# MCDF实验2

实验2的部分我们将主要回顾之前接口、仿真结束、类和包的使用。在这一个试验中，同学们将逐渐从使用硬件盒子验证过渡到使用接口和软件盒子（class）来验证设计。而这一个实验之所以重要也是因为它是硬件验证方式与软件验证方式之间的过渡，同时作为验证环境的启蒙，同学们在本实验的最后一个小实验中也能够初步体会到类的继承和层次包含关系，而这些都将作为日后学习高阶UVM知识的重要基础。好了，接下来让我们进入这次的4个小实验部分。

请下载lab2中的四个实验文件tb1.sv、tb2.sv、tb3.sv和tb4.sv，同时不要忘记检查最新的DUT MCDT的版本是否是最新的，这一点也很重要。。

## 接口的使用

tb1.sv的代码部分承接的是上一次实验的最后代码部分，不过最显著的差别在于，我们将验证组件和DUT之间通过接口来连接，所以验证组件chnl\_initiator的端口变得非常“干净”，即chnl\_intf intf。在使用接口之前我们需要定义接口chnl\_intf和内部的端口，同时我们也声明了一个始终块（clocking block），它的功能是为了消除可能存在的竞争（race）问题，确保时钟驱动数据之间有一定的延迟，以便于DUT顺利采样。

因此更新后的代码，同学们可以发现例化的实例包括了只产生数据的channel generator，只负责发送数据的channel initiator以及作为验证组件和DUT之间的接口chnl\_intf。而例化之后，在initial块中需要通过调用组件的方法完成初始化、名字设置和空闲周期的设置。这里初始化是为了设置ID，名字设置时为了稍后打印时容易区分各个组件，而空闲周期的设置则关系到我们接下来的实验要求：

### 要求1.1

首先观察波形，同学们可以发现channel initiator发送的数据例如valid和data与时钟clk均在同一个变化沿，没有任何延迟。同我们课程中所讲到的，这种0延迟的数据发送不利于波形查看和阅读，因此请同学们在已有代码的基础上使用intf.ck的方式来做数据举动，并且再观察波形，查看驱动的数据与时钟上升沿的延迟是多少。

### 要求1.2

为了更好地控制相邻数据之间的空闲间隔，我们又引入了一个变量idle\_cycles，它表示相邻有效数据之间的间隔。已有代码会使得有效数据之间保持固定的一个空闲周期，我们需要大家使用idle\_cycles变量，来灵活控制有效数据之间的空闲周期。通过这个方法，在tb的initial块中我们通过方法set\_idle\_cycles()使得三个channel initiator的空闲周期变为0，即可以实现有效数据的连续发送。

## 仿真的结束

tb2.sv中同学们需要课外学习fork-join的基本功能和使用方法，了解它的并行运行特性，以此来实现三个chnl\_initiator同时发送数据的要求。同时我们又将不同的test也组装到task中，以此来区分不同的测试内容，这是由于每一个测试任务的测试目的和要求都不相同，具体要求可以在代码中查找。tb2.sv需要首先移植tb1.sv的要求内容，接下来再完成新的实验要求。

### 要求2.1

可以参考task basic\_test()，来完成burst\_test()。它的要求是使得每个chnl\_initiator的idle\_cycles设置为0，同时发送500个数据，最后结束测试。

### 要求2.2

参考task basic\_test()来完成task fifo\_full\_test()。它的要求是无论采取什么数值的idle\_cycles，也无论发送多少个数据，只要各个chnl\_initiator的不停发送使得对应的channel缓存变为满标志（ready拉低），那么可以在三个channel都拉低过ready时（不必要同时拉低，先后拉低即可），便可以立即结束测试。

## 类的例化和类的成员

在这一部分，我们便将之前用来封装验证功能的硬件盒子（module）中的数据和内容移植到软件盒子（class）中来，同学们可以通过前后代码的相同点和不同点来比较实用类的时候，需要注意什么地方，同时也可以基本掌握类的例化，类的成员变量访问权限以及类的成员方法如何定义和使用。

### 要求3.1

在将module chnl\_initiator和module chnl\_generator分别改造为class chnl\_initiator和class chnl\_generator后，同学们也可以发现我们同时定义了一个用来封装发送数据的类chnl\_trans。要求3.1需要在iniital块中分别例化3个已经声明过的chnl\_initiator和3个chnl\_generator。

### 要求3.2

由于每一个chnl\_initiator都需要使用接口chnl\_intf来发送数据，在发送数据之前我们需要确保chnl\_initiator中的接口不是悬空的，即需要由外部被传递。所以接下来的实验要求需要通过调用chnl\_iniaitor的方法来完成接口的传递。

### 要求3.3

接下来就可以调用已经定义过的三个test任务来展开测试了。

### 要求3.4

最后是关于类的例化问题，请同学们观察chnl\_generator在例化chnl\_trans t时，有没有不恰当的地方，如果有请指出来现有的代码会造成什么样的潜在问题呢？

## 包的定义和类的继承

到了tb4.sv，我们又进一步引入了新的类chnl\_agent、chnl\_root\_test、chnl\_basic\_test、chnl\_burst\_test和chnl\_fifo\_full\_test，同时将所有的类（都是与channel相关的验证组件类），封装到专门包裹软件类的容器package chnl\_pkg中且完成编译。因此编译后的chnl\_pkg会被默认编译到work库中，与其它的module是一同并列放置的。

关于chnl\_agent，我们将它作为一个标准组件单元，它应该包括generator、driver(initiator)和monitor。在tb4.sv中，我们暂时只有chnl\_generator和chnl\_initiator，因此将它们在agent中例化。同时，我们也将之前用task来实现的测试任务也由类来实现。可以发现，父类是chnl\_root\_test，而我们已经先移植了chnl\_basic\_test，接下来需要同学们实现另外两个类。

### 要求4.1

由于我们将各个类首先封装在了package chnl\_pkg中，因此在module tb4中要声明类的句柄，首先应该从chnl\_pkg中引入其中定义的类。

### 要求4.2

可以参考之前已经实现的burst\_test()和fifo\_full\_test()任务，以及已经实现的类chnl\_basic\_test，按照同样的要求来实现两个新的类chnl\_burst\_test和chnl\_fifof\_full\_test。

### 要求4.3

例化已经声明过的三个test组件。

### 要求4.4

完成从test一层的接口传递任务，使得其内部各个组件都可以得到需要的接口。

### 要求4.5

调用各个test的方法，展开测试。

## 实验的最后

请同学们准备一张纸，用笔画出来你理解的tb4.sv的验证环境结构。环境结构中需要包含以下要素：

* **DUT**
* **接口**的名称和数量
* 不同**验证组件**的名称、数量以及结构关系

这不单单是一个实验要求，这对于接下来我们后续课程中不断强调的验证环境构成、模块环境到系统环境的集成都是有很大帮助的，所以请同学们务必尝试画出你认为的tb4.sv的验证结构来。尽管目前每个人对环境的认识都有差别，但考虑到绘制验证环境将是日后从事验证工作的基本功，所以无论你的第一幅验证框图是什么样的，至少请你从这个实验开始迈出这一步。