UVM 入门和进阶实验 1

当带领大家进入一个大的景观园区，为了防止迷路，“小朋友” 们最好还是不要乱跑，容易迷路的。我们接下来实验 1 的环节，将带领大家了解下面主要几个部分:

●工厂的注册、创建和覆盖机制

●域自动化以及 uvm object 的常用方法

●phase 机制

●config 机制

●消息管理

是不是觉得已经很多了呢?坚持，坚持一下！因为只要趟过这一关，接下来的实验 2 就要进入我们本次 UVM 模块学习的主线了，现在的铺垫极其重要，否则在进入主线学习的时候，可能会“消化不良”呢！路桑尽量让每一个特性在以下实验中独立开来，让大家在做每一个实验要求的时候，可以不被其它知识点所影响。所以，实验 1 的代码还是相对简单的，这有一点像 SV 模块的实验 0-3 的部分。不过接下来的实验 2 由于将要进入主线学习 UVM 的结构，这里会提醒大家，代码量可能会"暴增” ，希望能在实验 2 的时候多花点时间预习一下呢。另外，推荐大家在不懂的地方，可以查看 UVM 的类手册(class_reference)或者红宝书。我们所有培训模块的主线，都是从红宝书移植过来的哦。

在接下来进行各项实验之前，我们需要掌握除了使用实验 0 中的 run test("UVM_TESTNAME”)之外，还有什么方法可以灵活地选择创建哪一个 UVM 测试用例，即我们可以在仿真时通过传递仿真选项+UVM_TESTNAME= my test 来实现。这一点使得既避免了多次修改代码和编译，也使得我们可以通过仿真时参数来控制 UVM 的架构创建。

工厂的注册、创建和覆盖机制

我们需要理解为什么需要注册，而在 UVM 世界中的注册一共有哪几种方式呢？只有两种用来注册的宏，请你记住它们:

- `uvm_object_utils(T)
- `uvm_component_utils(T)

也就是说，凡是你继承的 uvm object 类或者 uvm component 类(不出意外的话，所有 UVM 世界中的类都是它们其中的一种)，你都需要在它们的代码中选择一种宏并且声明， 其差别只是需要你辨别该类是 uvm_object 的子类还是 uvm_component 的子类。

打开实验文件 factory_mechanism.sv，编译，在仿真时敲入命令：

vsim -novopt -classdebug +UVM_TESTNAME=object_create work.factory_mechanism

上述命令指的是我们将运行测试 object_create，注意观察代码中，是否所有的 UVM 类都已经用上述注册宏的一种来声明了呢？没错，所以这是我们在本节实验中学习到的第一个“套路”。接下来，我们先不解释为什么需要“注册宏”，而是进入创建和覆盖这两部分的实验要求，待你完成实验要求之后，我们可以再来回顾——为什么需要 "注册宏”。

- 请按照实验 1.1 和 1.2 的具体要求，实现代码，并且运行 UVM 测试 object_create 和 component_create，观察打印信息，检查 t2/t3/t4 和 u2/u3/u4 是否被创建。
- 请按照实验 1.3 和 1.4 的具体要求，实现代码，并且运行 UVM 测试 object_override 和 component_override，观察打印信息，检查 t2/t3/t4 和 u2/u3/u4 的类型是否被替换，再考虑为什么 t1 和 u1 没有被类型覆盖呢?

在完成了关于“创建”UVM 对象和“覆盖”UVM 类之后，我们还可以再来做以下尝

试，帮助同学进一步认识“注册宏”的作用：

- 请移除 trans 类型的`uvm_object_utils()注册宏说明，再进行编译仿真，观察是否有编译或者运行错误？请思考大致为什么?
- 同样地，请移除 unit 类型的 uvm_component_utils()注册宏说明，再进行编译仿真，观察是否有编译或者运行错误？请思考大致为什么?

域自动化和 uvm_object 的常用方法

到了域自动化的环节，我们需要了解常见的域声明相关的宏，以及在声明了之后，uvm_object 类所具备的常见方法。在这个实验环节中，我们主要就一些常见标量的域声明宏做以练习，继而再对 uvm_object 提供的几种常见方法和其回调函数(callback) 加以掌握。接下来，请在文件 uvm_object_methos.sv 中完成以下实验要求：

