.10.1 Where is test report

The location of test report is depends on the option values of “—check-rpt”.

By default the report name will be “check\_by\_flow.csv”

The location will be the value of “—top-dir”.

### 3.10.2 Report result definition

Lattice main script will run check flow according the “xx.conf” file located in test case folder(if script don’t find this file, PAR trace result will be checked by default).

If the check result is ok, you will see “PASS” in the report file for this case.

For detail check flow according with the configure file see [chapter 5.2](#_5.2_Behavior_check)

# 4 Test flow orientation

## 4.1 General flow

Section owner: Bohong

### 4.1.1 Object

General flow will focus on the general diamond project setting and implementation and check the result information, just like the GUI behavior.

### 4.1.2 Quick start

Quick start will list some command lines for you to make you launch test case (suite) quickly. Quick start also covers different test orientations:

1. How to write a command line based on Lattice main script to launch a test case.
2. How to write a command line based on suite support script to launch a test suite.

#### 4.1.2.1 Case support

All test command lines here are based on Lattice main script. Lattice main script has 5 option groups ([chapter 3.7.1](#_3.7.1_CMD_line)). We only focus on implementation option group here, for other option groups use default setting.

1. Run PAR and Trace:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx [–run-par-trce]”***

1. Run MAP flow:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --run-map”***

1. Run IBIS export:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --run-export-ibis”***

1. Update device and run PAR and Trace:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --devkit=xxx [--run-par-trce]”***

***Note:option in “[]” means this option can be omit.***

#### 4.12.2 Suite support

All test command lines here are based on suite support script ([chapter 5.1](#_5.1_suite_support)).

1. For test suite “suite”, launch 3 process to run test suite:

***“python BQS\_script/tools /suite\_support/suite\_support.py --suite-file=case\_list.ini --suite-name=suite --process=3 --lattice --options=" --diamond=C:/lscc/diamond/3.2\_x64 [–run-par-trce]”***

1. For test suite “suite”, launch 3 process to run test suite:

***“python BQS\_script/tools /suite\_support/suite\_support.py --suite-file=case\_list.ini --suite-name=suite --process=3 --lattice --options=" --diamond=C:/lscc/diamond/3.2\_x64 --run-export-ibis”***

***Note:option in “[]” means this option can be omit.***

### 4.1.3 Major options

Here is a list of major options for general flow use:

1. Major options for general flow

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option group** | **Option name** | **Specification** |
| 1 | Public Options | --top-dir=TOP\_DIR | specify the top source path name |
| 2 |  | --design=DESIGN | specify the design name |
| 3 |  | --job-dir=JOB\_DIR | specify the job working path name |
| 4 |  | --tag=TAG | specify the job tag name, default is \_scratch, which is created in the design job working path |
| 5 | Diamond Options | -X, --x86 | run with x86 vendor tools |
| 6 | Project Options | --devkit=DEVKIT | specify the devkit name |
| 7 |  | --synthesis=SYNTHESIS | specify synthesis name, else use default one |
| 8 | Implementation Options | --pushbutton | default pushbutton flow, run till the par trace |
| 9 |  | --run-translate | run Translate Design flow |
| 10 |  | --run-map | run Map flow |
| 11 |  | --run-map-trce | run Map Trace trace flow |
| 12 |  | --run-map-trce | run Map Trace trace flow |
| 13 |  | --run-map-vlg | generate Map Verilog Simulation File |
| 14 |  | --run-map-vhd | generate Map VHDL Simulation File |
| 15 |  | --run-par | run PAR flow |
| 16 |  | --run-par-trce | run Place & Route Trace flow |
| 17 |  | --run-par-iota | run I/O Timing Analysis flow |
| 18 |  | --run-export-ibis | run Export IBIS Model flow |
| 19 |  | --run-export-vlg | generate Export Verilog Simulation File |
| 20 |  | --run-export-vhd | generate Export VHDL Simulation File |
| 21 |  | --run-export-jedec | generate Export JEDEC File |
| 22 |  | --run-export-bitstream | generate Export Bitstream File |
| 23 |  | --till-map | run till map trace flow |

### 4.1.4 Work flow

The detail General work flows and the relationship with Lattice main script work flow show in the following picture:



1. General flow

As we see from above picture, general flow will launch TCL flow (belong to implementation flow) since project file find. Also the main work flow for general flow will be implementation flow including project setting and running. The simulation flow will be bypassed since there is no simulation option (we can start simulation by add simulation options).

### 4.1.5 General flow usage

#### 4.1.5.1 Environment setting

General setting: refer to [chapter 3.1](#_3.1_Environment_setting)

#### 4.1.5.2 Script initialization

General setting: refer to [chapter 3.2](#_3.2_Script_initialization)

#### 4.1.5.3 Test suite building

General setting: refer to [chapter 3.3](#_3.3_Test_case)

#### 4.1.5.4 Launch file writing

General writing: refer to [chapter 3.7](#_3.7_Launch_file)

#### 4.1.5.5 Launch test suite

General launch: refer to [chapter 3.8](#_3.8_Launch_test)

#### 4.1.5.6 Result check

General check: refer to [chapter 3.9](#_3.9_Result_check)

#### 4.1.5.7 Restrictions & status

NA

### 4.1.6 Examples

NA

## 4.2 Sweeping flow

Section owner: Jeffrey Ye

### 4.2.1 Object

Sweeping and seed are usually used in QoR, to monitor and judge the performance and quality of our EDA tools. Run sweeping and seed are required to specify format and customize options. There’re 2 entries for application. One uses suite mode, the other uses case mode.

### 4.2.2 Quick start

Quick start will list some command lines for you to make you launch test case (suite) quickly. Quick start also covers different test orientations:

1. How to write a command line based on Lattice main script to launch a test case.
2. How to write a command line based on suite support script to launch a test suite.

#### 4.2.2.1 Case support

All test command lines here are based on Lattice main script. Lattice main script has 5 option groups ([chapter 3.7.1](#_3.7.1_CMD_line)). We only focus on implementation option group here, for other option groups use default setting.

1. Run sweeping, delete original clock (in LPF file) frequency constraints and remain others, clock constraints are picked from MRP file. You can specify the design name, one design one command line, and change the options according to your requirement:

***“python trunk\bin\run\_lattice.py --job-dir=result\1 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --fmax-sweep=50 300 10 --synthesis=synplify --strategy=timing --design=CR53836\_uapc\_sap”***

1. Customer cases and run sweeping, empty customer LPF file, clock constraints are picked from MRP file:

***“python trunk\bin\run\_lattice.py --job-dir=result\2 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --fmax-sweep=50 300 10 --synthesis=synplify --strategy=timing --empty-lpf --design=CR53836\_uapc\_sap”***

1. Run multi-seed, use all original constraints and run multiple seed:

***“python trunk\bin\run\_lattice.py --job-dir=result\3 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --seed-sweep=1 5 1 --synthesis=synplify --strategy=timing --design=CR53836\_uapc\_sap”***

1. Run multi-seed without any constraints:

***“python trunk\bin\run\_lattice.py --job-dir=result\4 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --seed-sweep=1 5 1 --synthesis=synplify --strategy=timing --empty-lpf --design=CR53836\_uapc\_sap”***

1. Run sweeping and multi-seed, empty original lpf file, clock constraints are picked from mrp file, run sweeping and every frequency run multi-seed:

***“python trunk\bin\run\_lattice.py --job-dir=result\5 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --seed-sweep=1 5 1 --fmax-sweep=50 300 10 --fmax-seed --synthesis=synplify --strategy=timing --empty-lpf --design=CR53836\_uapc\_sap”***

1. Customer cases and run sweeping and multi-seed, delete original clock frequency constraints and remain other constraints, clock constraints are picked from mrp file, run sweeping and every frequency run multi-seed:

***“python trunk\bin\run\_lattice.py --job-dir=result\6 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --seed-sweep=1 5 1 --fmax-sweep=50 300 10 --fmax-seed --synthesis=synplify --strategy=timing --design=CR53836\_uapc\_sap”***

1. Use original clock to run sweeping, remain other constraints:

***“python trunk\bin\run\_lattice.py --job-dir=result\7 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --fmax-sweep=50 300 10 --use-ori-clks --synthesis=synplify --strategy=timing --design=CR53836\_uapc\_sap”***

1. Customer cases and use original clock to run sweeping and multi-seed, remain other constraints:

***“python trunk\bin\run\_lattice.py --job-dir=result\8 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --seed-sweep=1 5 1 --fmax-sweep=50 300 10 --fmax-seed --synthesis=synplify --strategy=timing --use-ori-clks --design=CR53836\_uapc\_sap”***

1. Run pushbutton:

***“python trunk\bin\run\_lattice.py --job-dir=result\9 --top-dir=cases --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --pushbutton --synthesis=synplify --strategy=timing --design=CR53836\_uapc\_sap”***

#### 4.2.2.2 Suite support

All test command lines here are based on suite support script ([chapters 5.1](#_5.1_suite_support)).

