UVM 入门和进阶实验 3

同学们进入了实验 3，在跟路桑一起学习了 TLM 通信、同步通信原件以后，我们又将会在之前实验 2 已经完成的验证环境结构的基础上，继续改造。在实验 3 中，我们将 SV 环境移植到 UVM 的重点将主要在以下几个方面：

- TLM 单向通信端口和多向通信端口的使用。
- TLM 的通信管道。
- UVM 的回调类型 uvm_callback。
- UVM 的一些仿真控制函数。

TLIM 单向通信和多向通信

在之前的 monitor 到 checker 的通信，以及 checker 与 reference model 之间的通信，都是**通过 mailbox 以及在上层进行其句柄的传递实现的。**我们在接下来的实验要求中，需要大家使用 TLM 端口进行通信，做逐步的通信元素和方法的替换。同时，路桑在代码中也做了实验要求的注释，方便大家再指定的地方按照要求实现代码。

如图 1 所示，大家可以发现，我们将会在 monitor、checker 和 reference model 上面添加若干 TLM 的通信端口，而其端口类型我们在实验代码中也有具体要求。接下来请逐步按照下列要求完成：

1. 请将在 **monitor 中的用来与 checker 中的 mailbox 通信的 mon_mb 句柄**替换为对应的 uvm_blocking_put_port 类型。
2. 在 checker 中声明与 monitor 通信的 import 端口类型，以及与 reference model 通信的 import 端口类型。具体类型可以参考代码中的注释，需要注意的是，由于 checker 与多个 monitor 以及 reference_model 通信，是典型的多方

向通信类型，因此，我们需要使用多端口通信的宏声明方法，请参考红宝书 12.2.3 的实例。在使用了宏声明端口类型之后，再在 checker 中声明其句柄，并且完成例化。

3. 根据声明的 import 端口类型，请分别实现其对应的方法。
4. 如图 1 所示，请继续在 mcdf_refmod 中声明用来与 mcdf_checker 中的 import 连接的端口，并且完成例化，同时注意其端口类型的区别。关于端口类型，可以参考红宝书表 12.1。在完成声明和例化之后，请继续将原来的 mailbox 句柄调用方法的方式，改为用 TLM 端口呼叫方法的方式。
5. 请在 mcdf_env 的 connect_phase()阶段，完成 monitor 的 TLM port 与 mcdf_checker TLM import 的连接。
6. 请在 mcdf_checker 的 connect_phase()阶段，完成 mcdf_refmode 的 TLM port 与 mcdf_checker 的 TLM import 的连接。

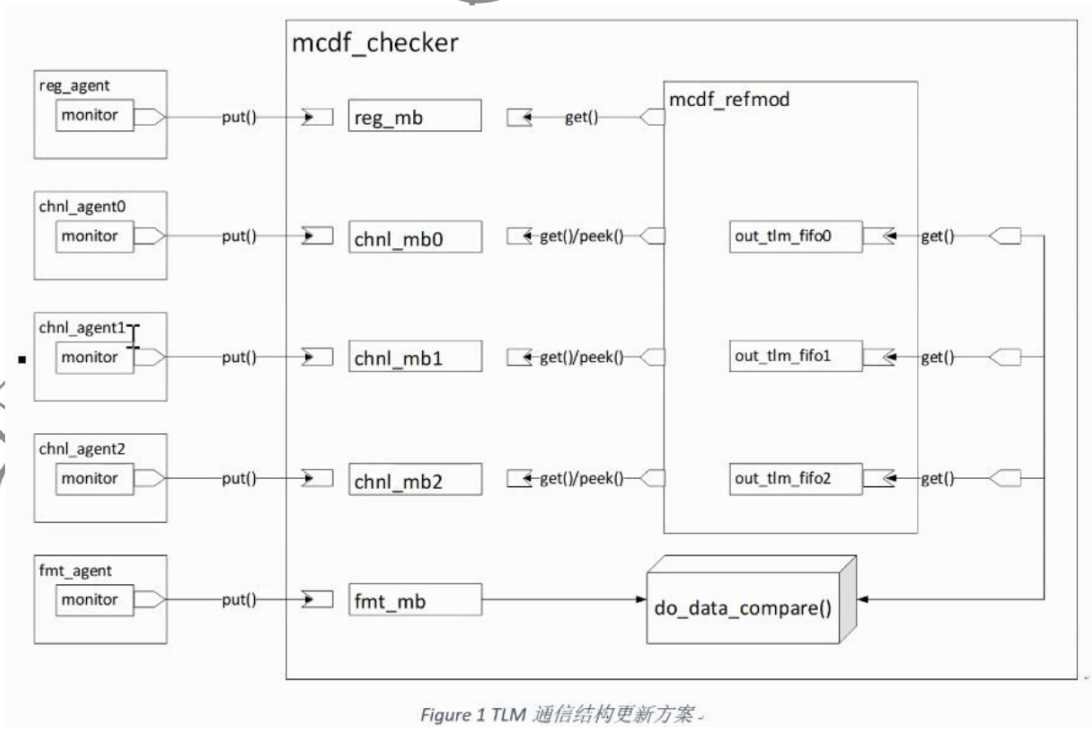

Figure 1 TLM 通信结构更新方案

TLM 通信管道

在完成了上述实验之后，你可能会抱怨，看起来工作量增加不少了呢！怎么会说，TLM 通信有它的好处呢？那路桑再阐述几个 TLM 通信的优点:

