目录

[1 SV和UVM问题 3](#_Toc86266794)

[1.1说一下phase机制的特点 3](#_Toc86266795)

[1.2 Phase机制怎么同步？例如objection机制没有drop objection会发生什么情况？例如2个componentA和B，A啥事都没干，B没有drop objection，那么A会不会跳转到后边？ 4](#_Toc86266796)

[1.3 OPP的特性？多态看的是基类句柄还是对象？ 5](#_Toc86266797)

[1.4 讲一下callback机制 5](#_Toc86266798)

[1.5 三个线程应该怎么办？fork join有哪三种？如果加上wait fork 之后运行情况有啥变化？ 5](#_Toc86266799)

[1.6 简述UVM的工厂机制 5](#_Toc86266800)

[1.7 SV中的interface的clock blocking的功能，如果clock blocking定义在下沿，最后的结果应该是什么样？ 6](#_Toc86266801)

[1.8 动态数组和联合数组的区别？ 6](#_Toc86266802)

[1.9 UVM从哪里启动，接口怎么传递到环境中 6](#_Toc86266803)

[1.10 接口怎么传递到验证环境中（uvm\_config\_db） 7](#_Toc86266804)

[1.11 UVM的优势，为什么要用UVM 8](#_Toc86266805)

[1.12 说一下ref类型，你用到过嘛 8](#_Toc86266806)

[1.13说一下component和object的区别，item是component还是object 8](#_Toc86266807)

[1.14 Virtual sequencer 和sequencer区别 9](#_Toc86266808)

[1.15 平台往里边输入数据的话怎么输入sequence， sequence，sequencer，driver之间的通信 10](#_Toc86266809)

[1.16 代码覆盖率、功能覆盖率和断言覆盖率的区别 11](#_Toc86266810)

[1.17 为什么选验证，IC设计流程也即ASIC设计流程 11](#_Toc86266811)

[1.18 Find 队列和find index队列 11](#_Toc86266812)

[1.19用过断言嘛？写一个断言，a为高的时候，b为高，还有a为高的时候，下一个周期b为高 11](#_Toc86266813)

[1.20 形式验证 12](#_Toc86266814)

[1.21 如何保证验证的完备性? 12](#_Toc86266815)

[1.22启动Sequence的方法 12](#_Toc86266816)

[1.23面向对象编程的优势 13](#_Toc86266817)

[1.24 事件触发，wait 和@的区别 13](#_Toc86266818)

[1.25 约束的几种形式 13](#_Toc86266819)

[1.26 哪些继承于component，哪些继承于object 14](#_Toc86266820)

[1.27 get\_next\_item（）和try\_next\_item（）有什么区别 14](#_Toc86266821)

[1.28 UVM的树形结构 14](#_Toc86266822)

[1.29断言 and 和 intersect 区别 15](#_Toc86266823)

[1.30 Task和function的区别 15](#_Toc86266824)

[1.31 Break；continue；return的含义，return之后，function里剩下的语句会执行吗 15](#_Toc86266825)

[1.32 触发器和锁存器的区别 15](#_Toc86266826)

[1.33 一个简单的UVM验证平台 16](#_Toc86266827)

[1.34 怎么编写测试用例? 16](#_Toc86266828)

[1.35 如果有很多测试用例，如何让它们自动执行？ 16](#_Toc86266829)

[1.36断言$past的用法 16](#_Toc86266830)

[1.37 如何打开和关闭约束 17](#_Toc86266831)

[1.38 队列的使用方法，以及push back和pop front的区别 17](#_Toc86266832)

[1.39 Rand 和randc的区别 17](#_Toc86266833)

[1.40 组件之间的通信机制，analysis port和其它的区别1 17](#_Toc86266834)

[1.41 AHB的传输类型，说一下4回环突发的传输应该是怎么样的，这个回环边界怎么确定 18](#_Toc86266835)

[1.43 对于UVM的基本类有哪些 20](#_Toc86266836)

[1.44 简述深拷贝和浅拷贝 20](#_Toc86266837)

[1.45阻塞和非阻塞（blocking和nonblocking） 20](#_Toc86266838)

[2 Verilog问题 21](#_Toc86266839)

[2.1二分频是怎么写的？ 21](#_Toc86266840)

[2.2阻塞和非阻塞及其应用 21](#_Toc86266841)

[2.3写一个100MHz的时钟 21](#_Toc86266842)

[2.4 Reg和wire的区别 22](#_Toc86266843)

[2.5 Logic和wire的区别，两者可以转换嘛 22](#_Toc86266844)

[2.6用你擅长的语言找出1:100的质数 22](#_Toc86266845)

[2.7 一个最简单的八位加法器应该怎么验证？才有完备性？ 22](#_Toc86266846)

[2.8 在一个CPU系统中，有2个master通过一个2\*1的AXI总线访问一个Slave，简述如何构造验证场景来进行验证，并保证验证的完备性。 23](#_Toc86266847)

[2.9 FIFO作为一个通用的逻辑单元模块，应该怎么测试？ 23](#_Toc86266848)

[2.10 异步FIFO的测试点 23](#_Toc86266849)

[2.11 对同步电路和异步电路的理解 24](#_Toc86266850)

[2.12 跨时钟域 24](#_Toc86266851)

[2.13 状态机描述方法 25](#_Toc86266852)

[2.14 对于建立时间违例的解决办法按优先级有 25](#_Toc86266853)

[2.15 对于保持时间违例的解决办法按优先级有 25](#_Toc86266854)

[3 综合问题 25](#_Toc86266855)

[3.1 IC验证的流程 25](#_Toc86266856)

[3.2验证过程中怎么和设计人员沟通,如果设计人员脾气不好你会怎样做 26](#_Toc86266857)

[4 各个公司的面试问题 27](#_Toc86266858)

[4.1深圳中微电一面 27](#_Toc86266859)

[4.2鼎信提前批 27](#_Toc86266860)

[4.3地平线（原型验证） 27](#_Toc86266861)

[4.4 鼎信终面 27](#_Toc86266862)

[4.5 合肥宏晶微电子 27](#_Toc86266863)

[4.6中感微电子 28](#_Toc86266864)

[4.7 飞腾公司 28](#_Toc86266865)

[4.8 中科芯58所 30](#_Toc86266866)

[4.9乐鑫提前批 31](#_Toc86266867)

[4.10中兴通讯 31](#_Toc86266868)

[4.11集创北方 31](#_Toc86266869)

[4.12星宸科技线下：一面技术面+HR面 33](#_Toc86266870)

[4.13地平线（上海）线上（一面） 33](#_Toc86266871)

[4.14星宸科技二面 36](#_Toc86266872)

[4.15联发科一面 36](#_Toc86266873)