1. For test suite “***suite1***”, launch 3 process to run fmax sweeping flow:

***“python trunk\tools\ suite\_support\suite\_support.py --suite-file=case\_list.ini --suite-name=suite1 --process=3 --top-dir=./ --lattice --options="--job-dir=result\1 --block-lpf --diamond=C:\lscc\diamond\3.2\_x64 --fmax-sweep=50 300 10 --synthesis=synplify --strategy=timing " ”***

### 4.2.3 Major options

Here a list of major options for sweeping flow use:

1. Major options for sweeping flow

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option group** | **Option name** | **Specification** |
| 1 | Public Options | --top-dir=TOP\_DIR | specify the top source path name |
| 2 |  | --design=DESIGN | specify the design name |
| 3 |  | --job-dir=JOB\_DIR | specify the job working path name |
| 4 |  | --tag=TAG | specify the job tag name, default is \_scratch, which is created in the design job working path |
| 5 | Diamond Options | -X, --x86 | run with x86 vendor tools |
| 6 |  | --diamond=DIAMOND | specify Diamond install path |
| 7 | Project Options | --run-scuba | run scuba to update the model files |
| 8 |  | --devkit=DEVKIT | specify the devkit name |
| 9 |  | --synthesis=SYNTHESIS | specify synthesis name, else use default one |
| 10 |  | --goal=GOAL | specify synthesis optimization goal |
| 11 |  | --synthesis-done | synthesis already done |
| 12 |  | --synthesis-only | run synthesis only |
| 13 | Implementation Options | --fmax-sweep=<from> <to> <step> | specify fmax sweeping range |
| 14 |  | --seed-sweep=<from> <to> <step> | specify seed sweeping range |
| 15 |  | --fmax-seed | run seed sweeping for every fmax points |
| 16 |  | --double-step | double the running step for running faster |
| 17 | Simulation Options | NA |  |

### 4.2.4 Work flow

Sweeping flow includes fmax sweeping and seed sweeping flows.

#### 4.2.4.1 Fmax sweeping flow

The main flow is show as following:

Synthesis(translate)

Map

Select clock net

Write clocks to lpf file, change the frequency to run sweeping

Frequency is XXMHz,

run map, par, trace

Copy results to folder Target\_fmax\_50MHz

1. Fmax weeping flow

Fmax sweeping flow:

a) Run till map.

b) According to option setting, select clock from MRP file or original LPF file.

c) Write the clock net to LPF file, change the frequency from 50MHz to 300MHZ.

d) Once the LPF file has been changed, run TCL till par trace.

e) Copy the results from implementation folder to Target\_fmax\_XXMHz folder.

#### 4.2.4.2 Fmax sweeping flow

The main flow is show as following:

Synthesis(translate)

Map

Use fixed lpf setting

Create seed seeting to run sweeping

Seed is xx,

run map, par, trace

Copy results to folder Target\_seed\_xx

1. Seed sweeping flow

Seed sweeping flow:

a) Run till map.

b) According to option setting, select clock from MRP file or original LPF file.

c) Write the clock net to LPF file, change the frequency from 50MHz to 300MHZ.

d) Once the LPF file has been changed, run TCL till par trace.

e) Copy the results from implementation folder to Target\_fmax\_XXMHz folder.

### 4.2.5 Sweeping flow usage

#### 4.2.5.1 Environment setting

General setting: refer to [chapter 3.1](#_3.1_Environment_setting)

Make sure the machines have the environment variable: “SEDISABLE=1”.

Make sure the test folder have enough space for sweeping flow.

#### 4.2.5.2 Script initialization

General setting: refer to [chapter 3.2](#_3.2_Script_initialization)

#### 4.2.5.3 Test suite building

General setting: refer to [chapter 3.3](#_3.3_Test_case)

#### 4.2.5.4 Launch file writing

There are four groups of options in scripts. You can write the command as following and the file name extension is “bat”.

**“*Python <trunk/bin/run\_lattice.py> <Public options> <Diamond options> <Project options> <Implementation options>*”**

1. Public options: specify the public test suite information.
2. –job-dir: specify the work path. You can specify the path according to your condition. If not specify, it will use the same path with “—top-dir”.
3. –top-dir: specify the path of test suite.
4. –design: specify the cases name in test suite.
5. Diamond options: specify the Diamond path and version.
6. –diamond: Diamond install path.
7. –x86: specify the 32bit Diamond. If not specify, default is 64bit Diamond.
8. Project options: specify run what.
9. –synthesis: choose the synthesis tool(synplify or lse).
10. –strategy: choose the Diamond strategy(timing ,area, if not specify, use default Strategy1).
11. –fanout: specify the fanout limitation number.
12. –frequency: specify the synthesis frequency.
13. –mixed-drivers: set synthesis strategy “Resolve Mixed Drivers” Ture.
14. –empty-lpf: empty the original lpf file, delete all the constraints.
15. –use-ori-clks: use default lpf file to run the flow.
16. Implementation options: specify how to run.(you can just select only one Implementation option in scripts)
17. Sweeping: use option “—fmax-sweep”.
18. Seed: use option “—seed-sweep”.
19. Sweeping and seed: use options combination “—fmax-sweep”, “—seed-sweep” and “—fmax-seed”.
20. Smoke: use option “--smoke”, let the scripts to run the first point.

#### 4.2.5.5 Launch test suite

1) Windows:

a) Single process: double click the bat file.

b) Multi processes: use suite mode to run multi processes.

c) Multi processes: use tool multi lines to run multi processes. Write the command as following, “—do-file” specify the launch file. Double click multi times.

“***python trunk\tools\runLines\run\_lines.py --do-file=XX.bat*”**

2) Linux: use “source launch file name” in Terminal. We can also use tool multi lines to run multi processes.

#### 4.2.5.6 Result check

##### 4.2.5.6.1 Where is the simulation result

1. General results: refer to [chapter 3.9.1](#_3.9.1_Where_is)
2. Sweeping results:

Synthesis result: ./<case>/\_scratch/< implementation>/

Sweeping result: ./<case>/\_scratch/Target\_fmax\_XXXMHz/<implementation>/

Seed result: ./<case>/\_scratch/Target\_seed\_XX/<implementation>/

Sweeping and seed result: ./<case>/\_scratch/Target\_fmax\_XX\_seed\_XX/<implementation>/

##### 4.2.5.6.2 Some debug files

1. General debug files: refer to [chapter 3.9.2](#_3.9.2_Some_debug)
2. Sweeping debug files:

Synthesis log: ./<case>/\_scratch/set\_strategy\_and\_run.log

MPAR log: ./<case>/\_scratch/Target\_fmax\_XX/.log

MAP log: ./<case>/\_scratch/Target\_fmax\_XX/<implementation>/.mrp

PAR log: ./<case>/\_scratch/Target\_fmax\_XX/<implementation>/.par

TRACE log: ./<case>/\_scratch/Target\_fmax\_XX/<implementation>/.twr

#### 4.2.5.7 Scan report

Scan report need the support of scan report application tool. See [chapter 5.3](#_5.3_Scan_report)

1. You can get the help of options:

***“python trunk\tools\scan\_report\bin\run\_scan\_lattice\_step\_general\_case.py -h”***

1. Scan pushbutton report:

***“python trunk\tools\scan\_report\bin\run\_scan\_lattice\_step\_general\_case.py –job-dir=..\1\_case\_mode\result\9”***

1. Scan sweeping report:

***“python trunk\tools\scan\_report\bin\run\_scan\_lattice\_step\_general\_case.py –job-dir=..\1\_case\_mode\result\1 --flow=qor”***

1. Scan seed report:

***“python trunk\tools\scan\_report\bin\run\_scan\_lattice\_step\_general\_case.py –job-dir=..\1\_case\_mode\result\3 --flow=seed”***

1. Scan sweeping and seed report:

***“python trunk\tools\scan\_report\bin\run\_scan\_lattice\_step\_general\_case.py --job-dir=..\1\_case\_mode\result\5 --flow=qor”***

#### 4.2.5.8 Restrictions & status

NA

### 4.2.6 Examples

NA

## 4.3 Crossover flow

Section owner: Jason Wang

Crossover flow was defined for the requirement of run the same project with two Diamond versions.

Frontend (synthesis) use one Diamond version, Backend (Translate, MAP, PAR and TRCE) uses another Diamond version.

### 4.3.1 Object

This section will detail the usage of crossover flow, including: the details of crossover work flow, how to construct test suite, how to write command line, how to run result check and finally an example will be given.

Also some restrictions and test status will be list out to let you know the status of our BQS scripts.

### 4.3.2 Quick start

Quick start will list some command lines for you to make you launch test case (suite) quickly. Quick start also covers different test orientations:

1. How to write a command line based on Lattice main script to launch a test case.
2. How to write a command line based on suite support script to launch a test suite.

#### 4.3.2.1 Case support

All test command lines here are based on Lattice main script. Lattice main script has 5 option groups ([chapter 3.7.1](#_3.7.1_CMD_line)). We only focus on simulation option group here, for other option groups use default setting.

