TGEN_ENV 用户手册

by Tencent Penglai Lab

1. 文档说明

gen_env.py 腾讯蓬莱实验室开发的一个产生以 UVM 方法论为基础的 system Verilog 验证环境的自动化脚本，本文档是脚本的应用说明，需要读者有 UVM 验证方法学基础和 SystemVerilog 的语言基础。

2. 脚本使用说明

2.1. 脚本运行准备

在 tgen_env 目录下包含 3 个文件(夹)，为脚本运行的必备文件：

- gen_env.py：脚本本身
- env_cfg.ini：生成验证环境的配置文件
- 文件夹 common_src：存放生成验证环境所需要的一些基本使用脚本及代码库

如果需要在其他文件夹使用脚本，需要将以上 3 个文件全部拷贝到同一文件夹下，才能正常运行环境产生脚本
脚本使用 python3，请读者确认自己 python 版本并将自己的解释器连接到对应位置：/usr/bin/python3 ；
脚本使用的 python3 的库包括：os, sys, time, re, random, ast, shutil, configparser, tkinter；如果没有，请安装对应的库，否则脚本运行失败。
其中 configparser 和 tkinter 必须手动安装，安装命令如下：

- tkinter

```
yum install python3-tk* -y
yum install tk-devel -y
```

- configparser

```
pip3 install configparser
```

其他库为 python 自带基本库，如因为其他库不存在而报错，请自行百度安装。

2.2. 脚本执行命令

脚本可以通过以下三种方式执行产生验证环境或 env_cfg.ini 文件

- 读取 xxx.ini(如 env_cfg.ini)配置环境产生验证环境

./gen_env.py xxx.ini

- 通过填写 gui 配置对话框后产生验证环境

./gen_env.py

gui 界面如图

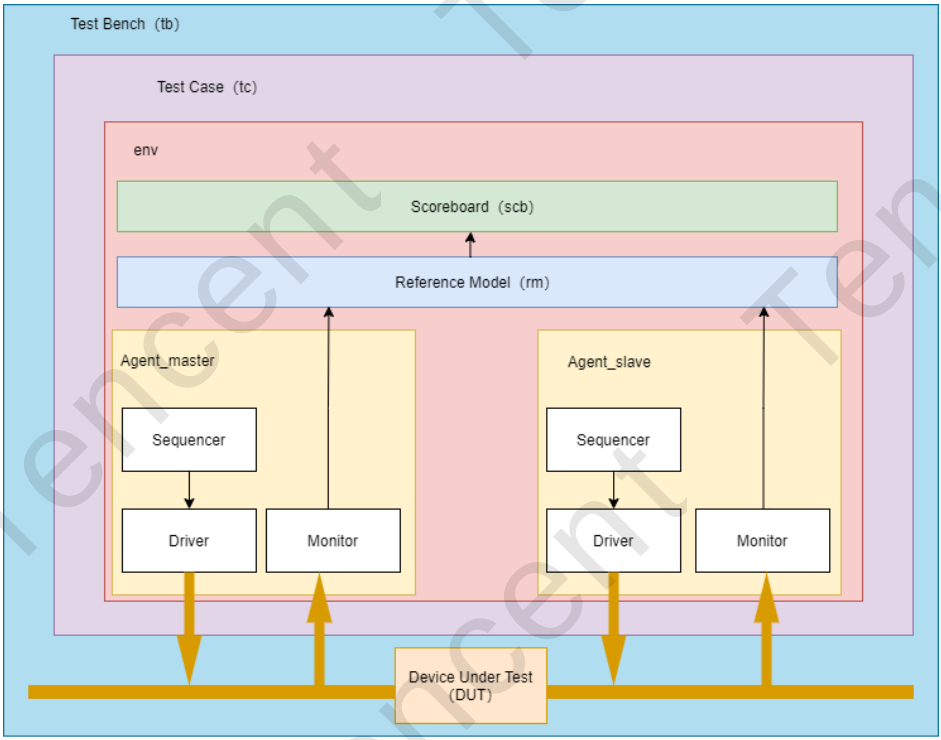
- 填写 gui 配置对话框产生配置文件 env_cfg.ini

./gen_env.py gen_ini

3. 生成验证环境结构和目录结构说明

3.1. 验证环境结构

脚本按照下图 UVM 验证环境结构图来生成环境：



- Driver：与 DUT 建立连接；向 sequencer 请求 sequence，并在发送完成后，返回一个 done 给 sequencer。将 transaction（uvm 环境内部数据结构）转化成 interface 数据，进行发送；
- Monitor：与 DUT 建立连接；与 model 和 scoreboard 建立通信；将 interface 数据转化成 transaction 数据，进行接收，并且发送给 reference model；
- Sequencer：调度 sequence 和 driver，确保它们正常进行 transaction 的产生和发送；
- Agent：实例化 driver、monitor、sequencer；连接 drv 和 sequencer；准备好发送数据给 scoreboard 或者 reference model；
- Reference Model：接收 agent 发送过来的 transaction，发送 transaction 给 scoreboard；建立与 DUT 同样功能的模型，让 transaction 包进入模型进行工作，然后将输出发送给 scoreboard；