[4.16 zeku一面 36](#_Toc86266874)

[4.17华为一面 37](#_Toc86266875)

[4.18 展锐一面（有一到三轮技术面） 37](#_Toc86266876)

[4.19商汤一面 37](#_Toc86266877)

[4.20联芸一面 37](#_Toc86266878)

[4.21 展锐二面 38](#_Toc86266879)

[4.22联发科二面 38](#_Toc86266880)

[4.23 商汤二面 38](#_Toc86266881)

[4.24禾赛科技一面 38](#_Toc86266882)

[4.25联发科三面 38](#_Toc86266883)

[4.26紫光同芯一面 39](#_Toc86266884)

[4.27 奥比中光一面 39](#_Toc86266885)

[4.28 寒武纪一面，正式批有三面，技术面，总监面（技术面少一些，定岗），HR面 39](#_Toc86266886)

[4.29 汇顶一面 39](#_Toc86266887)

[4.30 ARM一面 40](#_Toc86266888)

[4.31 晶晨半导体 40](#_Toc86266889)

# SV和UVM问题

## 1.1说一下phase机制的特点

在不同的时间做不同的事情，这是phase机制的特点，将UVM仿真阶段层次化，build phase和final phase是自顶向下，其它都是自底向上，只有run phase是耗时的，为task phase，其它都是function phase

Run\_phase可以细分12个phase, run\_phase与细分的12个phase是并行的，但是12个phase也是按先后顺序执行的。为了避免不必要的干扰，用户可以选择run\_phase，或者12个phase中的若干来完成激励，但是请不要将它们混合起来使用，因为这样容易导致执行关系的不明确。

所有的phase按照以下顺序自上而下自动执行：(九大phase,其中run phase又分为12个小phase)



其中，\*run\_phase\*按照以下顺序自上而下执行：

pre\_reset\_phase

reset\_phase

post\_reset\_phase

pre\_configure\_phase

configure\_phase

post\_configure\_phase

pre\_main\_phase

main\_phase

post\_main\_phase

pre\_shutdown\_phase

shutdown\_phase

post\_shutdown\_phase

## 1.2 Phase机制怎么同步？例如objection机制没有drop objection会发生什么情况？例如2个componentA和B，A啥事都没干，B没有drop objection，那么A会不会跳转到后边？

Task phase的同步，一个UVM验证平台有许多component组成，每个component都有自己的run\_phase，以及从pre\_reset 到post\_shutdown的12个小phase。只有所有component的每一个小task phase 完成，整个仿真平台才开始下一个小task phase的执行。 各个component的run\_phase之间，以及run\_phase于最后一个小phase--post\_shutdown\_phase之间，都有这样的同步。

A不会跳转到后边，因为只有所有component 的task phase完成之后，整个仿真平台才开始下一个小的task phase

## 1.3 OPP的特性？多态看的是基类句柄还是对象？

封装、继承和多态

封装:通过将一些数据和使用这些数据的方法封装在一个集合里，成为一个类。

继承:允许通过现有类去得到一个新的类，且其可以共享现有类的属性和方法。现有类叫做基类，新类叫做派生类或扩展类。

多态:得到扩展类后，有时我们会使用基类句柄去调用扩展类对象，这时候调用的方法如何准确去判断是想要调用的方法呢?通过对类中方法进行 virtual声明，这样当调用基类句柄指向扩展类时，方法会根据对象去识别，调用扩展类的方法，而不是基类中的。而基类和扩展类中方法有着同样的名字，但能够准确调用，叫做多态。

## 1.4 讲一下callback机制

Callback机制其作用是提高TB的可重用性,其还可进行特殊激励的产生等，与factory类似，两者可以有机结合使用。与factory不同之处在于 callback的类还是原先的类，只是内部的callback函数变了，而factory这是产生一个新的扩展类进行替换。

1) UVM组件中内嵌callback函数或者任务

2)定义一个常见的uvm\_callbacks class

3) 从UVM callback空壳类扩展uvm\_callback类

4)在验证环境中创建并登记uvm\_callback

## 1.5 三个线程应该怎么办？fork join有哪三种？如果加上wait fork 之后运行情况有啥变化？

Fork join:内部 begin end块并行运行，直到所有线程运行完毕才会进入下一个阶段。

Fork join\_any:内部 begin end块并行运行，任意一个begin end块运行结束就可以进入下一个阶段。

Fork join\_none:内部begin end块并行运行，无需等待可以直接进入下一个阶段。

wait fork：会引起调用进程阻塞，直到它的所有子进程结束，一般用来确保所有子进程（调用进程产生的进程，也即一级子进程）执行都已经结束。

disable fork：用来终止调用进程 的所有活跃进程， 以及进程的所有子进程。

## 1.6 简述UVM的工厂机制

Factory机制也叫工厂机制，其存在的意义就是为了能够方便的替换TB中的实例或者已注册的类型。一般而言，在搭建完TB后，我们如果需要对TB进行更改配置或者相关的类信息，我们可以通过使用factory 机制进行覆盖，达到替换的效果，从而大大提高TB的可重用性和灵活性。要使用factory机制先要进行：

1.将类注册到factory表中

2.创建对象，使用对应的语句 （type\_id::create）

3.编写相应的类对基类进行覆盖。

## 1.7 SV中的interface的clock blocking的功能，如果clock blocking定义在下沿，最后的结果应该是什么样？

Interface是一组接口，用于对信号进行一个封装，捆扎起来。如果像 verilog中对各个信号进行连接，每一层我们都需要对接口信号进行定义，若信号过多，很容易出现人为错误，而且后期的可重用性不高。因此使用interface接口进行连接，不仅可以简化代码，而且提高可重用性，除此之外，interface内部提供了其他一些功能，用于测试平台与DUT之间的同步和避免竞争。

Clocking block:在interface内部我们可以定义clocking块，可以使得信号保持同步,对于接口的采样和驱动有详细的设置操作，从而避免TB与 DUT的接口竞争，减少我们由于信号竞争导致的错误。采样提前，驱动落后，保证信号不会出现竞争。

如果clock blocking定义在下沿，可能会出现竞争

## 1.8 动态数组和联合数组的区别？

队列:队列结合了链表和数组的优点，可以在一个队列的任何位置进行增加或者删除元素。其通过[$]这样的符号进行申明: int q[$];

定宽数组:属于静态数组，编译时便已经确定大小。其可以分为压缩定宽数组和非压缩定宽数组:压缩数组是定义在类型后面，名字前面;非压缩数组定义在名字后面。Bit [7:0][3:0] name; bit[7:0] name [3:0];

