MCDF 实验 4

首先路桑要恭喜你，你已经迈过了 SV 小白的阶段。在实验 3 中，你可以通过一个初具规模的 MCDT 的验证环境理解验证的一些要素。在接下来进入实验 4 之前，你头脑中需要再复习这些概念。**第一，验证环境按照隔离的观念，应该分为硬件 DUT，软件验证环境，和处于信号媒介的接口 interface。第二，对于软件验证环境，它需要经历建立阶段(build)，连接阶段(connect)，产生激励阶段(generate) 和发送激励阶段(transfer)。**只有当所有的激励发送完毕并且比较完全之后，才可以结束该测试。在课堂上，你跟着路桑回顾了实验 3 的代码结构之后，你就可以进入实验 4 了。

实验 4 的环境是完善的，这对你而言，是个好消息。不妨再像个刘姥姥进大观园一样，看看实验 4 的代码与实验 3 相比，是不是又更上一层楼了呢？这其中的原因主要有两方面，一方面是因为我们从实验 4 开始在验证**更大的子系统**，即 MCDF。与 MCDT 相比，MCDF 主要添加了寄存器控制和状态显示功能，同时也添加了一个重要的数据整形功能，因此，你会发现实验 4 的验证文件多了。另外一方面，代码之所以增多是为了让整个验证环境的各个组件之间相互独立，功能清晰，**你可以发现不同的 package 之间的功能是独立的**，同一个 package 中的各个验证组件的功能也是独立的。那么不同组件之间的同步和通信依靠什么呢？没错，那就是我们已经学习到的 event 和 mailbox。

怎么开始试验 4 呢？大胆地编译所有的文件，然后给出仿真命令就可以了：

```
Vsim -novopt -classdebug -solvefaildebug -sv_seed 0
+TESTNAME=mcdf_data_consistence_basic_test -
|mcdf_data_consistence_basic_tast.log work.tb
```

来看看上面主要的仿真命令项：

-classdebug，这是为了提供更多的 SV 类调试功能

-solvefaildebug，这是为了在 SV 随机化失败之后有更多的信息提供出来

-sv_seed 0，暂时给固定的随机种子 0

+TESTNAME=mcdf_data_consistence_basic_test，这是指定仿真选择的调试

-| mcdf_data_consistence_basic_test_sim.log，这是让仿真的记录保存在特定的测试文件名称中

那么在开始测试并且最终结束之后，在你的项目目录下会有两个文件产生 mcdf_data_consistence_ basic_test_sim.log 会保存所有的仿真信息，而 mcdf_data_consistence_ basic_test_check.log 则只会保存 mcdf_checker 做数据比较和最终比较报告的信息。需要注意的是，在你测试的过程中，如果测试发生了错误则会立即停止下来，你可以根据当前时刻的错误报告定位出数据比较错误的时间点，再去核对波形，如果经过确认确实是硬件的行为有问题，那么路桑需要恭喜你，**你发现了一个 BUG，赶快举手来跟路桑领红包吧！**

那么验证环境这么完善，是不是你不需要做实验 4 了呢？ NONON...我们实验 4 的重点是要求同学们在本次试验中领会，**怎么样才是一个完整的验证计划和实施。**还记不记得我们课程中谈到的，什么是验证计划的核心要素点？那就是围绕着设计的功能点罗列出需要展开测试的功能点，以及如何展开测试，并且指明测试的验收标准是什么。**我们实验 4 中主要以测试是否通过为原则，而在实验 5，同学们将进一步掌握代码覆盖率和功能覆盖率的验收标准。**

要求 1.1（必做）

从路桑带领大家勾画出实验 3 的验证环境框图，想必你已经对验证框图的重要性有体会了吧？没错，从验证框图可以更好地在理解验证环境的组件和组件之间的通信连接情况。无论对于你接下来着手构建环境，还是将它作为你验证代码的一个形象说明，它都比代码更直接地形容整个验证环境。那么接下来，请你仍然按照我们之前课堂中所讲的，准备好章 A4 纸、铅笔和橡皮，来**理解实验 4 中的验证环境。**不用着急，从我们这次实验一开始所讲的，在大脑中回顾一下，验证环境的核心要素是什么，然后开始画一下你理解的验证结构吧。

要求 1.2（必做）

接下来，路桑就会指定你需要在 mcdf_pkg.sv 中参考以后测试类 mcdf_data_consistence_ basic_test 而去创建其它的测试类名，**它们对应的测试功能点，以及测试标准是什么。**在你按照要求，实现了所有的测试之后，记得将每个测试都至少跑一遍，如果你跑的过程中发现 mcdf_checker 报错，那么看看是不是发现了什么设计问题，如果你有信心它是一个漏洞，那么就赶快联系助教吧，助教会帮你把它记录到漏洞跟踪表格中，而后带设计修改确认漏洞修复之后，他会发布下一版的设计，到时候也会及时在班级群众通知大家，大家记得要及时下载更新后的设计版本哦。理论上，如果你对同一个测试，每次使用不同的随机种子即“-sv_seed RANDOM”，那么你跑得次数越多，就越有机会发现新的“惊喜”，而至于你需要跑多少遍，或者每个测试中需要发送多少次数据包（发送的越多越好，那么究竟要发送多少次才可以停下来呢？标准是什么？）？我们将在实验 5 中为同学揭晓覆盖率的验证量化方式。从目前的测试来看，那就尽管让仿真跑一下去吧，你完全可以去吃一顿饭，在吃饭的功夫中让测试不断发包，而目前测试的标准是一旦数据包比较错误就会停止，到时候就可以在“案发”的第一时间去探索真相了。