- scoreboard：准备好接收 monitor 和 model 的数据 port；并行接收数据，然后进行 compare、report 等操作；
- env：实例化 agent、model、scoreboard；连接 reference model 和 scoreboard，reference model 和 agent；
- tc：实例化 env；控制改变输入激励，以达到不同的测试效果；

- tb：实例化 DUT，通过 interface 将 DUT 信号和 agent 相连接；

3.2. 目录结构

脚本认为项目的目录结构为固定结构，如下图所示：

```
|-- rtl
|-- scr
|   `-- verif
|-- ver
|   |-- bt
|   |-- cmodel
|   |-- common
|   |   |-- agent
|   |   `-- tcnt_base
|   |       `-- src
|   |-- formal
|   |-- fw
|   |-- it
|   |-- st
|   `-- ut
|       `-- data_bypass
|           |-- agent
|           |   `-- data_in_agent
|           |       `-- src
|           |-- cfg
|           |   |-- vcs_mk
|           |   |-- verif
|           |   `-- xrun_mk
|           |-- common
|           |   `-- data_bypass_common
|           |       `-- src
|           |-- env
|           |   `-- src
|           |-- regress
|           |-- sim
|           |-- sva
|           |-- tb
|           |   `-- assertion
|           `-- tc
|               `-- src
```

其中红色框部分为脚本生成验证环境相关：

- $PROJECT/trunk/digital_data/src： 项目相关的通用脚本，如 proj_tools.cshrc，project.cshrc 等
- $PROJECT/trunk/digital_data/src/verif： 验证环境相关的通用脚本，如 project_cfg.mk，DoRegress.py 等
- $PROJECT/trunk/digital_data/ver： 验证环境相关目录，其种
    - cmodel： 验证环境相关的 c 对应 dpi 封装代码及编译 so 可执行文件

- common： 验证 VIP 及 tcnt_base 存放路径
  - tcnt_base 为脚本扩展自 UVM 的生成的环境所有相关基类，包括：agent 所有组件相关基类、xaction 数据基类、scoreboard 代码等。
- formal： formal 相关环境
- fw： 项目相关 firmware 的 hex/bin 文件存放路径
- bt/it/st/ut： 不同验证登记模块的验证环境存放路径