- 实验 2.1，请学习使用域自动化的宏方法。可参考红宝书表 10.2 和表 10.3。
- 实验 2.2，请学习 uvm object:compare( ) 方法。
- 实验 2.3，请尝试掌握 uvm pkg.中常见的一些全局控制对象，例如 uvm_default_comparer，该对象可参考 uvm_comparer 类提供的方法。具体还包括哪些全局对象，可以参考红宝书图 10.7 uvm .pkg 的全局对象。
- 实验 2.4，请学习自定义 uvm object 的一些回调函数，例如 do. compare()，并且理解预定义函数 compare()与 do. compare()的调用顺序和关系。
- 实验 2.5、2.6，请学习 uvm_object:print()及 uvm_object::copy()函数，再结合之前的 compare()函数，理解域自动化的意义以及它带来的便捷性。

注意，此时应在仿真器中输入指令：

vsim -novopt -classdebug +UVM_TESTNAME=object_methods_test work.object_methods

Phase 机制

这一部分实验可以参照红宝书 10.5 节 phase 机制。phase 机制使得验证环境从组建、到连接、再到执行得以分阶段执行，按照层次结构和 phase 顺序严格执行，继而避免一些依赖关系， 也使得 UVM 用户可以正确地将不同的代码放置到不同的 phase 块中。请在文件 phase_order.sv 中完成以下实验要求:

- 实验 3.1，请参考 comp1 类定义的几个主要 phase 方法，也分别对类 comp2 和 comp3 定义相应的 phase 方法。在完成后，可运行 phase_order_test，观察按照层次结构，phase_order.test、comp1、comp2 和 comp3 在不同 phase 阶段所执行 phase 的先后顺序。
- 实验 3.2，请完成 phase_order_test 的另外 2 个细分的 phase，即 reset_phase 和 main_ phase， 可以参考其 run_ phase 的实习方法，并且也分别耗时 1us。然后再运行 phase_order_test，再次观察各个组件在执行不同 phase 时的顺序，对于 phase_order_test 而言，由于引入了 run_phase 并行开始执行的 reset_phase 和 main_phase， 这会使得仿真最终在什么时间结束?为什么在此时结束呢?

在仿真器中输入指令：

vsim -novopt -classdebug +UVM_TESTNAME=phase_order_test work.phase_order

config 机制

这一部分实验可以参照红宝书 10.6 节 config 机制。我们在本次实验中，也将分别完成以下几种传递方式：

- 接口传递
- 单一变量传递
- 对象传递

请在文件 uvm_config.sv 中完成以下实验要求：

- 实验 4.1，请完成接口从 uvm_config 模块到验证环境中的传递，使得 c1 和 c2 可以得到接口，并且检查接口是否最终得到。
- 实验 4.2，请完成配置对象 config.obj 从 uvm_config.test 到 c1 和 c2 的传递。
- 实验 4.3，请完成在顶层 uvm_config_test 对 c1.var1 和 c2.var2 的变量设置。
- 实验 4.4，请分别进一步思考如下问题：
  - 。接口传递与 run_test()之间是否存在顺序?
  - 。在 uvm_config_test::build_ phase()中， 如果将 c1 的例化提前到刚进入 build_phase()中时，而将后续 config_db 传递的操作放置于其后，是否可行？为什么?
  - 。通过上述两个问题，你认为 config.db 在使用时，需要注意什么地方?

在仿真器中输入指令：

vsim -novopt -classdebug +UVM_TESTNAME=uvm_config_test work.uvm_config

消息管理

请同时参考红宝书 10.7 节消息管理，在已经初步掌握消息宏`uvm_info()、`uvm_warning()、`uvm_error()以及 `uvm_fatal()的同时， 也需要学习，如何在后期验证环境较为稳定的时候，减少不必要消息的打印，以此来协助提高仿真速度。请在文件 uvm_message.sv 中完成以下要求:

- 实验 5.1，请使用消息过滤方法 set_report_verbosity_level_hier()在 uvm_message_test:build_phase()中屏蔽所有层次的消息，也就是不允许有任何 uvm_message_test 及其以下组件的消息在仿真时打印出来。
- 实验 5.2，请先注释实验 5.1 的代码，继而转用 set_report_id_verbosity_level_hier(来过滤以下 ID 的消息，即“BUILD”、"CREATE"和"RUN"。待方法使用之后，请与实验 5.1 打印消息的结果进行对比，检查两种方式打印结果一致。
- 实验 5.3，也许你已经发现，config_obj 的"CREATE”消息以及 uvm_message 的"TOPTB”消息依然被打印了出来，请思考这是为什么?是否之前 5.1 和 5.2 提供的方法也可以屏蔽这些消息?如果不可以，那么是否有其它办法可以屏蔽这些消息？请尝试使用 uvm_root::get()来获取最顶层的（即 uvm_message_ test 的顶层）来控制过滤"CREATE”和"TOPTB" 的消息。

在仿真器中输入指令：

vsim -novopt -classdebug +UVM_TESTNAME=uvm_message_test work.uvm_message