1. Run RTL simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx –diamond-fe=xxx –diamond-be=xxx --sim-rtl”***

1. Run post MAP Verilog simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx –diamond-fe=xxx –diamond-be=xxx --sim-map-vlg”***

#### 4.3.2.2 Suite support

All test command lines here are based on suite support script ([chapter 5.1](#_5.1_suite_support)).

1. For test suite “suite”, launch 3 process to run RTL simulation:

***“python BQS\_script/tools /suite\_support/suite\_support.py --suite-file=case\_list.ini --suite-name=suite --process=3 --lattice --options=" –diamond-fe=xx –diamond-be=xx”***

### 4.3.3 Major options

Here a list of major options for crossover flow use:

1. Major options for crossover flow

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option group** | **Option name** | **Specification** |
| 1 | Public Options | -d, --debug | print debug message |
| 2 |  | --top-dir=TOP\_DIR | specify the top source path name |
| 3 |  | --design=DESIGN | specify the design name |
| 4 |  | --job-dir=JOB\_DIR | specify the job working path name |
| 5 |  | --tag=TAG | specify the job tag name, default is \_scratch, which is created in the design job working path |
| 6 |  | --check-rpt=CHECK\_RPT | specify the check report file |
| 7 | Diamond Options | -X, --x86 | run with x86 vendor tools |
| 8 |  | --diamond-fe=DIAMOND\_FE | specify Diamond install path for running synthesis |
| 9 |  | --diamond-be=DIAMOND\_BE | specify Diamond install path for running mpar |
| 10 | Project Options | --run-scuba | run scuba to update the model files |
| 11 |  | --devkit=DEVKIT | specify the devkit name |
| 12 |  | --synthesis=SYNTHESIS | specify synthesis name, else use default one |
| 13 | Implementation Options | --pushbutton | default pushbutton flow, run till the par trace |
| 14 |  | --run-translate | run Translate Design flow |
| 15 |  | --run-map | run Map flow |
| 16 |  | --run-map-trce | run Map Trace trace flow |
| 17 |  | --run-map-trce | run Map Trace trace flow |
| 18 |  | --run-map-vlg | generate Map Verilog Simulation File |
| 19 |  | --run-map-vhd | generate Map VHDL Simulation File |
| 20 |  | --run-par | run PAR flow |
| 21 |  | --run-par-trce | run Place & Route Trace flow |
| 22 |  | --run-par-iota | run I/O Timing Analysis flow |
| 23 |  | --run-export-ibis | run Export IBIS Model flow |
| 24 |  | --run-export-vlg | generate Export Verilog Simulation File |
| 25 |  | --run-export-vhd | generate Export VHDL Simulation File |
| 26 |  | --run-export-jedec | generate Export JEDEC File |
| 27 |  | --run-export-bitstream | generate Export Bitstream File |
| 28 |  | --till-map | run till map trace flow |

### 4.3.4 Work flow

The detail crossover work flows is same as general flow, please refer to [chapter 4.1.4](#_4.1.4_Work_flow)

### 4.3.5 Crossover flow usage

#### 4.3.5.1 Environment setting

General setting: refer to [chapter 3.1](#_3.1_Environment_setting)

#### 4.3.5.2 Script initialization

General setting: refer to [chapter 3.2](#_3.2_Script_initialization)

For crossover flow here are some specific items:

Before launch script we should change or confirm some script setting in default.ini file (see [chapter 2.1](#_2.1_Scripts_directory) for file location):

**QA section**:

1. default.ini qa section value

|  |  |  |  |
| --- | --- | --- | --- |
| NO. | Option name | Default value | Comments |
| 1 | diamond | c:/lscc/diamond/3.2\_x64 | Diamond install path for windows |
| 2 | diamond\_lin | NA | Diamond install path for Linux |
| 3 | diamond\_sol | NA | Diamond install path for Solaris |
| 4 | diamond\_fe\* | NA | Diamond install path for frontend use |
| 5 | diamond\_be\* | NA | Diamond install path for backend use |
| 6 | modelsim\_path | C:\modeltech\_6.6d\win32 | ModelSIM install path |
| 7 | questasim\_path | NA | QuestaSIM install path |

\*Note: diamond\_fe, diamond\_be options were commented by semicolon by default, please **add** these two options in crossover flow.

#### 4.3.5.3 Test suite building

General setting: refer to [chapter 3.3](#_3.3_Test_case)

#### 4.3.5.4 Launch file writing

General writing: refer to [chapter 3.7](#_3.7_Launch_file)

#### 4.3.5.5 Launch test suite

General launch: refer to [chapter 3.8](#_3.8_Launch_test)

#### 4.3.5.6 Result check

General check: refer to [chapter 3.9](#_3.9_Result_check)

#### 4.3.5.7 Restrictions & status

NA

### 4.3.6 Examples

NA

## 4.4 Simulation flow

Section owner: Jason Wang

### 4.4.1 Object

This section will detail the usage of simulation flow, including: the details of simulation work flow, how to construct test suite, how to write simulation command line, how to run simulation result check and finally an example will be given.

Also some restrictions and test status will be list out to let you know the status of our BQS scripts.

### 4.4.2 Quick start

Quick start will list some command lines for you to make you launch test case (suite) quickly. Quick start also covers different test orientations:

1. How to write a command line based on Lattice main script to launch a test case.
2. How to write a command line based on suite support script to launch a test suite.

#### 4.4.2.1 Case support

All test command lines here are based on Lattice main script. Lattice main script has 5 option groups ([chapter 3.7.1](#_3.7.1_CMD_line)). We only focus on simulation option group here, for other option groups use default setting.

1. Run RTL simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --sim-rtl”***

1. Run post MAP Verilog simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --sim-map-vlg”***

1. Run post PAR Verilog simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --sim-par-vhd”***

1. Run RTL, post MAP Verilog, post PAR Verilog simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx --sim-rtl --sim-map-vlg --sim-par-vlg”***

1. Run RTL, post MAP VHDL, post PAR VHDL simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top\_dir=xxx --design=xxx --diamond=xxx --sim-rtl --sim-map-vhd --sim-par-vhd”***

1. Run RTL, post MAP VHDL/Verilog, post PAR VHDL/Verilog simulation with Active-HDL:

***“python ./BQS\_script/bin/run\_lattice.py --top\_dir=xxx --design=xxx --diamond=xxx --sim-all”***

1. Run RTL simulation with ModelSIM:

***“python ./BQS\_script/bin/run\_lattice.py --top\_dir=xxx --design=xxx --diamond=xxx --sim-rtl --run-modelsim”***

#### 4.4.2.2 Suite support

All test command lines here are based on suite support script ([chapter 5.1](#_5.1_suite_support)).

1. For test suite “suite”, launch 3 process to run RTL simulation:

***“python BQS\_script/tools/suite\_support/suite\_support.py --suite-file=case\_list.ini --suite-name=suite --process=3 --lattice --options=" --diamond=C:/lscc/diamond/3.2\_x64 –sim-rtl””***

1. For test suite “suite”, launch 3 process to run RTL simulation with ModelSIM:

***“python BQS\_script/tools/suite\_support/suite\_support.py --suite-file=case\_list.ini --suite-name=suite --process=3 --lattice --options=" --diamond=C:/lscc/diamond/3.2\_x64 –sim-rtl --run-modelsim””***

### 4.4.3 Major options

Here a list of major options for simulation flow use:

1. Major options for simulation flow

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option group** | **Option name** | **Specification** |
| 1 | Public Options | --top-dir=TOP\_DIR | specify the top source path name |
| 2 |  | --design=DESIGN | specify the design name |
| 3 |  | --job-dir=JOB\_DIR | specify the job working path name |
| 4 |  | --tag=TAG | specify the job tag name, default is \_scratch, which is created in the design job working path |
| 5 | Diamond Options | -X, --x86 | run with x86 vendor tools |
| 6 | Project Options | --devkit=DEVKIT | specify the devkit name |
| 7 |  | --synthesis=SYNTHESIS | specify synthesis name, else use default one |
| 8 | Implementation Options | --pushbutton | default pushbutton flow, run till the par trace |
| 9 | Simulation Options | --modelsim-path= | specify Modelsim install path |
| 10 |  | --questasim-path= | specify Questasim install path |
| 11 |  | --run-modelsim | run simulation with Modelsim |
| 12 |  | --run-questasim | run simulation with Questasim |
| 13 |  | --sim-rtl | run rtl simulation |
| 14 |  | --sim-map-vhd | run MapVHDL simulation |
| 15 |  | --sim-map-vlg | run MapVerilog simulation |
| 16 |  | --sim-par-vhd | run ParVHDL simulation |
| 17 |  | --sim-par-vlg | run PARVerilog simulation |
| 18 |  | --sim-all | run all simulation flow |

### 4.4.4 Work flow

Simulation flow will be launched when script find any simulation group options. Such as “—sim-rtl”, “—sim-map-vhd”…

The detail simulation work flows and the relationship with Lattice main script work flow show in the following picture:



1. Simulation work flow

All red arrows in the picture above link the main script work flows (total 6 main flows for our script). Here we detailed the simulation flow which constructed by the following two subroutines:

1. Simulation source file prepare: Script need to get the simulation source files according with the simulation flow we specified in the run command line. For example, if we need run map Verilog simulation (with –sim-map-vlg in command line) this subroutine will search the location of map .vo file which was generated by implement flow.
2. Simulation run: as we show in the picture in this subroutine, script will first collect run information (including: test bench top module, simulation run time which we listed in info file). Secondly, script will generate the default macro (.do) file if user not specified a valid macro file. Finally, script will launch simulation to generate simulation output file for future use.