对于每个生成验证环境，如上图的 ut/data_bypass，内部对应文件路径说明如下：

- agent： 存放根据配置文件自动生成的所有接口 agent
  - 每个 agent 按照 agent_name 存为为一个文件夹，脚本针对每个 agent 自动生成 agent_sequencer、agent_default_sequence、agent_driver、agent_monitor、agent_xaction、agent_cfg、agent 组件代码，以及相关的接口代码 agent_interface 和 parameter 包 agent_dec
  - 用户需要根据 agent 实际功能修改 driver 中的发包部分、monitor 中的组包部、xaction 中事务打包函数、以及 agent_cfg 可能需要的配置信息
- cfg： 存放脚本生成的模块 filelist，包括 rtl.f 和 tb.f，同时还存放用户可修改的 makefile 扩展文件：extern_declare_cfg.mk 和 extern_cfg.mk
- common： 存放以下内容：
  - 脚本自动生成模块公共事务类 env_name_common_xaction，用于转换接口事务送到 scoreboard
  - 脚本自动生成的功能覆盖率模板，功能覆盖率需要用户根据模块具体功能改写
  - 如果环境为参数化环境，脚本还会自动生成参数代码 env_name_dec
- env： 存放以下内容：
  - 自动生成的验证环境顶层 env，在 env 根据配件自动实例化所有 agent 和 rm、scb，并通过 fifo 将 agent、rm、scb 三者连接，三种之间的通信事务为 env_name_common_xaction
  - 自动生成的验证模型 rm，需要用户根据模块功能改写为实际的验证模型
  - 自动生成验证环境的配置文件 env_cfg，根据实际需要用户可能需要对配置内容进行填写
- regress： 存放回归脚本，当前版本环境暂时不支持生成回归脚本，需要用户自己放入
- sim： 用于仿真的目录，脚本自动生成对应 makefile，用户在这一目录下，参考《Makefile 用户手册》使用对应命令进行仿真
- sva： 此为空文件夹，用于存放模块相关断言代码，需要用户自行根据模块功能添加
- tb： 存放测试平台相关内容包括：
  - 脚本自动生成的 dut 在 testbench 中的实例化

- 脚本自动根据配置信息生成的 ENV 和 dut 连线代码
- 脚本自动生成的后仿读取 sdf 代码，这里需要用户根据实际后仿修改 sdf 文件路径
- 脚本自动生成的 testbench 顶层代码 top_tb

- tc： 存放测试激励相关内容，包括：
  - 脚本自动生成的测试 case 基类 tc_base，如有需要用户可以在此代码中修改仿真错误退出的次数，除此之外一般不建议用户修改此处代码，此代码自动设置所有 agent 的 default_sequence
  - 脚本自动生成的测试 sanity 代码 tc_sanity，并自动生成所有 default_sequence 的扩展过载类内容，用户可以根据此模板完善对应 sanity 用例并实现其他仿真 case
  - 脚本自动生成的一个小脚本 GenTc.py 脚本，用于辅助用户快速创建一个新 case 模板

4. ini 配置文件说明

脚本读取 ini 配置文件得到要生成环境的相关配置，然后生成对应环境及环境所需 agent 并自动例化和连线。
ini 配置文件包括两个部分内容

- 通用配置 section
- 模块接口 agent 配置 section

4.1. 通用配置 section

这一 section 主要指定生成验证环境对应路径、环境名称、dut 名称、dut filelist 路径、环境等级、环境参数等配置 option
PS： 这一 section 要求名称一定是 ENV_GENERAL，如果脚本读取 ini 文件后查找不到这一 section，则报错(ERROR::::there is no ENV_GENERAL section in ini file!)并退出，不生成任何验证环境

- prj_path：环境所在工程路径，可以是相对路径也可以是绝对路径，需要指定到 **$PROJECT/trunk/digital_data** 这一层次，建议使用绝对路径