动态数组:其内存空间在运行时才能够确定，使用前需要用new[]进行空间分配。

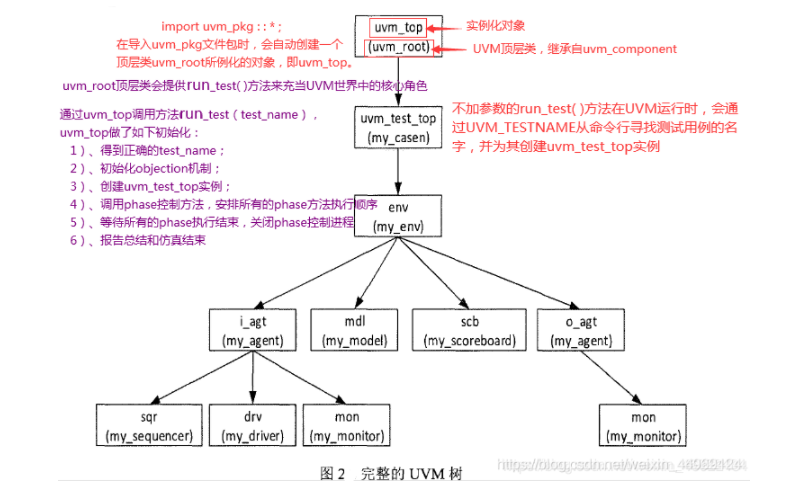
关联数组:其主要针对需要超大空间但又不是全部需要所有数据的时候使用，类似于hash，通过一个索引值和一个数据组成: bit [63:0] name[bit[63:0]];索引值必须是唯一的。

1. 【关联数组】可以用来保存稀疏矩阵的元素。当你对一个非常大的地址空间寻址时，该数组只为实际写入的元素分配空间，这种实现方法所需要的空间要小得多。
2. 此外，关联数组有其它灵活的应用，在其它软件语言也有类似的数据存储结构，被称为哈希(Hash)或者词典(Dictionary)，可以灵活赋予键值(key)和数值(value) 。

## 1.9 UVM从哪里启动，接口怎么传递到环境中

UVM的启动

总结：1）在导入uvm\_pkg文件时，会自动创建UVM\_root所例化的对象/UVM\_top，UVM顶层的类会提供run\_test()方法充当UVM世界的核心角色，通过UVM\_top调用run\_test()方法。 2）在环境中输入run\_test来启动UVM验证平台，run\_test语句会创建一个my\_case0的实例，得到正确的test\_name 2）依次执行uvm\_test容器中的各个component组件中的phase机制，按照顺序， 1.build-phase（自顶向下构建UVM 树） 2.connet\_phase(自低向上连接各个组件) 3.end\_of\_elaboration\_phase 4.start\_of\_simulation\_phase 5.run\_phase() objection机制仿真挂起，通过start启动sequence（每个sequence都有一个body任务。当一个sequence启动后，会自动执行sequence的body任务）,等到sequence发送完毕则关闭objection，结束run\_phase()（UVM\_objection提供component和sequence共享的计数器，当所有参与到objection机制中的组件都落下objection时，计数器counter才会清零，才满足run\_phase()退出的条件（UVM入门P45）） 5.执行后面的phase



## 1.10 接口怎么传递到验证环境中（uvm\_config\_db）

1. 传递virtual interface到环境中；
2. 配置单一变量值，例如int、string、enum等；
3. 传递配置对象（config\_object）到环境；
4. 传递virtual interface到环境中；
5. 虽然SV可以通过层次化的interface的索引完成传递，但是这种传递方式不利于软件环境的封装和复用。通过使用uvm\_config\_db配置机制来传递接口，可以将接口的传递与获取彻底分离开。
6. 接口传递从硬件世界到UVM环境可以通过uvm\_config\_db来实现，在实现过程中应当注意：
7. 接口传递应发生在run\_test()之前。这保证了在进入build\_phase之前，virtual interface已经被传递到uvm\_config\_db中。
8. 用户应当把interface与virtual interface区分开来，在传递过程中的类型应当为virtual interface，即实际接口的句柄。
9. 配置单一变量值，例如int、string、enum等；

在各个test中，可以在build\_phase阶段对底层组件的各个变量加以配置，进而在环境例化之前完成配置，使得环境可以按照预期运行。

1. 传递配置对象（config\_object）到环境；

在test配置中，需要配置的参数不只是数量多，可能还分属于不同的组件。对这么多层次的变量做出类似上边的单一变量传递，需要更多的代码，容易出错且不易复用。如果整合各个组件中的变量，将其放置在一个uvm\_object中，再对中心化的配置对象进行传递，将有利于整体环境的修改维护，提升代码的复用性。

## 1.11 UVM的优势，为什么要用UVM

UVM 其实就是 SV 的一个封装，将我们在搭建测试平台过程中的一些重复性和重要的工作进行封装，从而使我们能够快速的搭建一个需要的测试平台，并且可重用性还高。但是UVM又不仅仅是封装。

## 1.12 说一下ref类型，你用到过嘛

Ref参数类型是引用

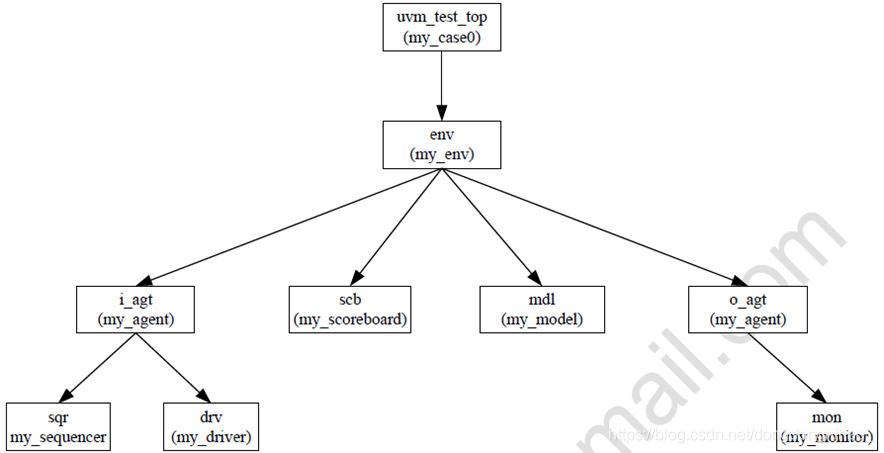
1. 向子程序传递数组时应尽量使用ref获取最佳性能，如果不希望子程序改变数组的值，可以使用const ref类型
2. 在任务里可以修改变量而且修改结果对调用它的函数随时可见。