\*Note:

We consider “Map vo output”, “Map vho output”, “Par vo output”, “Par vho output” (blocks show in figure1) as implementation flows, simulation flows only search these files for simulation run. As this strategy implementation flow will also sensitive for simulation flow options, such as “—sim-map-vlg”, and generate the output for simulation flow use.

### 4.4.5 Simulation flow usage

#### 4.4.5.1 Environment setting

General setting: refer to [chapter 3.1](#_3.1_Environment_setting)

#### 4.4.5.2 Script initialization

General setting: refer to [chapter 3.2](#_3.2_Script_initialization)

#### 4.4.5.3 Test suite building

General setting: refer to [chapter 3.3](#_3.3_Test_case)

#### 4.4.5.4 Launch file writing

General writing: refer to [chapter 3.7](#_3.7_Launch_file)

For simulation flow, here are some details for simulation options writing:

Currently we have the following simulation options:

1. --modelsim-path: specify Modelsim install path
2. --questasim-path: specify Questasim install path
3. --run-modelsim: run simulation with Modelsim
4. --run-questasim: run simulation with Questasim
5. --sim-rtl: run rtl simulation
6. --sim-map-vhd: run MapVHDL simulation
7. --sim-map-vlg: run MapVerilog simulation
8. --sim-par-vhd: run ParVHDL simulation
9. --sim-par-vlg: run PARVerilog simulation
10. --sim-all: run all simulation flow

We can simply construct them into three groups:

1. Group1: option a, b simulation tools install path.
2. Group2: option c, d which simulation tool will be used for script running.
3. Group3: option e-j which simulation flow will be run.

So the options for simulation section will be:

“””<Group1> < Group2> < Group3>”””

1. <Group 1>:

Simulation tools install path setting.

If Active-HDL used: omit this group

If ModelSIM used: “--modelsim-path” required (omit it after Script initialization)

If QuestaSIM used: “--questasim-path” required (omit it after Script initialization)

1. < Group2>:

Decide which simulation tool will be used.

If Active-HDL used: omit this group

If ModelSIM used: “—run-modelsim” required

If QuestaSIM used: “—run-questasim” required

1. < Group3>:

Decide which simulation flow will be run.

These options can be accumulated except “—sim-all” which means the sum of others options. This means you need one more option when you need do one more simulation flow.

If we need run RTL simulation, option is: --sim-rtl

If we need RTL and MAP Verilog, options are: --sim-rtl –sim-map-vlg

If we need MAP Verilog and PAR Verilog, options are: –sim-map-vlg –sim-par-vlg

If we need run all simulation flow, options can simply like: --sim-all

#### 4.4.5.5 Launch test suite

General launch: refer to [chapter 3.8](#_3.8_Launch_test)

#### 4.4.5.6 Result check

General check: refer to [chapter 3.9](#_3.9_Result_check)

For simulation flow, here are some details for simulation result checking:

##### 4.4.5.6.1 Where is the simulation result

Simulation result location will depends on your command line options, the key options are list here:

1. –job-dir --default value is same as –top-dir option
2. --design
3. –tag --default value is “\_scratch”
4. <simulation flow options>

For item 4 simulation flow options means:

1. Folder name for every simulation flow options

|  |  |  |
| --- | --- | --- |
| NO. | Option Name | Generate Name |
| 1 | --sim-rtl | sim\_rtl |
| 2 | --sim-map-vlg | sim\_map\_vlg |
| 3 | --sim-map-vhd | sim\_map\_vhd |
| 4 | --sim-par-vlg | sim\_par\_vlg |
| 5 | --sim-par-vhd | Sim\_par\_vhd |

So our simulation result path will be:

“ ***<job\_dir>/<design>/<tag>/<simulation\_folder>*** ”

For example:

In our command line setting for some test case:

--job-dir: not find. Default value is the value for –top-dir or test suite folder

--design: must have. Assume design name is “case1”.

--tag: not find. Default value is “\_scratch”

--sim-rtl:

--sim-map-vlg:

--sim-par-vlg:

So our RTL simulation result folder will be:

“”” ***<value for top\_dir or test suite folder>/case1/\_scratch/sim\_rtl*** ”””

The MAP Verilog simulation result folder will be:

“”” ***<value for top\_dir or test suite folder>/case1/\_scratch/sim\_map\_vlg*** ”””

The PAR Verilog simulation result folder will be:

“”” ***<value for top\_dir or test suite folder>/case1/\_scratch/sim\_par\_vlg*** ”””

##### 4.4.5.6.2 Some debug files

There are some files you may be interesting:

1. Simulation output files:

The simulation output file name will depends on your test bench writing.

1. Simulation macro files:

The macro file (.do) name will be “do\_<generate name>.do”, see table1 for generate name.

1. Simulation run command line:

The simulation command line will be stored here:” “run\_<generate name>.time”, see table1 for generate name.

1. Simulation log file:

Log file name will be: sim\_log.txt or “run\_<generate name>.log”, they are almost same with each other.

#### 4.4.5.7 Restrictions & status

Here is a table for BQS scripts restrictions and test status:

1. Script restrictions and test status:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NO. | simulation tool | Platform | Script Support | Test Status | Comments |
| 1 | Active-HDL | windows | YES | YES | Can use now |
| 2 | Active-HDL | Linux | NO | NO | no license |
| 3 | ModelSIM | windows | YES | YES | Can use now |
| 4 | ModelSIM | Linux | YES | NO |  |
| 5 | QuestaSIM | windows | YES | NO | no license |
| 6 | QuestaSIM | Linux | YES | NO | no license |

### 4.4.6 Examples

#### 4.4.6.1 Test suite folder structure

Here is basic test suite structure for simulation flow:

<..>/suite1

./BQS\_scripts/

./case1/

./case2/

…

./caseN/  
 ./Vlib/

./test.bat

#### 4.4.6.2 Lines in my launch file (test.bat)

The following are three lines in my launch file:

1. Line 1: Python BQS\_script/bin/run\_lattice.py –design=case1 –run-par-trce –sim-rtl
2. Line 2: Python BQS\_script/bin/run\_lattice.py –design=case1 –run-par-trce –run-modelsim –sim-map-vlg –sim-par-vlg
3. Line N: Python BQS\_script/bin/run\_lattice.py –tag=simulation –design=case1 –run-par-trce –sim-rtl

#### 4.4.6.3 Launch file explains

Every command line can be grouped as:

***“””***

***Python < BQS\_script/bin/run\_lattice.py > < Public options > < Diamond options > < Project options > < Implementation options > < Simulation options >***

***“””***

1. Line 1 Explains:

* < Public options > used to set the suite information:

--top-dir: since our test script located in test suite, omit this option

--job-dir: since we want to run design in the original place, omit this option

--tag: we use the default value, omit this option

* < Diamond options > used to set diamond install information:

-X, --x86: this is for 32bit Diamond, we are using 64bit now, omit this option

--diamond: we have set the right value in [default.ini](#_4.2.2_Script_initializes) file, omit this option

* < Project options >: used to set project information:

We use default project setting (do not change device, use different synthesis tools, or something else), omit these options

* < Implementation options >: used to set which diamond process will be run
* run MPAR & Trce< Simulation options >: used to set which simulation flow will be run

--modelsim-path: we use Active-HDL run simulation, omit this option

--questasim-path: same as above

--run-modelsim: same as above

--run-questasim: same as above

--sim-rtl: we want run RTL simulation, use this option

1. Line 2 Explains:

* < Public options > used to set the suite information:

Same as line1

* < Diamond options > used to set diamond install information:

Same as line1

* < Project options >: used to set project information:

Same as line1

* < Implementation options >: used to set which diamond process will be run

Same as line1

* < Simulation options >: used to set which simulation flow will be run

--modelsim-path: we have set the right value in [default.ini](#_4.2.2_Script_initializes) file, omit this option

--questasim-path: we use ModelSIM run simulation, omit this option

--run-modelsim: we want to run simulation with ModelSIM, use this option

--run-questasim: we use ModelSIM run simulation, omit this option

--sim-map-vlg: we want MAP Verilog simulation, use this option

--sim-par-vlg: we want PAR Verilog simulation, use this option

1. Line N Explains:

* < Public options > used to set the suite information:

--tag: we want all script implement results in “simulation” folder

* < Diamond options > used to set diamond install information:

Same as line1

* < Project options >: used to set project information:

Same as line1

* < Implementation options >: used to set which diamond process will be run

Same as line1

* < Simulation options >: used to set which simulation flow will be run

Same as line1

#### 4.4.6.4 Script implementation result

In this section we just list out the final result of this example suite, for detail result generation see  [chapter 4.4.5.6](#_4.4.5.6_Result_check)