PS： 脚本会识别指定路径下面是否有 scr/ver 等相关文件夹，如果不存在则自动创建对应文件夹并生成验证环境。

- author：模块验证负责人
- env_name：环境名称，常用 dut 模块名称

- env_level：环境层级，从上往下层级选择为 st/it/bt/ut，只能是这四个层级种的一个，指定其他则脚本报错(ERROR::::env_level(At) not in ['st', 'it', 'bt', 'ut'], please check the xxx_env_cfg.ini)并退出，不生成任何验证环境
- rtl_top_name：生成验证环境对应 DUT 的顶层 module name
- u_rtl_top_name：指定 DUT 在验证环境种的实例化名称
- rtl_list：指定 DUT 对应 filelist 所在路径
- env_parameter：生成环境所需参数指定，如果环境为参数化环境，则会存在这里的配置指定，否则则不需要指定，此 option 可以屏蔽掉或者不填写任何内容

PS：参数内容必须按照给定格式 {"params1 名称"：params1 默认值，"params2 名称"：params2 默认值，...}，如果填入格式错误，则生成环境不带任何参数并报 warning(WARN::::env_parameter = aaa is illegal, please check the cfg.ini!!!)

4.2. 模块接口 agent 配置 section

除了通用配置 section(ENV_GENERAL)外，脚本将 ini 配置文件中其他 section 都视为接口 agent 对应的配置 section，一个 section 对应一个 agent，section 名称即为 agent 的名称，每一个 section 均包含模式、实例化源、实例化个数、agent 内信号 list、agent 参数等配置 option

- agent_name：agent 的名称，需要修改成所需要 agent 的名字；
- agent_mode：agent 的结构，有两种选择：
  - master：生成 agent 在验证环境中默认配置 sequencer/driver/monitor 均被打开；
  - only_monitor：生成 agent 在验证环境中默认配置 sequencer/driver 被打开，只有 monitor 被打开；
- instance_by：指定当前 agent 在验证环境中由哪个 agent 实例化，存在两种实例化来源
  - 由自身实例化：这时候配置值一定是 self，脚本生成对应 agent 代码并在 env 中实例化调用；
  - 由其 agent 复用实例化：这时候配置值为其他 agent 名称，脚本不生产对应 agent 组件，值在 env 中实现 agent 的实例化创建并在 testbench 中根据 interface 信息进行连线
- instance_num：之前当前 agent 在环境中的实例化个数，如果是多次实例化，可以由两种指定方式；
  - 指定实例化个数：这时候配置值一定一个大于 0 的整数，环境在实例化时按照静态数组方式声明 agent 并在 testbench 中依次连线，如果此时配置值为 1，则认为单个实例化，这时候的声明不为数组。
  - 指定每个实例化的名称：这时候的{*}配置值是一个 string 列表{*}，如 \["new","old","org"\]，环境在实例化时按照 string 关联数组的方式声明 agent 并在 testbench 中依次连线。

- agent_interface_list：指定 agent 内的所有接口，每个接口包含 4 个信息：
  - 接口方向，可以配置 input、output、inout，方向为 rtl 内端口方向；
  - bit，此为脚本内部识别关键字
  - 位宽信息，可以使用 parameter 进行配置，如果为 1bit，可以不填；
  - 接口信号名称

PS ： 对于 agent_interface_list，根据 instance_by 不同会有不同的填写要求

1. 如果 instance_by 配置值为 self，这时候此 option 必须填写完整的内容
2. 如果 instance_by 配置值不为 self，并且想要复用的 agent 没有在 ini 中有配置描述(例如 VIP)，这时候此 option 也必须填写完整的内容
3. 如果 instance_by 配置值不为 self，并且想复用的 agent 在 ini 中有相关配置描述(即本身 agent 复用)，这时候 option 无需定义，脚本会使用对应的 agent 中的 agent_interface_list 中的配置内容

- dut_interface_list(0|1|2|...)：指定 agent 内部在 dut 对应 module 上声明的接口名称，每个接口同 agent_interface_list 一样包含同样的 4 个信息。**要求 dut_interface_list 中的信号位置与 agent_interface_list 中的信号位置描述必须一致，否则脚本连线会出错。**