## 1.13说一下component和object的区别，item是component还是object

UVM中 component也是由object派生出来的，不过相比于object, component有很多其没有的属性，例如phase机制和树形结构等。在UVM中，不仅仅需要component 这种较为复杂的类，进行TB的层次化搭建，也需要object这种基础类进行TB的事务搭建和一些环境配置等。

Item是object。

UVM的树形结构



UVM验证环境的组成：

Sequencer:负责将数据转给 driver ,driver负责数据的发送;driver有时钟/时序的概念。

Agent:其实只是简单的把driver ,monitor和sequencer封装在一起。

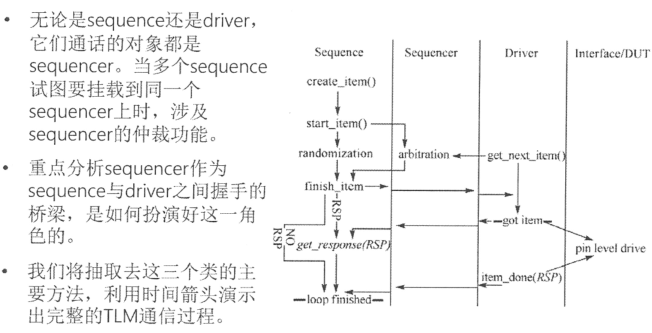
Agent:对应的是物理接口协议,不同的接口协议对应不同的agent ，一个平台通常会有多个agent。

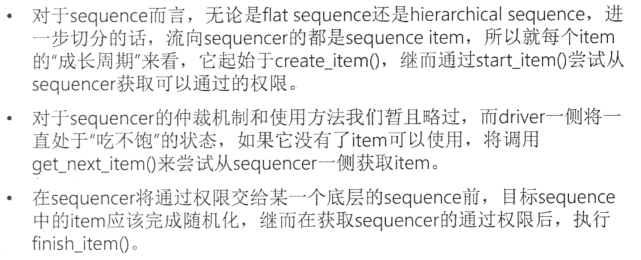
Env：则相当于是一个特大的容器，将所有成员包含进去。

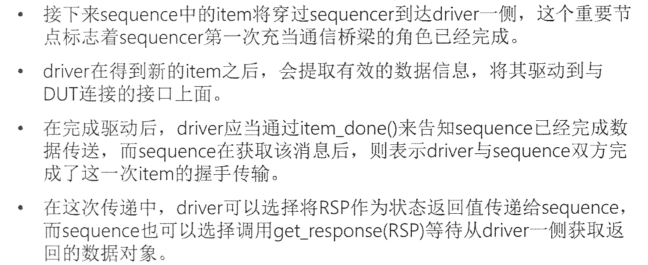
## 1.14 Virtual sequencer 和sequencer区别

Virtual sequencer主要用于对不同的agent进行协调时，需要有一定顶层的sequencer对内部各个agent中的sequencer进行协调，因此 virtual sequencer是面向多个sequencer的多个sequence群，而sequencer是面向一个sequencer 的sequence群。Virtual sequencer桥接着所有底层的sequencer 的句柄，其本身也不需要传递item，不需要和 driver连接。只需要将其内部的底层sequencer句柄和sequencer 实体对象连接。

## 1.15 平台往里边输入数据的话怎么输入sequence， sequence，sequencer，driver之间的通信







在多个sequence同时向sequencer发送item时，需要有ID信息表明该item从哪个sequence来，ID信息在sequence创建item时就赋值了。

## 1.16 代码覆盖率、功能覆盖率和断言覆盖率的区别

代码覆盖率——是针对RTL设计代码的运行完备度的体现，包括行覆盖率、条件覆盖率、FSM覆盖率、跳转覆盖率、分支覆盖率，只要仿真就可以收集，可以看DUT的哪部分代码没有动，如果有一部分代码一直没动看一下是不是case没有写到。

功能覆盖率---与spec比较来发现，design是否行为正确，需要按verification plan来比较进度。用来衡量哪些设计特征已经被测试程序测试过的一个指标，首要的选择是使用更多的种子来运行现有的测试程序；其次是建立新的约束，只有在确实需要的时候才会求助于定向测试，改进功能覆盖率最简单的方法是仅仅增加仿真时间或者尝试新的随机种子。验证的目的就是确保设计在实际环境中的行为正确。设计规范里详细说明了设备应该如何运行，而验证计划里则列出了相应的功能应该如何激励、验证和测量

断言覆盖率---用于检查几个信号之间的关系，常用在查找错误，主要是检查时序上的错误，测量断言被触发的频繁程度。

## 1.17 为什么选验证，IC设计流程也即ASIC设计流程

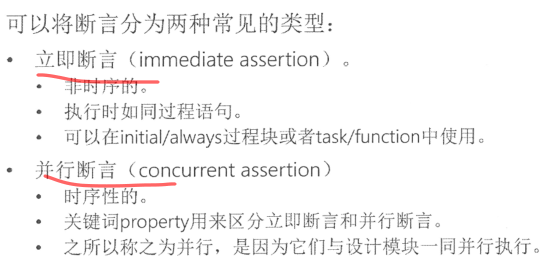
芯片架构-RTL设计-功能仿真-综合&扫描链的插入（DFT）-等价性检查-形式验证-静态时序分析（STA）-布局规划-布局布线-布线图和原理图比较-设计规则检查-GDII

## 1.18 Find 队列和find index队列

find的队列应该是返回队列的值，一般的话是和with配合使用，find index应该是返回索引值

## 1.19用过断言嘛？写一个断言，a为高的时候，b为高，还有a为高的时候，下一个周期b为高

立即断言和并行断言



## 1.20 形式验证

形式验证指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。形式验证方法分为等价性验证、模型检验和定理证明等。

形式验证主要验证数字IC设计流程中的各个阶段的代码功能是否一致，包括综合前RTL代码和综合后网表的验证，因为如今IC设计的规模越来越大，如果对门级网表进行动态仿真，会花费较长的时间，而形式验证只用几个小时即可完成一个大型的验证。另外，因为版图后做了时钟树综合，时钟树的插入意味着进入布图工具的原来的网表已经被修改了，所以有必要验证与原来的网表是逻辑等价的

## 1.21 如何保证验证的完备性?

首先不可能百分百完全完备，即遍历所有信号的组合，这既不经济也不现实。所以只能通过多种验证方法一起验证尽可能减少潜在风险，一般有这些验证流程：ip级验证、子系统级验证、soc级验证，除这些以外，还有upf验证、fpga原型验证等多种手段。前端每走完一个阶段都需要跟设计以及系统一起review验证功能点，测试用例，以及特殊情况下的波形等。

芯片后端也会做一些检查，像sta、formality、DFM、DRC检查等，也会插入一些DFT逻辑供流片回来测试用。流片归来进行测试，有些bug可以软件规避，有些不能规避，只能重新投片