1. Line 1 result:

RTL simulation result location: <..>/suite1/case1/\_scratch/sim\_rtl/

Simulation macro file: do\_sim\_rtl.do

Simulation output file: depends on our test bench file

Simulation log file: sim\_log.txt

Simulation command line file: run\_sim\_rtl.time

1. Line 2 result:
2. MAP Verilog simulation result:

Simulation result location: <..>/suite1/case1/\_scratch/sim\_map\_vlg/

Simulation macro file: do\_sim\_map\_vlg.do

Simulation output file: depends on our test bench file

Simulation log file:sim\_log.txt

Simulation command line file: run\_sim\_map\_vlg.time

1. PAR Verilog simulation result:

Simulation result location: <..>/suite1/case1/\_scratch/sim\_par\_vlg/

Simulation macro file: do\_sim\_par\_vlg.do

Simulation output file: depends on our test bench file

Simulation log file:sim\_log.txt

Simulation command line file: run\_sim\_par\_vlg.time

1. Line N result:

RTL simulation result location: <..>/suite1/case1/simulation/sim\_rtl/

Simulation macro file: do\_sim\_rtl.do

Simulation output file: depends on our test bench file

Simulation log file: sim\_log.txt

Simulation command line file: run\_sim\_rtl.time

## 4.5 Command flow

Section owner: Jason Wang

### 4.5.1 Object

This section will detail the usage of CMD flow, including: the details of CMD work flow, how to construct test suite, how to write command line, how to run CMD result check and finally an example will be given.

### 4.5.2 Quick start

Quick start will list some command lines for you to make you launch test case (suite) quickly. Quick start also covers different test orientations:

1. How to write a command line based on Lattice main script to launch a test case.
2. How to write a command line based on suite support script to launch a test suite.

#### 4.5.2.1 Case support

All test command lines here are based on Lattice main script. Lattice main script has 5 option groups ([chapter 3.7.1](#_3.7.1_CMD_line)). But CMD flow only responds for the first two option groups (Public Options & Diamond Options)

1. Run CMD flow, detail action was defined in project info file:

***“python ./BQS\_script/bin/run\_lattice.py --top-dir=xxx --design=xxx --diamond=xxx ”***

#### 4.5.2.2 Suite support

All test command lines here are based on suite support script ([chapter 5.1](#_5.1_suite_support)).

1. For test suite “suite”, launch 3 process to run CMD flow:

***“python BQS\_script/tools /suite\_support/suite\_support.py --suite-file=case\_list.ini –top-dir=./ --process=3 --lattice --options=" --diamond=C:/lscc/diamond/3.2\_x64”***

### 4.5.3 Major options

Here a list of major options for CMD flow use:

1. CMD flow major options

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option group** | **Option name** | **Specification** |
| 1 | Public Options | -d, --debug | print debug message |
| 2 |  | -D, --dry-run | Dry run the test flow, generate the Lattice Diamond File(.ldf) only |
| 3 |  | -F FILE, --file=FILE | specify user configuration file, which format is same as the conf\default.ini file |
| 4 |  | --conf=CONF | specify the configuration path name, default is conf. |
| 5 |  | --top-dir=TOP\_DIR | specify the top source path name |
| 6 |  | --design=DESIGN | specify the design name |
| 7 |  | --job-dir=JOB\_DIR | specify the job working path name |
| 8 |  | --tag=TAG | specify the job tag name, default is \_scratch, which is created in the design job working path |
| 9 |  | --qas | will run qas check flow |
| 10 |  | --check-rpt=CHECK\_RPT | specify the check report file |
| 11 | Diamond Options | -X, --x86 | run with x86 vendor tools |
| 12 |  | --diamond=DIAMOND | specify Diamond install path |
| 13 |  | --diamond-lin=DIAMOND\_LIN | specify Diamond install path(Linux) |
| 14 |  | --diamond-sol=DIAMOND\_SOL | specify Diamond install path(Solaris) |
| 15 |  | --diamond-fe=DIAMOND\_FE | specify Diamond install path for running synthesis |
| 16 |  | --diamond-be=DIAMOND\_BE | specify Diamond install path for running mpar |

### 4.5.4 Work flow

The detail CMD work flows and the relationship with Lattice main script work flow show in the following picture:



1. CMD work flow

All red arrows in the picture above link the main script work flows (total 6 main flows for our script).

While script entry CMD flow, script will run the following things:

#### 4.5.4.1 Parse info file

Script will parse [QA] section to get all source files. Here is a table of total available source file and project info.

1. QA section available options

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option Name** | **Default Value** | **Specification** |
| 1 | project\_name | PrjName |  |
| 2 | impl\_name | Impl1 |  |
| 3 | inc\_path | NA | macro search path |
| 4 | devkit | NA | device |
| 5 | src\_files | NA | RTL source files |
| 6 | lpf\_file | NA | LPF file |
| 7 | edf\_file | NA | EDF file |
| 8 | ngo\_file | NA | NGO file |
| 9 | ngd\_file | NA | NGD file |
| 10 | map\_ncd | NA | MAP NCD file |
| 11 | prf\_file | NA | PRF file |
| 12 | par\_ncd | NA | PAR NCD file |

Script will parse [CMD\_FLOW] section to get which implementation flow will be run, current available implementation flow list in the following table:

1. Implementation option

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option Name** | **Default Value** | **Available value** |
| 1 | run\_scuba | 0 | 0,1 |
| 2 | run\_synthesis | 0 | 0,1 |
| 3 | synthesis | synplify | synplify, lse |
| 4 | synp\_goal | Timing | Timing, Area, Balanced |
| 5 | run\_translate | 0 | 0,1 |
| 6 | run\_map | 0 | 0,1 |
| 7 | run\_map\_trce | 0 | 0,1 |
| 8 | run\_par | 0 | 0,1 |
| 9 | run\_par\_trce | 0 | 0,1 |

In the local info file, [command] section can be empty or have some specify command settings. The scripts will read the original [command] options which comes from:

* --file=xxx; the default value is the info file in the case folder.
* Default.ini which is in the configure path.

The priority of a command arguments is 1) case info file; 2) user file (--file=xx); 3) default.ini.

If there is nothing special command specified by user, the default command will be used. The default command line list in the following:

1. Default command line

|  |  |  |
| --- | --- | --- |
| NO. | CMD Name | Default CMD line |
| 1 | edif2ngd | edif2ngd -l "@(family)s" -d @(pty)s "@(edf\_file)s" "@(ngo\_file)s" |
| 2 | ngdbuild | ngdbuild -a "@(family)s" -d @(pty)s "@(ngo\_file)s" "@(ngd\_file)s" |
| 3 | map | map -a "@(family)s" -p @(pty)s -t @(pkg)s -s @(spd)s -oc @(opt)s "@(ngd\_file)s" -o "@(map\_ncd)s" -pr "@(prf\_file)s" -mp "@(mrp\_file)s" -lpf "@(lpf\_file)s" |
| 4 | map\_trce | trce -v 1 -gt -mapchkpnt 0 -sethld -o "@(tw1\_file)s" "@(map\_ncd)s" "@(prf\_file)s" |
| 5 | map\_vsim | ldbanno "@(map\_ncd)s" -n Verilog -o "@(map\_vo)s" -w -neg |
| 6 | map\_vhdsim | ldbanno "@(map\_ncd)s" -n VHDL -o "@(map\_vho)s" -w -neg |
| 7 | par | par -w -l 5 -i 6 -t 1 -c 0 -e 0 -exp parUseNBR=1:parCDP=auto:parCDR=1:parPathBased=OFF "@(map\_ncd)s" "@(par\_ncd)s" @(prf\_file)s |
| 8 | par\_trce | trce -v 10 -gt -sethld -sphld m -o @(twr\_file)s @(par\_ncd)s @(prf\_file)s |
| 9 | par\_iotiming | iotiming "@(par\_ncd)s" "@(prf\_file)s" |
| 10 | export\_vsim | ldbanno "@(par\_ncd)s" -n Verilog -o "@(par\_vo)s" -w -neg |
| 11 | export\_vhdsim | ldbanno "@(par\_ncd)s" -n VHDL -o "@(par\_vho)s" -w -neg |

#### 4.5.4.2 Get source files

Script will collect all source files list by info file [QA] section.

#### 4.5.4.3 Sanity check

CMD flow sanity check will be run by default to make sure all following flow will be run correctly.

Since we have source files and run command lines collect from info file, so we need to confirm the combination. Currently we support the following combination:

1. Supported combination

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **CMD Flow** | **Required Source Files** | **Required Project Setting** |
| 1 | run\_scuba | src\_files | devkit,project\_name,impl\_name |
| 2 | run\_synthesis | src\_files, lpf\_file | devkit,project\_name,impl\_name |
| 3 | synthesis | NA | NA |
| 4 | synp\_goal | NA | NA |
| 5 | run\_translate | edf\_file, lpf\_file | devkit,project\_name,impl\_name |
| 6 | run\_map | ngo\_file, lpf\_file | devkit,project\_name,impl\_name |
| 7 | run\_map\_trce | map\_ncd, prf\_file | project\_name,impl\_name |
| 8 | run\_par | map\_ncd,prf\_file | project\_name,impl\_name |
| 9 | run\_par\_trce | par\_ncd,prf\_file | project\_name,impl\_name |

#### 4.5.4.4 Run process

In run process flow, script will use the user defined command line(or default command line) and the source files collect from info file to run the required CMD flow defined by [CMD\_FLOW] section.

