UVM 入门和进阶实验 0

嗨，大咖们，在我们结束了 SV 的验证平台构建的学习之后，我们终于来到了 UVM 的世界。有没有一点兴奋呢？希望在你们短暂的兴奋过后不要紧接着就是“无从下手”。就像我们在课堂上学习到的，UVM 整个验证结构的包如果要仔细钻研起来，真的要花不少的功夫。如果我们真得先从“一”开始学习，那么要学习到 UVM 验证平台搭建的精髓恐怕在有限的时间内还是难免差强人意。那么怎么办呢？这也是路桑在书中谈到，我们不会同其它 UVM 培训一样，将 UVM 解构得七零八碎，仔细教同学们知识点，因为这样仍然很难让大家在培训结束时，对 UVM 建立起完整的理念，深刻理解其优点。

所以，路桑在这里再次提醒大家，我们一开始就走的是高举高打的路子。至少在大家学习了 SV 的验证平台打发，掌握了框架建立的套路之后，已经能够对接下来 UVM 世界的理解输送基本弹药了。所以，我们不会围绕着各个语法点、特性来逐个解释，而是仍然按照学习 SV 时候的核心流程，即：

- 如何搭建验证框架
- 验证组件之间的连接和通信
- 如何编写测试用例，继而完成复用和覆盖率收敛

熟悉吗?没错!思路是一样的。所以同学们需要跟随这些核心点，在已经从 SV 的森林中走出来之后，我们即将又要踏进 UVM 这一篇茂密的雨林，那里有很多未知的、神秘的、令人惊奇的、各式各样的特性和组件在等待着大家呢。但是不管走到哪里，无论是课上听讲，还是课下实验，当你有点困惑、有点手足无措的时候，再将我们接下来的知识地图即上面的“核心流程”默念一遍，想一想我们在 UVM 的世界中，究竟是走到哪里了?哪怕是迷失，那也只是短暂性的。

我们的 UVM 入门和进阶实验 0 还是同之前 SV 验证实验 0 思想一样让大家通过简单的实验要求，在轻松愉悦的气氛下，踏出 UVM 世界的第一步，从而掌握下面的基本概念和仿真操作：

- 懂得如何编译 UVM 代码
- 理解 SV 和 UVM 之间的关系
- 了解 UVM 验证顶层盒子与 SV 验证顶层盒子之间的联系
- 掌握启动 UVM 验证的必要步骤。

编译

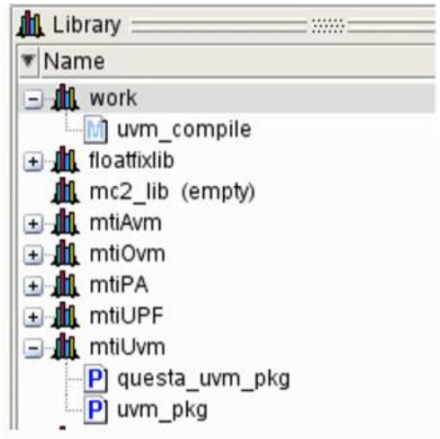
编译文件 uvm_compile.sv，待正常编译正常结束。在 work 库中仿真模块 uvm_compile, 在命令窗口中敲入"run -all"，可以观察到仿真输出语句：

恭喜你，已经将 UVM 包导入了进来。在文件中的这两句接下来请你记住，无论在什么地方，我们的验证顶层都将需要这两句话：

import uvm_pkg: :*;

`include "uvm_macros.svh"

这两句即代表着从预编译的 UVM 库(UVM 开源库+ Questa 的 UVM 定制部分库)，这一点可以在 Library view 中观察到：



在仿真开始后，你将可以观察到-些欢迎语，即“UVM-1.1d”的版本以及有哪些 EDA 行业大佬参与背书到其中了。如果你在使用更新的 Questasim 版本，那么默认情况下，你的 UVM 版本可能会显示"UVM-1.2" 。理论上呢，越新的版本使用起来在后期需要更新知识的成本最小，从 UVM-1.1d 到 UVM-1.2 还是有一些使用方法的变化， 不过暂时不会影响到同学们的正常使用。因为在我们接下来的实验部分的代码对于两个版本都是通用的，不会存在版本兼容的问题。

接下来，你可以看到我们使用了 UVM 的消息宏打印出来的消息，"Hell.." ，"Bye...", 那么这是不是意味着你已经"进入” UVM 的世界了呢?其实还不...接下来的部分路桑会给你解释，UVM 世界的入口究竟在哪里。

```
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(215) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.2
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(217) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO /vobs/soc_db-sys-prj/MCDF/uvm_basic_labs/lab0/sv_class_inst.sv(15) @ 0: reporter [SV_TOP] test started
# UVM_INFO /vobs/soc_db-sys-prj/MCDF/uvm_basic_labs/lab0/sv_class_inst.sv(9) @ 0: reporter [SV_TOP] SV TOP creating
# UVM_INFO /vobs/soc_db-sys-prj/MCDF/uvm_basic_labs/lab0/sv_class_inst.sv(17) @ 0: reporter [SV_TOP] test finished
```

SV 和 UVM 之间的关系

接下来，添加 sv_class_inst.sv 文件，编译，仿真，看看发生了什么？实际上这个实

验是 SV 模块实验环节的抽象，它只是在顶层 module 容器中要例化软件验证环境的顶层，即 SV class top.