## 1.22启动Sequence的方法

严格意义上有2种：

1. 通过sequence.start的方式显示启动
2. 通过default sequence来隐式启动

通过‘uvm\_do系列宏启动

## 1.23面向对象编程的优势

1、易维护

采用面向对象思想设计的结构，可读性高，由于继承的存在，即使改变需求，那么维护也只是在局部模块，所以维护起来是非常方便和较低成本的。

2、质量高

在设计时，可重用现有的，在以前的项目的领域中已被测试过的类使系统满足业务需求并具有较高的质量。

3、效率高

在软件开发时，根据设计的需要对现实世界的事物进行抽象，产生类。使用这样的方法解决问题，接近于日常生活和自然的思考方式，势必提高软件开发的效率和质量。

4、易扩展

由于继承、封装、多态的特性，自然设计出高内聚、低耦合的系统结构，使得系统更灵活、更容易扩展，而且成本较低。

## 1.24 事件触发，wait 和@的区别

用来触发事件时，使用**->**；用来等待事件使用**@**或者**wait，@和wait的区别：**

## 1.25 约束的几种形式

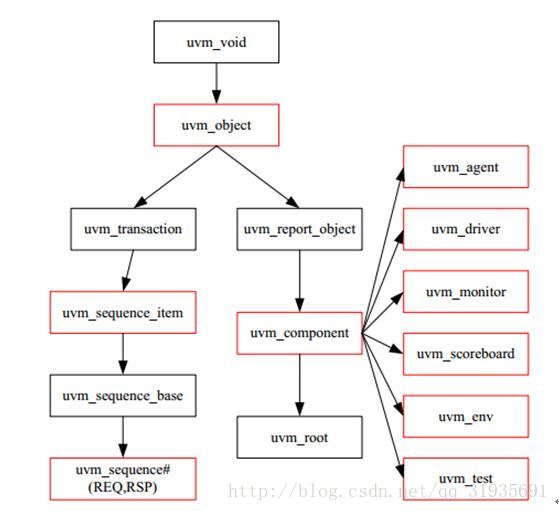
随机化是SV中极其重要的一个知识点，通过设定随机化和相关约束，我们可以自动随机想要的数据。

权重约束 dist：有两种操作符：:=n :/n 第一种表示每一个取值权重都是n,第二种表示每一个取值权重为n/num。

条件约束 if else 和->（case）：if else 就是和正常使用一样；->通过前面条件满足后可以触发后面事件的发生。

范围约束inside：inside{[min:max]}；范围操作符，也可以直接使用大于小于符号进行，不可以连续使用，如 min<wxm<max 这是错误的

## 1.26 哪些继承于component，哪些继承于object



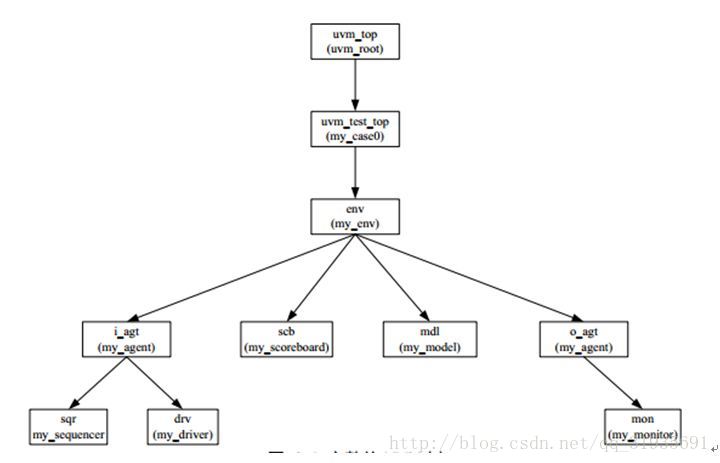
除了driver、monitor、agent、model、scoreboard、env、test之外全部用uvm\_object。

## 1.27 get\_next\_item（）和try\_next\_item（）有什么区别

get\_next\_item（）是一个阻塞调用，直到存在可供驱动的sequence item为止，并返回指向sequence item的指针。

try\_next\_item（）是非阻塞调用，如果没有可供驱动的sequence item，则返回空指针。

## 1.28 UVM的树形结构



## 1.29断言 and 和 intersect 区别

And 指的是两个序列具有相同的起始点，终点可以不同。

Intersect 指的是两个序列具有相同的起始点和终点。

Or 指的是两个序列只要满足一个就可以

Throughout 指的是满足前面要求才能执行后面的序列

## 1.30 Task和function的区别

1. 函数能调用另一个函数，但不能调用任务，任务能调用另一个任务，也能调用另一个函数
2. 函数总是在仿真时刻0就开始执行，任务可以在非零时刻执行
3. 函数一定不能包含任何延迟、事件或者时序控制声明语句，任务可以包含延迟、事件或者时序控制声明语句
4. 函数至少有一个输入变量，可以有多个输入变量，任务可以没有或者多个输入(input)、输出(output)和双向(inout)变量
5. 函数只能返回一个值，函数不能有输出(output)或者双向(inout)变量，任务不返回任何值，任务可以通过输出(output)或者双向(inout)变量传递多个值