### 4.5.5 CMD flow usage

#### 4.5.5.1 Environment setting

General setting: refer to [chapter 3.1](#_3.1_Environment_setting)

#### 4.5.5.2 Script initialization

General setting: refer to [chapter 3.2](#_3.2_Script_initialization)

#### 4.5.5.3 Test suite building

General setting: refer to [chapter 3.3](#_3.3_Test_case)

#### 4.5.5.4 Launch file writing

General writing: refer to [chapter 3.7](#_3.7_Launch_file)

#### 4.5.5.5 Launch test suite

General launch: refer to [chapter 3.8](#_3.8_Launch_test)

#### 4.5.5.6 Result check

General check: refer to [chapter 3.9](#_3.9_Result_check)

#### 4.5.5.7 Restrictions & status

NA

### 4.5.6 Examples

NA

# 5 Application tool usages

## 5.1 suite support usage

Section owner: Yu Zhao

### 5.1.1 Object

Suite support is a tool located in the BQS/tools/suite\_support. You can use it to run any scripts either in just one process or in multi-process. With the tool you don’t have to write all the command in hand, it will produce the command in a file with timestamp.

Tips: It’s a good idea to run many cases not a signal design.

### 5.1.2 Quick start

[1] If you want to use the tool to run general flow, all the cases located in top-dir:

**Python suite\_support.py –top-dir=\*** **--lattice –options=” --diamond=c\lscc”**

[2] If you want to use the tool to run general flow, all the cases located in top-dir, but just run a special suite in the suite file:

**Python suite\_support.py –top-dir=\* --suite-file=\* --suite-name=\* --lattice –options=” --diamond=c\lscc”**

[3] If you want to use the tool to run general flow, all the cases located in top-dir, but just run cases located in suite file:

**Python suite\_support.py –top-dir=\* --suite-file=\* --lattice –options=” --diamond=c\lscc”**

[4] If you want to use the tool to run check flow, all the cases located in top-dir:

**Python** **suite\_support.py –top-dir=\* --check –options=” --diamond=c\lscc”**

[5] If you want to use the tool to run scan report, all cases located in top-dir:

**Python suite\_support.py --top-dir=\* --scan-cases --options=”--job-dir=\* --flow=\*”**

[6]If you want to use the tool to run other scripts you write which has an option as –design, all designs located in top-dir, you can run it as:

**Python suite\_support.py –top-dir=\* –options=”your\_scripts your\_options”**

**Python suite\_support.py –top-dir=designs –options=”python rundelete.py –diamond=c:\lscc –job=11.csv”**

[7]If you want to run general flow, all the cases located in top-dir with 3 processes:

**Python suite\_support.py --top-dir=\* --lattice –options=” --diamond=c\lscc” --process=3**

[8]if you want to write part of --options=”--your command--” in a file, you can write your command as:

**Python suite\_support.py --top-dir=\* --lattice –options=”part1” --cmd-file=file\_name**

### 5.1.3 Detail usage

Usage: suite\_support.py [options]

Options:

-h, --help show this help message and exit

--top-dir=TOP\_DIR specify top dir path

--suite-file=SUITE\_FILE

specify suite file

--suite-name=SUITE\_NAME

specify suite name

--lattice if you want to run run lattice with this script, add it in command

--check if you want to run run check with this script, add it in command

--scan-cases if you want to scan case resource results with this script, add it in command

--process=PROCESS specify process number

--options=OPTIONS specify options for the case running

--cmd-file=CMD\_FILE use the cmd in the file

--timeout=TIMEOUT specify timeout value in seconds during case running, default is 5 hours

When you use the tool, you have to make sure you have added --top-dir in the command. If you don’t set –top-dir=\* in command, you have to set it in the --options as --options=”--top-dir=\* …”

If you don’t set --suite-file in the command, the script will collect all cases in top-dir.

If you set top-dir and –suite-file but no –suite-name in the command, the script will collect all cases in the suite file. The case directory will be:

Top\_dir/section\_name/design

\*section\_name: the section name in the suite file

If you set top-dir, suite-file, suite-name in the command, the script will collect all cases in the sutie name section. The case directory will be:

Top\_dir/suite\_name/design

Below chart shows you the algorithm of searching cases:

YES

Find all cases located as:

Top\_dir/\*/suite\_content

\*:all suite section name in the ini file

**NO**

Suite name

YES

[Create list ]Find all cases**[1]** located as:

Top\_dir/\*/case\_name

**NO**

suite-file

YES

Return

**NO**

Top\_dir

Begin find case

Find all cases located as:

Top\_dir/suite\_name/suite\_content

[1]

*In the script, I will iterate the directory, find any sub-directory that contain info or par/ldf file. If the script finds a directory contains info or par/ldf file, the directory will be the case directory.*

--lattice: the tool will add run\_lattice.py in the signal case running command

--check: the tool will add check.py in the signal case running command.

--options: all the options that you added to the scripts you run the cases.

You don’t have to add python run\_lattice.py or python check.py at here.

You have to write the full command you run the cases.

When you write this option, you have to keep in mind that you have to write all the options run your script but not write –design=\*\*, as –design will be added by the tool.

--process: the tool will run the cases with mulit-process and the process number you specified.

--timeout: This option used to kill the sub-process if the running time is longer than set value. The default is 5 hours.

--cmd-file: write partial command in --options into a file.

Example:

suite\_support.py --top-dir=./ --lattice --suite-file=try.txt --suite-name=ss --cmd-file=aa.txt

aa.txt:

--diamond=C:\lscc\diamond\3.2\_x64 --block-lpf

You can write it different options in the same line or multi-lines.

How to write suite file:

**[suite\_name1**] suite options

Case\_name1 case\_options

Case\_name2 case\_options

…

[**suite\_name2**] suite options

Case\_name1 case\_options

Case\_name2 case\_options

Example:

[suite] --run-export-bitstream

24\_preserve\_xo2\_jtag –run-map-par

22\_wake\_on\_lock/sys\_config\_ecp3

23\_wake\_up/sys\_config\_ecp3

Reference:

Develop demo:

D:\Python27\python trunk\tools\scan\_report\tools\suite\_support\suite\_support.py --top-dir=suite --lattice --options="--report=aa.csv --report-path=C:\Users\yzhao1\Desktop\alcatel42 "

D:\Python27\python trunk\tools\scan\_report\tools\suite\_support\suite\_support.py --top-dir=C:\Users\yzhao1\Desktop\suite\_test1 --suite-file=case\_list.ini --lattice --options=" --diamond=C:\lscc\diamond\3.2\_x64 --synthesis=synplify"

python D:\test\_dir\bchen\_regression\D3\_2\dd\BE\_14\_Preference\merge\tools\scan\_report\tools\suite\_support\suite\_support.py --suite-file=case\_list.ini --suite-name=03\_PIO --top-dir=03\_PIO --options="python run\_delete.py --diamond=C:\lscc\diamond\3.2\_x64 --synthesis=synplify"

How to write check-conf file:

The check-conf file should be a csv file. The first line should be the title line. Till now, we just support check\_lines in this way.

|  |  |  |  |
| --- | --- | --- | --- |
| Case\_ID | Check\_file | Check\_method | message |
| warning/W\_BM100 | synthesis.log | check\_lines | resetting it to 1MHz(1000ns) |

The first line is the title. You should never change the titles (including character and capital format).

For the Case\_ID, Just write the related path according to the top-dir used in the options.

For Check\_file, Just write the file related path according to the case directory

For Check\_method, Just write check\_lines.

For message, write the line content you want to find in the check file.

## 5.2 Behavior check usage

Section owner: Jason Wang

### 5.2.1 Object

Behavior check usually performs checking report after running flow or other operation.

### 5.2.2 Quick start

**python trunk\tools \suite\_support\check.py –help**

You can use this demo to get detailed help for **check** script**.**

**python trunk\tools\ check\****check.py** **--design=07\_i2c\_port\_enable** **--top-dir=test**

Generally we should specify the --top-dir which is your work dir, --design which specify the design name.

The check is based on the config file. If there is any fail in the check-report, you should check the config file and debug with the test object.

**python trunk\tools\ suite\_support\suite\_support.py --top-dir=d:\test-dir\bchen\D3\_2 --suite-file=case\_list.ini --suite-name=suite --check --options=" --report-path=rpt --report=check\_result.csv"**

This case is applied to check the report by suite, and it specify the report-path and report name.

### 5.2.3 Detail Usage

Usage: check.py [options]

Options:

-h, --help show this help message and exit

--top-dir=TOP\_DIR specify top working directory

--design=DESIGN specify design name

--report-path=REPORT\_PATH

specify where you want to store the report

--tag=TAG replace the tag in the conf file

--report=REPORT specify report name, default is check.csv

For –tag option, the script will use the TAG value to replace “\*tag\*” in the conf file. The updated conf file will be stored in the case directory.