同时，根据 instance_num 的配置不同，dut_interface_list 的个数及名称也不同，如果 instance_num 为 1，则此时只有一个 dut_interface_list(0)；否则，有多少个 instance_num，就需要多少份 dut_interface_list0|1|2...，并用后缀 0|1|2...描述分开，保证脚本能正确连线另外，如果没有指定 dut_interface_list，则连线时脚本均会复用 agent_interface_list 中信号来进行连线，如果指定 dut_interface_list 个数不够，脚本也会复用 dut_interface_list(0)来进行连线，然后报告对应 warning:
WARN:::this is no "dut_interface_list" or "dut_interface_list0" in {agent_name}, use the "agent_interface_list" as "dut_interface_list" 或 WARN:::this is no "dut_interface_list" in {agent_name}, use the "dut_interface_list0" as "dut_interface_list"

- parameter：指定当前 agent 是否为可参数化的 agent，parameter 的指定规则同 ENV_GENERAL.env_parameter；
- filelist_path：instance_by 配置值不为 self，并且想要复用的 agent 没有在 ini 中有配置描述(例如 VIP)，则需要指定这时候 agent 对应的 filelist 路径，否则生成验证环境的 tb.f 存在错误

如果按照 filelist_path 无法找到对应 agent 那么 agent 将会引用默认 filelist （../../../common/{_instanceby}/{_instanceby}.f）；

5. 生成环境内容说明

本章大致描述生成环境的部分内容

5.1. agent

对于每一个配置为 self 的 agent，脚本均会自动生成对应组件代码存放到 agent 文件夹下，生成的 agent 代码结构如图：

- xxx_agent.f： agent 对应的 filelist，指定 include src 中所有代码，并指定./xxx_agent_pkg.sv
- xxx_agent_pkg.sv： agent 对应 package
- src/xxx_agent_cfg： agent 的配置文件
- src/xxx_agent_dec： agent 的参数，如果在 ini 中指定 parameter，则 parameter 内容在此体现
- src/xxx_agent_default_sequence： agent 的 default sequence 文件
- src/xxx_agent_driver： agent 的 driver 文件
- src/xxx_agent_interface： agent 的 interface 文件
- src/xxx_agent_monitor： agent 的 monitor 文件
- src/xxx_agent_sequencer： agent 的 sequencer 文件
- src/xxx_agent： agent 的主体文件
- src/xxx_agent_xaction： agent 的 transaction 文件；

5.2. common

生成环境会存在一个通用的 xaction，用于存放环境的公共信息，默认用作 scoreboard 的输入载体.

同样，common 中还生成 fcov 用于描述功能覆盖率。

5.3. tb

在 tb 中，脚本生成顶层 testbench(top_tb.sv)，以及 top_tb.sv 中 include 的 dut 实例化代码，接口连线代码等：

- xx_connect.sv： 相应接口自动连线的宏声明代码
- dut_inst.sv： dut 在环境中的实例化代码
- gen_wave.sv： 备份用的由 verilog 生成波形的代码
- read_sdf.sv： 后仿使用的读取 sdf 代码
- top_tb.sv： testbench 顶层代码

脚本为了生成环境能够更快速地被上层验证环境调用，生成的接口连线均使用宏声明方式实现，这样子在上层环境的 testbench 中，只需要直接 include 相应宏并调用即可.

5.4. tc

tc 中存放了默认生成的 testcase
这里面带有一个脚本 GenTc.py，用于自动 copy 一个原有 tc 来生成一个新的 tc，并自动将新 tc 填入 tc filelist 和 tc package 中，脚本带有以下参数

- --tc_old：指定 copy tc 的来源
- --tc_new：指定新 tc 名称
- --author：指定新 tc 维护作者，不指定的话为 xxx.ini 中配置的 author
- --tc_list：指定新 tc 要放到哪一个 tc list 中，不指定的话为 tc.f
- --tc_pkg：指定新 tc 要放到哪一个 package 中，不指定的话为 tc_pkg.sv

例如

./GenTc.py --tc_old tc_sanity --tc_new tc_case1