- 通信函数可以定制化，例如你可以定制 put()/get()/peek()的内容和参数，这其实比 mailbox 的通信更加灵活。
- 将组件实现了完全的隔离，可以参考红宝书图 12.4，因为只有通过层次化的 TLM 端口连接，我们就可以很好地避免直接将不同层次的数据缓存对象的句柄进行“空中传递”。而 TLM 端口按照层次的连接,虽然看起来有点繁复，但也正因为这一点，可以使得组件之间保持很好的独立性呢。

那么有没有既可以使用 TLM 端口，又不用像上述实验需要自己实现具体的 get()/peek()/put()方法呢？当然有啦！依然可以参考红宝书 12.3.1 节，关于 uvm_tlm_fifo 类的使用。请按照以下要求，完成本实验：

1. 将原本在 mcdf_refmod 中的 out_mb 替换为 uvm_tlm_fifo 类型，并且完成例化，以及对应的变量名替换。
2. 将原本在 mcdf_checker 中的 exp_mbs[3]的邮箱句柄数组，替换为 uvm_blocking_get_port 类型句柄数组，并且做相应的例化以及变量名替换。
3. 在 mcdf_checker 中，完成在 mcdf_checker 中的 TLM port 端口到 mcdf_refmod 中的 uvm_tlm_fifo 自带的 blocking_get_export 端口的连接。

在完成这个实验环节之后，请开始编译原有的仿真测试，进行仿真，检查仿真结果是否与实验 2 的结果保持一致。另外，请在思考，上述两个实验环节中，针对一般的数据存储和 TLM 端口连接，那一种方式更为简便？

UVM 回调类

接下来，我们将练习 uvm_callback 的定义、链接和使用方式。由此，我们可以将原有的 mcdf_data_consistence_basic_test 和 mcdf_full_random_test 的类实现方式（即类继承方式）修改为回调函数的实现方式，帮助同学们认识，完成类的复用除了可以使用继承，还可以使用回调函数。请按照以下要求实现代码：

1. 请在路桑给的 uvm_callback 类中，预先定义需要的几个虚方法。
2. 请使用 callback 对应的宏，完成目标 uvm_test 类型与目标 uvm_callback 类型的关联。
3. 请继续在目标 uvm_test 类型指定的方法中，完成 uvm_callback 的方法回调指定。
4. 请分别完成 uvm_callback 和对应 test 类的定义：
    a. cb_mcdf_data_consistence_basic 和 cb_mcdf_data_consistence_basic_test
    b. cb_mcdf_full_random 和 cb_mcdf_full_random_test

在完成上述代码后，可以指定经过修改的 test 类 cb_mcdf_data_consistence_basic_test 和 cb_mcdf_full_random_test， 并且与其之前对应的两个类 mcdf_data_consistence_basic_test 和 mcdf_full_random_test 做比较，由此加深理解——uvm_callback 的方式是如何协助完成类的复用的。同时也可以对比之前 SV 模块的 callback 方法实现，体会 uvm_callback 的优势在什么地方？

UVM 仿真控制方法

我们也可以回顾实验 1 对 uvm_root 类的应用，学习更多关于 uvm_root 类的方法。请按照以下实验要求实现代码：

1. 请在 mcdf_base_test 类中添加新的 phase 函数 end_of_elaboration_phase()。同时利用 uvm_root 类来将信息的冗余度设置为 UVM_ HIGH，以此来允许更多低级别的信息打印出来。另外，请 uvm_root::set_report_max_quit_count()函数来设置仿真退出的最大数值，即如果 uvm_error 数量超过其指定值，那么仿真会退出。该变量默认数值为-1，表示仿真不会伴随 uvm_error 退出。
2. 请利用 uvm_root::set_timeout()设置仿真的最大时间长度，同时由于此功能的生效，可以清除原有方法 do.watchdog()的定义和调用。

哇哦，到这里路桑要恭喜你呢，我们可是胜利在望啊！下一 次实验，我们将会结合对于 sequence item, sequence, sequencer 和 driver 的理解，来将目前验证结构中的 generator 和 driver 之间的结构关系，修改为 UVM 的结构和序列，一起期待它吧！