How to write conf file:

Scripts Usage:

-----------------------------------------------Part I: options-------------------------------------------------------

1. common options for total.py and run\_synthesis.py

--help, -h show this help message and exit

--top-dir=TOP\_DIR Specify top working directory name

--suite=SUITE Specify the test suite you want to run

--id=ID Specify project ID name

--family=FAMILY Specify family name

--group=GROUP Specify group name

--synthesis=SYNTHESIS Specify synthesis tool name

--bit64 Use 64bit Windows system

--clean-up Clean up working directory

--diamond=DIAMOND\_PATH Specify Lattice Diamond install path

2. flow options:

(1)for run\_synthesis.py

--run-scuba Update model files, should be used with other flow options.

--run-scuba-only Update model files only

(2)for total.py

--trace-only Run Trace only

--run-trace Run till Trace

--run-map Run till map

--run-par Run till par

--map-only Only run map

--par-only Only run par

--map-trace-only Only map\_trac

--produce\_report\_only Only auto check and generate report

--bit-stream run bitstream: from edf2ngd to par and bitstream

-----------------------------------------------Part II: configuration------------------------------------------------

1.caselist.ini format:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[suite1] --suite name is also the folder name

design1 = 01\_NoPortName

design2 = 01\_preference\02\_ClosestClockEdge --can use relative path to infer the test case

#design2 = 01\_preference\03\_PAR\_ADJ --support "#" to pass the case

...

[suite2]

design1 = PLL\_config1\_Four\_Outputs\_normal

design2 = 03\_PAR\_ADJ

design3 = 04\_HoldMargin

...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.config.conf format:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[configuration information]

area = STA

type = Arc

[method]

check\_lines\_1 = 1

check\_lines\_2 = 1

check\_block = 1

check\_data = 1

check\_flow = 1

[check\_lines\_1]

title = check\_test1

file = \_scratch\impl\NoPortName\_one.twr

check\_1 = Preference: FREQUENCY PORT "clka" 10.000000 MHz ;

check\_2 = 2 items scored

[check\_lines\_2]

title = check\_test2

file = \_scratch\impl\NoPortName\_one.twr

check\_1 = Preference: FREQUENCY 20.000000 MHz ;

check\_2 = 4 items scored

[check\_block]

title = check\_blocks

compare\_file = \_scratch\impl\NoPortName\_one.jed

golden\_file = Gold\_NoPortName\_one.jed

[check\_data]

file=\_scratch\impl\NoPortName\_one.twr

start\_line = Preference: INPUT\_SETUP GROUP "Data" 5.000000 ns CLKPORT "Clock" ;

result=6,8

line1=20,1,-

line2=19,1,+

line3=21,1,-

line4=22,1,-

line5=23,1

[check\_flow]

file= \_scratch\impl\NoPortName\_one.par

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NOTE:(1) for auto check, one (or more than one) method failure will lead to the case's failure in report.

(2) if the item value is set to 1, then script will enable this method

(3) if one method is used more than once, it can be named as check\_lines\_1, check\_lines\_2, ...

-----------------------------------------------Part III: Method------------------------------------------------

1.check\_lines

(1)format

[check\_lines]

title = check\_test1

file = <path>\<file>

check\_1 = <string\_1>

times = <number> --optional

check\_<num> = <string\_2>

(2)description

this method will try to find the <string\_1> assigned by check\_1 in <file> and treat it as a start point (line 1),

(if times option is used, sripts will find <string\_1> <number> times and then treat the last one as start point),

then try to check whether line <num> (a shift value) has <string\_2>. if yes, the result of this method is true.

2.check\_data

(1)format

[check\_data]

file=<path>\<file>

start\_line = <string\_1>

times = <number> --optional

result = <num> / <line>,<shift>

line<num1> = <num>,<operation\_symbol> / <line>,<shift>,<operation\_symbol>

line<num2> = <num>,<operation\_symbol> / <line>,<shift>,<operation\_symbol>

...

line<numn> = <num> / <line>,<shift>

(2)description

this method will try to find the <string\_1> assigned by start\_line in <file> and treat it as a start point (line 1),

(if times option is used, sripts will find <string\_1> <number> times and then treat the last one as start point),

then try to calculate the the result.

user can either use the absolute number or to use the line + shift to indicate the number.

3.check\_block

(1)format

[check\_block]

title = check\_blocks

compare\_file = <path>\<file>

golden\_file = <Gold\_file>

(2)description

this method will try to compare the <Gold\_file> and <file>, if <Gold\_file> is included in <file>, return true.

4.check\_flow

(1)format

[check\_flow]

file=<path>\<file>

(2)description

this method will try to find the string "All signals are completely routed." in the par report.

<file> need to be a par report.

5.check\_multiline

(1)format

[check\_multiline]

file = <path>\<file>

check\_line = <strings\_in\_multi\_lines>

(2)description

Sometimes, the string you want to check will be divided and printed in multi-lines due to different reasons (like the length of path may be changed and affect the message you want).

This method will check a continuous string regardless the "space" and "line break".

6.check\_no

(1)format

[check\_no]

file = <path>\<file>

check\_line =<string>

(2)description

this method will try to find the <string>. If the <string> is not found, the method return pass.

7.check\_grep

(1)format

[check\_grep]

file = <path>\<file>

grep = <grep>

modifier = <modifier> --optional

(2)description

This method will try to search the file with the given "regular expression" list by check\_grep.

modifier will be used to list the search behavior, such as: re.I or re.IGNORECASE : ignore case sensitive.

For example:

if you want to check the format of the following string:

Number of registers: 4 out of 877 (0%)

you can use:

grep = number of registers:\s+(\d+?)\sout\sof\s+(\d+?)\s\(

modifier = re.IGNORECASE

8. check\_compare\_par

This method used to check the seed order.

1. format:

[method]

check\_compare\_par = 1

[check\_compare\_par]

mode = <MODE> at here, mode can be valued as ws or ts. The default is ws

par\_dir = <dir>, the par file directory.

9. check\_sdf

This method used to check the sdf file

1. format:

[method]

check\_sdf = 1

[check\_sdf]

sdf\_dir = <dir>, the directory of sdf file directory.

## 5.3 Scan report usage

Section owner: Yu Zhao

### 5.3.1 Object

The scan report tool is located in trunk\tools\scan\_report. You can use it to scan resources after the cases run over. Till now, we support multi-flows as: QoR, General, and Seed.

### 5.3.2 Quick Start

**[1] Scan a case for qor:**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92 --design=21\_orca\_tut –flow=qor

**[2] Scan all case for qor:**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92 –flow=qor

**[3] Scan all case for qor and write pass/fail log files and also write rerun case bat file.**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92 –flow=qor –qor-auto

**[4] Scan all case for run mutli-seed:**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92 –flow=seed

**[5] Scan all case for tcl result:**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92

**[6] Scan cases and specify the report name:**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92 –flow=seed –report-name=’a.csv’

**[7] Scan cases and specify the report path:**

Python run\_scan\_lattice\_step\_general\_case.py --job-dir=Sapphire\_92 **--report-path=\***

### 5.3.3 Detail Usage

Options

|  |  |
| --- | --- |
| **-h, --help** | show this help message and exit |
| **--job-dir=JOB\_DIR** | specify job working path |
| **--design=DESIGN** | specify design name |
| **--conf-file=CONF\_FILE** | specify the configure file |
| **--pap** | dump Lattice Performance Achievement Percentage data |
| **--report-name=REPORT\_NAME** | specify report\_name you want to store |
| **--report-path=REPORT\_PATH** | specify the report path |
| **--force-na** | force the data become NAN if cannot get data from par file |
| **--special-structure=SPECIAL\_STRUCTURE** | specify the special structure for the case |
| **--flow=FLOW** | specify the case flow |
| **--qor-auto** | Write the pass/fail log in QoR automation flow |
| **--list-order** | sort the report as special order listed in the case\_list file located in conf directory |
| **--name-list=NAME\_LIST** | specify the list name in the case\_list.conf, this used  for write sorted report |
| **--ldf=LDF** | specify the ldf file name in the case |

\*Attention: For special order in the report, If you definitely know the order name, you can set –name-list=\* in the command. Or you just need to add –list-order in the command.