测试功能点	测试内容	测试通过标准	测试类名
寄存器读写测试	所有控制寄存器的读写测试所有状态寄存器的读写测试。	读写值是否正确	mcdf_reg_write_read_test
寄存器稳定性测试	非法地址读写，对控制寄存器的保望域进行读写，对状态寄存器进行写操作。	通过写入和读出，确定寄存器的值是预期值，而不是紊乱值，同时非法寄存器操作也不能影响 MCDF 的整体功能	Mcdf_reg_illegal_access_test
数据通道开关测试	对每一个数据通道对应的控制寄存器域 en 配置为 0，在关闭状态下测试数据写入是否通过	在数据通道关闭情况下，数据无法写入，同时 ready 信号应该保持为低，表示不接收数据，但又能使得数据不。被丢失，因此数据只会停留在数据通道端口	Mcdf_channel_disable_test
优先级测试	将不同数据通道配置为相同或者不同的优先级，在数据通道使能的情况下进行测试	如果优先级相同，那么 arbiter 应该采取轮询机制从各个通道接收数据，如果优先级不同，那么 arbiter 应该先接收高优先级通道的数据，同时，最终所有的数据都应该从 MCDF 发送出来	Mcdf_arbiter_priorit_test
下行从端低带宽测试	将 MCDF 下行数据接收端设置为小存储量，低带宽的类型，由此使得在由 formatter 发送出数据之后，下行从。端有更多的机会延迟 grant 信号的置位，用来模拟真实. 场景	在 req 拉高之后，grant 应该在至少两个时钟周期以后拉高，以此来模拟下行从端数据余量不足的情况。当这种激励时序发生 10 次之后，可以停止测试。	Mcdf_formatter_grant_test

要求 1.3（选做）

也许你在阅读要求 1.2 的测试标准的时候，心生疑惑，是否路桑目前提供的 mcdf checker 有能力可以完成所有的这些通过标准检查呢？实际上，我们还有一些地方 需要填补，你能找到 mcdf checker 还需要单独实现哪些用来检查的功能吗？这里，路桑给你一点提示吧：

●寄存器的读写， 无论是否非法，都可以参考 mcdf_data_consistence_basic_test 中比较数据的函数来做比对。

●数据通道的开关，在数据通道关闭的情况下，mcdf checker 应该不会收到输入端或者输出端的监测数据(想想为什么？)，因此不应该出现数据比较的信息。那么在这种情况下，怎么来判断这个测试通过呢？最简单的是，这个测试在最后的报告阶段总结中，error 信息的统计数量为 0，但这仍然不够。因为我们还要检查-个简单的时序，即当 slave channel 的被关闭时，当 valid 拉高时，ready 信号不应该出现拉高的情况，否则设计是有问题的，因为数据可能没有被真正写入到 FIFO，或者 slave channel 此时并没有被真正关闭。那么如何实现这一检查呢？ 路桑给的建议是，在 mcdf_intf 中， 将 en 信号监测到，将其传入 mcdf_checker，同时也将 3 个 chnl intf 传入 mcdf checker。这样，mcdf checker 可以观测到所需的 valid、ready 和 en 信号，来完成这个检查。不妨给个 task 名称来保持代码统一吧， mcdf checker:do. channel disable check()。

●对于优先级的检查，mcdf checker 依然无法从 mcdf refmod 的预测数据中获取数据包前后的信息，也就说是，mcdf checker 依然无法判断最终从 mcdf 送出的数据是否符合了优先级的数据包发送顺序？那么，不妨依然将需要观察的 arbiter 信号在 mcdf intf 中声明并且观测，再新建一个 task 来保证优先级的检查吧，

mcdf checker:do. arbiter. priority .check0.

●那么发包的长度，mcdf checker 是否已经可以保证检查到了呢？是的，已经可以保证这点了，那么你是否能够从代码中看出，路桑是如何保证 formatter 最终发送的各个通道从端的数据包长度的检查的呢？谈谈你的看法。

●至于最后下行从端低带宽的测试，需要同学使用好如何对下行从端的配置约束，使其可以实现在低带宽数据消耗的情况下，自身的缓存量逐渐减少，而频繁在 formatter request 信号拉高时而延迟 grant 信号的拉高，以此来模拟真实情况。那么你还需要考虑的是，如何来观测 request 与 grant 之间超过两个时钟周期的时序延迟呢？你可以考虑由 test 直接使用 fmt intf 中的信号来做时序观察和计数，当满足 10 次之后，即可以停止数据的发送。如何停止数据的发送呢？可以考虑使用 fork-join/join. any/join. none 和 disablefork 等用法。

不过即使你最终没有完成要求 1.3，也不必为此难过和叹气，你现在就跟登山的探路者一样，回过头去再看实验 1、实验 2 和实验 3，是不是恍然间产生一种感觉， 那真是- -山又比一山高， 不识庐山真面目，只缘身在此山中。所以，只管尽力去实现你的代码，要求 1.3 即便在验收的时候遗憾没有做完，你也已经做完实验 4 的必选部分了呢，距离最后一个实验 5 只是一步之遥，而路桑会在实验 5 的时候给出实验 4 的参考代码，到时候你再看看，你跟路桑的思路，是不是你的代码更加飘逸更有创造性呢？一定是你的！因为无知者无畏嘛！