## 1.31 Break；continue；return的含义，return之后，function里剩下的语句会执行吗

break 语句结束整个循环。

continue 立即结束本次循环，继续执行下一次循环。

return 语句会终止函数的执行并返回函数的值(如果有返回值的话)。

Return之后，function里剩下的语句不能执行，其是终止函数的执行，并返回函数的值。

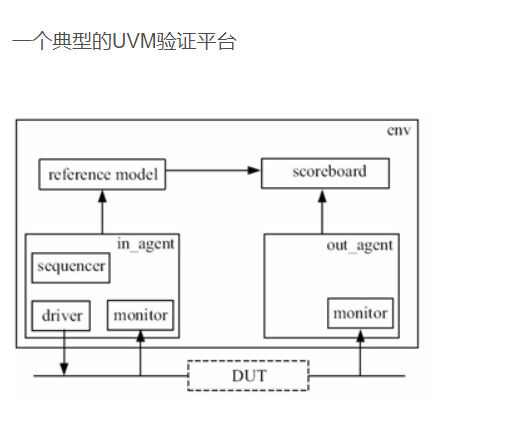
## 1.32 触发器和锁存器的区别

触发器：时钟触发，受时钟控制，只有在时钟触发时才采样当前的输入，产生输出。

锁存器由电平触发，非同步控制。在使能信号有效时锁存器相当于通路，在使能信号无效时锁存器保持输出状态。触发器由时钟沿触发，同步控制。

锁存器对输入电平敏感，受布线延迟影响较大，很难保证输出没有毛刺产生；触发器则不易产生毛刺

## 1.33 一个简单的UVM验证平台



## 1.34 怎么编写测试用例?

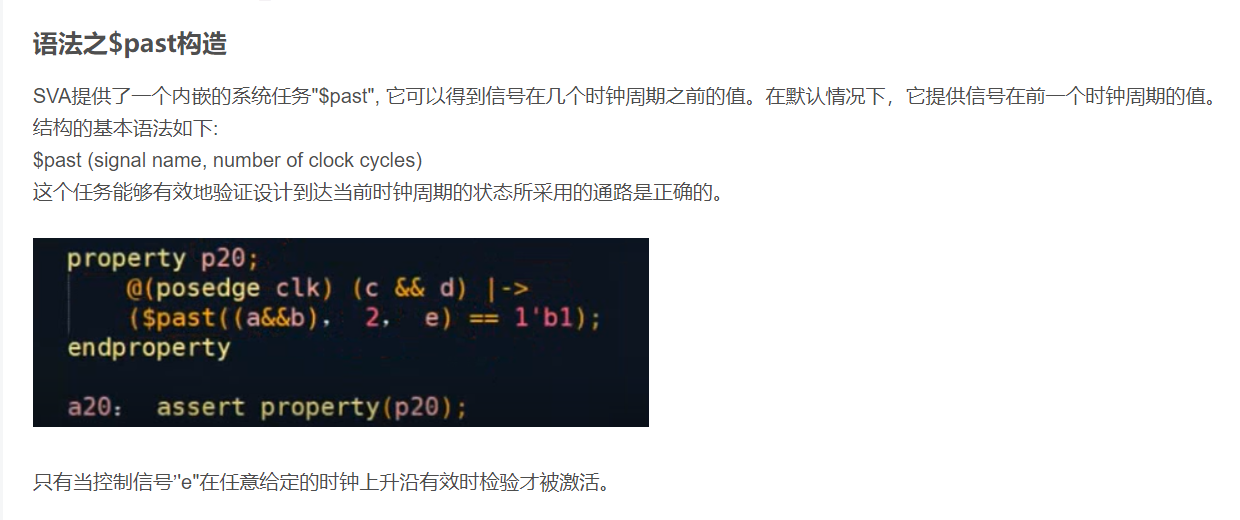
主要是编写sequence，然后在body里面根据测试功能要求写相应的激励，然后再通过reference model和checker判断功能是否实现？

## 1.35 如果有很多测试用例，如何让它们自动执行？

可以写脚本让它们自动执行，例如makefile...

## 1.36断言$past的用法

abcd四个信号在时钟沿处监测，当cd同时为1时，在时钟的前两个周期要ab同时为1



## 1.37 如何打开和关闭约束

通过constraint\_mode（0）关闭默认范围的约束块

constraint\_mode(1)是打开约束

可以用soft关键字修饰特定的约束语句，这样既可以让变量在一般的情况下取默认值，也可以直接给变量赋默认值范围外的取值。

## 1.38 队列的使用方法，以及push back和pop front的区别

队列的使用方法：insert，delete,push\_back和pop\_front

Push插入，pop取出

Front前边，back后边

## 1.39 Rand 和randc的区别

rand修饰符：rand修饰的变量，每次随机时，都在取值范围内随机取一个值，每个值被随机到的概率是一样的，就像掷骰子一样。

randc修饰符：randc表示周期性随机，即所有可能的值都取到过后，才会重复取值

## 1.40 组件之间的通信机制，analysis port和其它的区别1

1、通信分为，单向通信，双向通信和多向通信

单向通信：指的是从initiator到target之间的数据流向是单一方向的

双向通信：双向通信的两端也分为initiator和target，但是数据流向在端对端之间是双向的

多向通信：仍然是两个组件之间的通信，是指initiator与target之间的相同TLM端口数目超过一个时的处理解决办法。

2、blocking阻塞传输的方法包含:

Put（）：initiator先生成数据Tt，同时将该数据传送至target。

Get（）:initiator从target获取数据Tt，而target中的该数据Tt则应消耗。

Peek(): initiator从target获取数据Tt，而target中的该数据Tt还应保留。

3、通信管道：

1. TLM FIFO：可以进行数据缓存，功能类似于mailbox，不同的地方在于uvm\_tlm\_fifo提供了各种端口（put、get、peek）供用户使用
2. analysis port：一端对多端，用于多个组件同时对一个数据进行处理，如果这个数据是从同一个源的TLM端口发出到达不同组件，则要求该端口能够满足一端到多端，如果数据源端发生变化需要通知跟它关联的多个组件时，我们可以利用观察者模式实现，即广播模式
3. analysis TLM FIFO

* 由于analysis端口提出实现了一端到多端的TLM数据传输，而一个新的数据缓存组件类uvm\_tlm\_analysis\_fifo为用户们提供了可以搭配uvm\_analysis\_port端口uvm\_analysis\_imp端口和write()函数。
* uvm\_tlm\_analysis\_fifo类继承于uvm\_tlm\_fifo，这表明它本身具有面向单一TLM端口的数据缓存特性，而同时该类又有一个uvm\_analysis\_imp端口analysis\_export并且实现了write()函数:

