

## 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，而这些都是作为日后学习高阶 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。

### tb1 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，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时钟块(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.mk 的方式来做数据驱动，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。

### tb2 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。

## tb3 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 中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?

## tb4 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_fifo_full_test`。

### ■要求 4.3

例化三个测试环境。在 4.2 中继承了三个 `chnl_root_test` 的子类, 并在 `import package` 之后, 创建了对应的句柄。现在要例化三个 `test`, 即创建对应的对象, 并将句柄指向这些对象。

### ■要求 4.4

将每个通道的接口, 赋值给对应的 `chnl_initiator`。

### ■要求 4.5

开始运行。