## 5.4 Multi line usage

You can read help for suite\_support in the upper chapter. Just add –process=\*(integer number )

# 6 Reference

BQS\_requirement.docx [\\d25609\BQS\_script\DOC\BQS\_requirement.docx](file:///\\d25609\BQS_script\DOC\BQS_requirement.docx)

BQS\_enhancement.docx [\\d25609\BQS\_script\DOC\BQS\_enhancement.xlsx](file:///\\d25609\BQS_script\DOC\BQS_enhancement.xlsx)

BQS\_spec\_case\_build.docx [\\d25609\BQS\_script\DOC\BQS\_spec\_case\_build.docx](file:///\\d25609\BQS_script\DOC\BQS_spec_case_build.docx)

BQS\_spec\_suite\_build.docx [\\d25609\BQS\_script\DOC\BQS\_spec\_suite\_build.docx](file:///\\d25609\BQS_script\DOC\BQS_spec_suite_build.docx%20)

# 7 Demo case & suite

## 7.1 Test case demo

See location: <http://d25315/viewvc/auto/bqs_scripts/trunk/docs/>

## 7.2 Test suite demo

See location: <http://d25315/viewvc/auto/bqs_scripts/trunk/docs/>

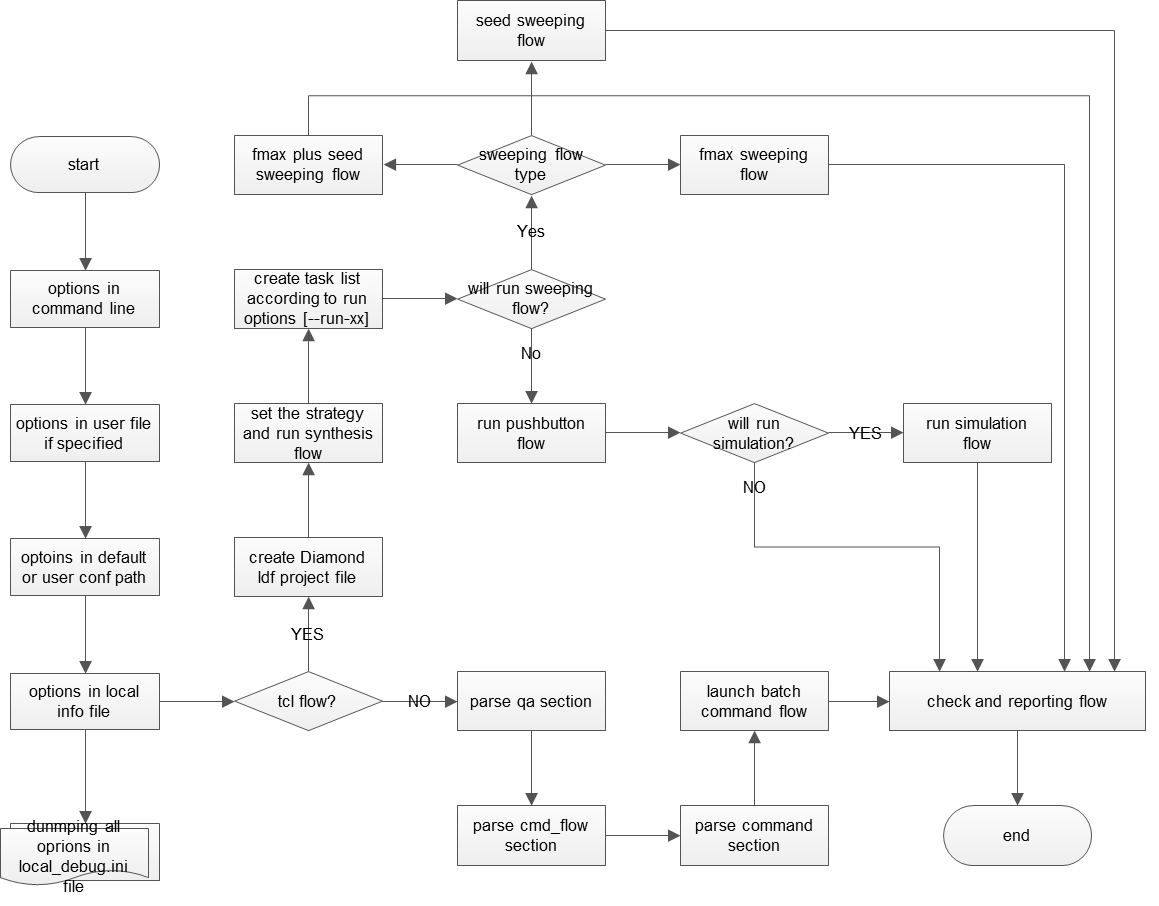
# 8 Appendices

## 8.1 Lattice main script options

1. Lattice main script options

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **Option group** | **Option name** | **Specification** |
| 1 | Public Options | -d, --debug | print debug message |
| 2 |  | -D, --dry-run | Dry run the test flow, generate the Lattice Diamond File(.ldf) only |
| 3 |  | -F FILE, --file=FILE | specify user configuration file, which format is same as the conf\default.ini file |
| 4 |  | --conf=CONF | specify the configuration path name, default is conf. |
| 5 |  | --top-dir=TOP\_DIR | specify the top source path name |
| 6 |  | --design=DESIGN | specify the design name |
| 7 |  | --job-dir=JOB\_DIR | specify the job working path name |
| 8 |  | --tag=TAG | specify the job tag name, default is \_scratch, which is created in the design job working path |
| 9 |  | --qas | will run qas check flow |
| 10 |  | --check-rpt=CHECK\_RPT | specify the check report file |
| 11 | Diamond Options | -X, --x86 | run with x86 vendor tools |
| 12 |  | --diamond=DIAMOND | specify Diamond install path |
| 13 |  | --diamond-lin=DIAMOND\_LIN | specify Diamond install path(Linux) |
| 14 |  | --diamond-sol=DIAMOND\_SOL | specify Diamond install path(Solaris) |
| 15 |  | --diamond-fe=DIAMOND\_FE | specify Diamond install path for running synthesis |
| 16 |  | --diamond-be=DIAMOND\_BE | specify Diamond install path for running mpar |
| 17 | Project Options | --run-scuba | run scuba to update the model files |
| 18 |  | --devkit=DEVKIT | specify the devkit name |
| 19 |  | --synthesis=SYNTHESIS | specify synthesis name, else use default one |
| 20 |  | --use-sdc | use sdc file for synthesis flow if found it |
| 21 |  | --strategy=STRATEGY | specify Diamond strategy name, if None, use Strategy1 or design current strategy |
| 22 |  | --goal=GOAL | specify synthesis optimization goal |
| 23 |  | --fanout=FANOUT | specify fanout limitation number |
| 24 |  | --frequency=FREQUENCY | specify the synthesis frequency constraint number |
| 25 |  | --mixed-drivers | set Resolve Mixed Drivers True |
| 26 |  | --block-lpf | Do NOT transmit constraint from sdc/ldc to lpf/prf file |
| 27 |  | --set-strategy=SET\_STRATEGY | set strategy, example:--set-strategy="syn\_res\_sharing=False maptrce\_check\_unconstrained\_connections=True |
| 28 |  | --empty-lpf | use default lpf file to run the first step flow |
| 29 |  | --use-ori-clks | use original clocks in the flow |
| 30 |  | --synthesis-done | synthesis already done |
| 31 |  | --synthesis-only | run synthesis only |
| 32 |  | --clean | clean the working path before running the flow |
| 33 | Implementation Options | --pushbutton | default pushbutton flow, run till the par trace |
| 34 |  | --run-translate | run Translate Design flow |
| 35 |  | --run-map | run Map flow |
| 36 |  | --run-map-trce | run Map Trace trace flow |
| 37 |  | --run-map-trce | run Map Trace trace flow |
| 38 |  | --run-map-vlg | generate Map Verilog Simulation File |
| 39 |  | --run-map-vhd | generate Map VHDL Simulation File |
| 40 |  | --run-par | run PAR flow |
| 41 |  | --run-par-trce | run Place & Route Trace flow |
| 42 |  | --run-par-iota | run I/O Timing Analysis flow |
| 43 |  | --run-export-ibis | run Export IBIS Model flow |
| 44 |  | --run-export-vlg | generate Export Verilog Simulation File |
| 45 |  | --run-export-vhd | generate Export VHDL Simulation File |
| 46 |  | --run-export-jedec | generate Export JEDEC File |
| 47 |  | --run-export-bitstream | generate Export Bitstream File |
| 48 |  | --till-map | run till map trace flow |
| 49 |  | --fmax-sweep=<from> <to> <step> | specify fmax sweeping range |
| 50 |  | --seed-sweep=<from> <to> <step> | specify seed sweeping range |
| 51 |  | --fmax-seed | run seed sweeping for every fmax points |
| 52 |  | --double-step | double the running step for running faster |
| 53 |  | --smoke | run smoke test |
| 54 | Simulation Options | --modelsim-path=MODELSIM\_PATH | specify Modelsim install path |
| 55 |  | --questasim-path=QUESTASIM\_PATH | specify Questasim install path |
| 56 |  | --run-modelsim | run simulation with Modelsim |
| 57 |  | --run-questasim | run simulation with Questasim |
| 58 |  | --sim-rtl | run rtl simulation |
| 59 |  | --sim-map-vhd | run MapVHDL simulation |
| 60 |  | --sim-map-vlg | run MapVerilog simulation |
| 61 |  | --sim-par-vhd | run ParVHDL simulation |
| 62 |  | --sim-par-vlg | run PARVerilog simulation |
| 64 |  | --sim-all | run all simulation flow |

## 8.2 Lattice main script flow chart



1. Lattice main script flow chart