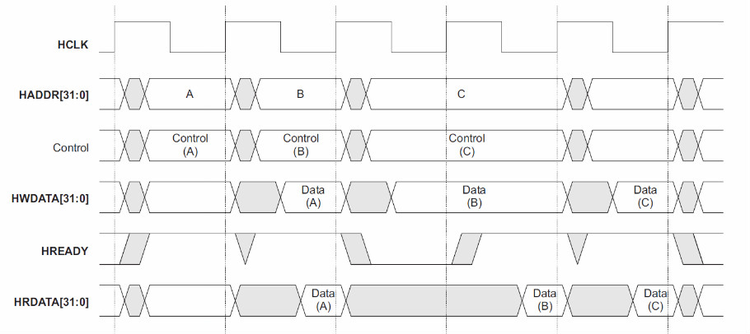
1. request & response通信管道

双向通信端口transport，即通过在target端实现transport()方法可以在一次传输中既发送request又可以接收response。

## 1.41 AHB的传输类型，说一下4回环突发的传输应该是怎么样的，这个回环边界怎么确定

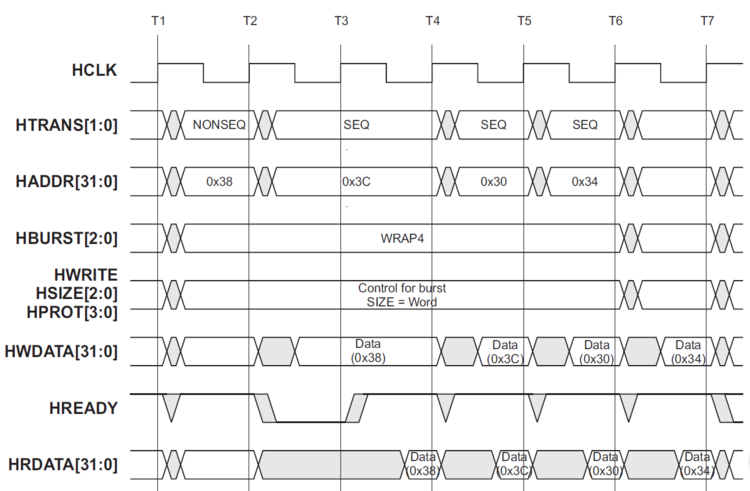
AHB的传输类型burst，INCR，WRAP

多个single transfer的pipeline操作

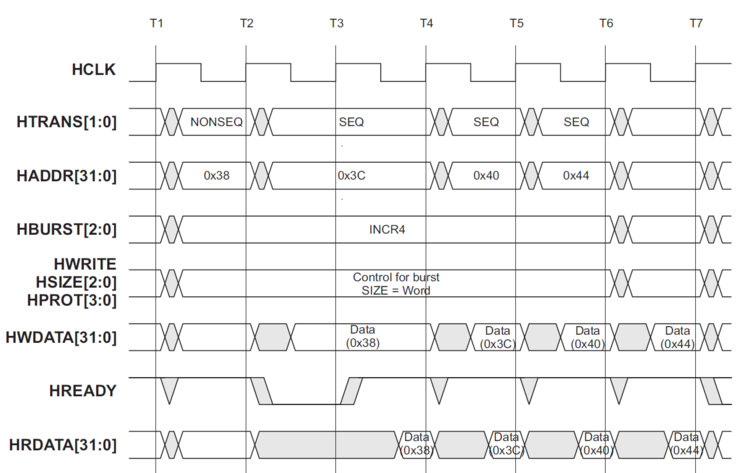




WRAP4



跟之前唯一的区别在于地址的不同，在0x3C地址之后，根据回环的地址边界，第三拍的地址变为0x30。一共四拍，4个地址，每个地址各不相同，这四个地址是一个回环范围。起始地址决定了回环操作的回环范围。



和上面的基本一致，只不过地址在0x30并不回环，而是递增。

4回环突发的传输：

如果是4的话，它的4回环突发应该是什么样的4，8，C，0，一直循环

## 1.43 对于UVM的基本类有哪些

Uvm\_component类和uvm\_object类

## 1.44 简述深拷贝和浅拷贝

1. 浅拷贝可以使用列表自带的copy()函数（如list.copy()），或者使用copy模块的copy()函数。深拷贝只能使用copy模块的deepcopy(),所以使用前要导入：from copy import deepcopy
2. 如果拷贝的对象里的元素只有值，没有引用，那浅拷贝和深拷贝没有差别，都会将原有对象复制一份，产生一个新对象，对新对象里的值进行修改不会影响原有对象，新对象和原对象完全分离开。
3. 如果拷贝的对象里的元素包含引用（像一个列表里储存着另一个列表，存的就是另一个列表的引用），那浅拷贝和深拷贝是不同的，浅拷贝虽然将原有对象复制一份，但是依然保存的是引用，所以对新对象里的引用里的值进行修改，依然会改变原对象里的列表的值，新对象和原对象完全分离开并没有完全分离开。而深拷贝则不同，它会将原对象里的引用也新创建一个，即新建一个列表，然后放的是新列表的引用，这样就可以将新对象和原对象完全分离开。

## 1.45阻塞和非阻塞（blocking和nonblocking）

阻塞和非阻塞通常用来形容多线程间的相互影响。比如一个线程占用临界区资源，那么其它所有需要这个资源的线程就必须在这个临界区中进行等待，等待会导致线程挂起。这种情况就是阻塞。此时，如果占用资源的线程一直不愿意释放资源，那么其它所有阻塞在这个临界区上的线程都不能工作。

功能覆盖率是自己写的嘛，包含了哪些？

Ref参数，如果task或function里对ref数据进行改变，外面的会变化吗

子类句柄可以指向父类句柄嘛？父类可以指向子类嘛？为什么父类指向子类要用cast。用cast什么状况下会成功，什么状况下会失败

Typedef的用法

假如有个item constraint是0-10，现在有个case，需要这么item在10-20里随机，应该怎么做

Tlm端口

对于内部寄存器的访问，是通过virtual sequence进行一个访问操作嘛

最得意的debug经历

简述深拷贝浅拷贝

对于一个VIP的修改（对驱动信号的修改），有什么思路

RAL是如何进行一个读写的操作

Uvm\_callback在项目中是如何使用的；

Run phase 和main phase之间的关系

Main\_phase要如何跳转到reset\_phase

断言同时开始同时结束

带参数的断言

Event的使用

为什么要用interface这种方式来链接UVM和DUT

可不可以直接在tb里边driver一个信号用tb这种hierarchy的方式直接连到RTL里边

介绍一下寄存器模型，你觉得为什么需要寄存器模型，有什么好处

同一个seq发数据，怎么做到乱序

# Verilog问题

## 2.1二分频是怎么写的？

偶数分频：写一个0：（N-1）计数器，再写个分频器，在N/2-1和N-1的时候翻转。

奇数分频：写两个计数器和分频器（上升沿计数器（0：N-1），下降沿计数器，上升沿分频器（在（N-1）/2和N-1的时候翻转），下降沿分频器），然后再将分频结果进行或运算。

## 2.2阻塞和非阻塞及其应用

阻塞赋值语句（“=”）和非阻塞赋值语句（“<=”）

阻塞：一般对应电路中的组合逻辑赋值，等号右端的结果会立刻赋值给左端。

非阻塞：一般对应电路中的时序逻辑赋值，等号右端的结果不会立刻赋值给左端。

在always语句中，阻塞赋值等号左端的参数如果参与该模块的其他运算，则按照赋值后的结果参与运算，而非阻塞赋值等号左端的参数依旧按照未赋值前的结果参与运算。

Assign语句中用阻塞赋值，

组合电路用阻塞赋值，时序电路用非阻塞赋值

## 2.3写一个100MHz的时钟

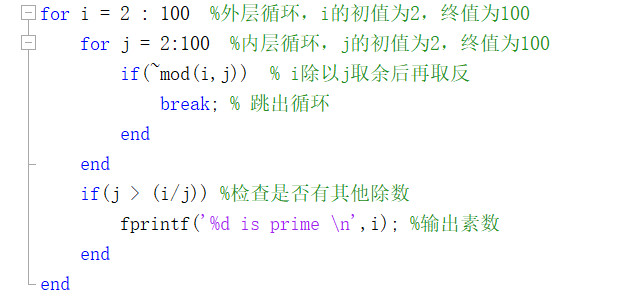
## 2.4 Reg和wire的区别

1. wire型数据常用来表示以assign关键字指定的组合逻辑信号，模块的输入输出端口类型都默认为wire型，wire相当于物理连线，默认初始值是z。
2. reg型表示的寄存器类型，用于always模块内被赋值的信号，必须定义为reg型，代表触发器，常用于时序逻辑电路，reg相当于存储单元，默认初始值是x。
3. wire只能被assign连续赋值，reg只能在initial和always中赋值
4. inout是一个双向端口, inout端口不能声明为reg类型，只能是wire类型。