在接下来的阶段，从打印出的信息可以看得出来，相当于从测试的开始，到验证环境的搭建，激励的发送，检查的执行等，最后又到了测试的结束。因此这是 SV 模块实验的“一”，即一生二，二生三，三生万物的那个"顶层”。

我们接下来再添加 uvm_class_inst.sv, 编译,仿真,看看发生了什么?从打印的信息来看，也是在模拟验证结构的创立，只不过这一次我们利用 了 UVM 的类 uvm_component 来定义了 top 类，继而在创建了这个"顶层”验证结构。那么这个顶层，算不算做是 UVM 环境呢？

```
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(215) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.2
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(217) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO /vobs/soc_db-sys-prj/HCDF/uvm_basic_labs/lab0/uvm_class_inst.sv(18) @ 0: reporter [UVM_TOP] test started
# UVM_INFO /vobs/soc_db-sys-prj/HCDF/uvm_basic_labs/lab0/uvm_class_inst.sv(11) @ 0: t [UVM_TOP] SV TOP creating
# UVM_INFO /vobs/soc_db-sys-prj/HCDF/uvm_basic_labs/lab0/uvm_class_inst.sv(20) @ 0: reporter [UVM_TOP] test finished
```

先不着急下结论，同学们做到这里，请注意，我们所谓的 UVM 验证环境指的首先是提供一个 UVM 的“容器”，即接下来所有的 UVM 对象都会放置在其中，这样也可以成为 UVM 的顶层,这就类似于之前 SV 模块实验中的顶层容器 test 一样。

UVM 验证顶层与 SV 验证顶层的对比

我们再来看另外-个文件 uvm_test_ inst.sv, 编译，仿真，再看看发生了什么？似乎打印的信息比较多了,如果我们将其划分的，可以分为两个部分：

- 测试运行时的消息，这一些消息与之前的几个例子没有太多差别
- 仿真结束后的总结消息，这一些消息是之前例子没有的，至于其中的具体含义，我们可以在后续学习了更多的 UVM 知识后，带领大家解读。

```
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(215) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.2
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(217) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO /vobs/soc_db-sys-prj/HCDF/uvm_basic_labs/lab0/uvm_test_inst.sv(22) @ 0: reporter [UVM_TOP] test started
# UVM_INFO /vobs/soc_db-sys-prj/HCDF/uvm_basic_labs/lab0/uvm_test_inst.sv(11) @ 0: uvm_test_top [UVM_TOP] SV TOP creating
# UVM_INFO @ 0: reporter [RNTST] Running test top...
# UVM_INFO /vobs/soc_db-sys-prj/HCDF/uvm_basic_labs/lab0/uvm_test_inst.sv(15) @ 0: uvm_test_top [UVM_TOP] test is running
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 0: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    7
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVM_TOP]     3
# ** Note: $finish    : /nfs/za/proj/inway/eda/mentor/questasim/10.2e_3/questasim/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 0 ns  Iteration: 216  Instance: /uvm_test_inst
```

启动 UVM 验证的必要步骤

这个例子即 uvm_test_inst 显然与 uvm_class_inst 带给人的感官不同，如果路桑问你哪个可以作为 UVM 验证顶层容器的话，你估计会选择 uvm test inst 吧。没错,我们之所以这样选择，是基于下面几处代码的不同以及它们所带来的影响所造成的:

- 只有继承于 uvm test 的类，才有"资格"可以作为 UVM 验证环境的顶层。（暂时不用问路桑为什么，但是请结合之前 SV 模块的实验常识，是不是觉得这个规定有它的考量在呢？）
- 创建顶层验证环境，有且只能依赖于 run_test("UVM_TEST_NAME")来传递，或者稍后你可以学习到可以通过仿真参数传递，而不是通过构建函数 new( ) 来实验的。尽管 new( ) 可以创建一个对象，但是不能做与顶层验证环境相关的其它工作。(这一点，我们也暂时了解到这里，因为我们还只是刚刚买票“入园参观”，对于其 UVM 世界的秘密我们还有很多很多需要学习。）

所以，同学们，你已经学习到进入 UVM 世界的"随意门”。利用路桑给你的套路，在接下来，无论我们要讲 UVM 关于工厂和其它的特性，还是 UVM 验证组件的创建和连接，又或者其它的示例，我们都将利用这一“随意门”来进入 UVM 世界。至于这样做有什么神奇的功效，路桑会在 UVM 组件的介绍课堂中，跟大家进一步探讨 uvm_test 类的作用。