## 2.5 Logic和wire的区别，两者可以转换嘛

1. Logic是reg类型的改进，既可被过程赋值也能被连续赋值，编译器可自动推断logic是reg还是wire，但是logic只允许一个输入，不能被多重驱动，所以inout类型端口不能定义为logic。
2. 单驱动时logic可完全替代reg和wire。
3. 多驱动时，如inout类型端口，使用wire。

## 2.6用你擅长的语言找出1:100的质数



## 2.7 一个最简单的八位加法器应该怎么验证？才有完备性？

1. 首先拿到这个加法器时先看基本逻辑，比如有没有复位信号，有复位信号，验证复位功能是否正常
2. 然后从功能角度，这个加法器是只用来做正数加正数还是负数的运算也可以，所以应验证其正数+正数，正数+负数以及负数+负数
3. 从数据随机的角度来说00001111这种边界情况需要测，还有随机出现数据
4. 最后还要考虑模块出错的情况，如果我们的数据源出现了X值，这个加法器模块会怎么处理
5. 加法器如果没有带时序逻辑进行采样，就是一个组合逻辑电路，还可以验证是否出现毛刺（待定）

## 2.8 在一个CPU系统中，有2个master通过一个2\*1的AXI总线访问一个Slave，简述如何构造验证场景来进行验证，并保证验证的完备性。

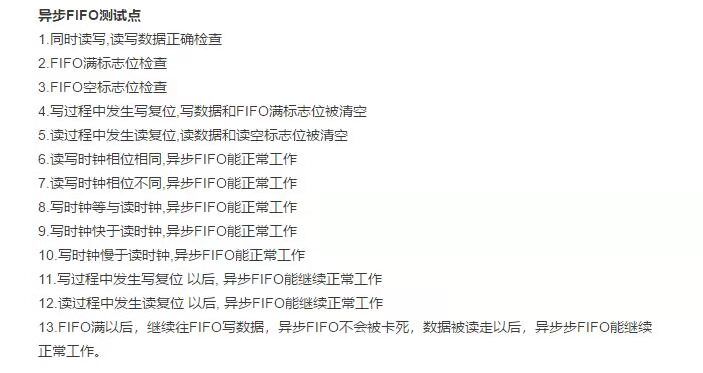
1. 第一个就是最基本的测试，就是单master访问slave用master0访问slave，然后用master1访问slave，然后每个master访问slave的时候要对axi的所有协议都要进行测试，这是最基本的测试。
2. 第二个的话可以测一下他那个地址边界，如果你这个slave是memory的话，那我访问你这个memory地址范围以外的话，你是不能正常读写的。
3. 第三个就是仲裁访问测试，两个master同时访问slave的时候，要看谁的优先级更高，这个要测一下。
4. 最后一个测试要测的是数据的一致性完整性，Master写进去的数和读出来的数必须要一样，而且master0写进去的数，然后master1读出来也必须要一样。

## FIFO作为一个通用的逻辑单元模块，应该怎么测试？

1. FIFO深度检查
2. FIFO位宽检查
3. FIFO空满判断
4. FIFO满后再继续写数据，或者是FIFO空后再继续读数据，会发生什么
5. FIFO清空检查，软件复位后可以清空FIFO嘛

## 异步FIFO的测试点

1. 同时读写，读写数据正确检查
2. FIFO满标志位检查
3. FIFO空标志位检查
4. 写过程中发生写复位，写数据和FIFO满标志位被清空
5. 读过程中发生读复位，读数据和读空标志位被清空
6. 读写时钟相位相同，异步FIFO能正常工作
7. 读写时钟相位不同，异步FIFO能正常工作
8. 写时钟等于读时钟，异步FIFO能正常工作
9. 写时钟快于读时钟，异步FIFO能正常工作
10. 写时钟慢于读时钟，异步FIFO能正常工作
11. 写过程中发生写复位以后，异步FIFO 能继续正常工作
12. 读过程中发生读复位以后，异步FIFO 能正常工作
13. FIFO满以后，继续往FIFO写数据，异步FIFO不会被卡死，数据被读走以后，异步FIFO 能继续正常工作



## 对同步电路和异步电路的理解

同步电路和异步电路的区别在于电路触发是否与驱动时钟同步，从行为上讲，就是所有电路是否在同一时钟沿下同步地处理数据。

同步复位和异步复位电路是同步电路和异步电路中两个典型的逻辑单元。在同步复位电路中，当复位信号有效时，必须要等到时钟沿有效时，才能处理复位信号相关逻辑行为；而在异步复位电路中，当复位信号有效时，立即处理复位信号相关逻辑行为。

同步时钟sync是指时钟之间相位相对确定，异步时钟async是指时钟之间相位不确定。看到一些地方认为同步时钟sync是指时钟之间频率和相位都相对确定，异步时钟async是指时钟之间频率或相位不确定，也不能说没有道理吧。不过我坚持认为根源是在相位确不确定上，例如33M和100M时钟就一定不可能是同步时钟，因为无论如何做不到相位保持一致；而50M和100M时钟就有可能是同步时钟，因为相位可以保持一致。

## 跨时钟域

单bit（慢时钟域到快时钟域）：用快时钟打两拍，直接采一拍大概率也是没问题的，两拍的主要目的是消除亚稳态；

单bit（快时钟域到慢时钟域）：握手、异步FIFO、异步双口RAM；快时钟域的信号脉宽较窄，慢时钟域不一定能采到，可以通过握手机制让窄脉冲展宽，慢时钟域采集到信号后再“告诉”快时钟域已经采集到信号，确保能采集到；

多bit：异步FIFO、异步双口RAM、握手、格雷码；

多bit中，强烈推荐异步FIFO，我在实际工程中使用多次，简单方便。

## 状态机描述方法

状态机描述时关键是要描述清楚几个状态机的要素，即如何进行状态转移，每个状态的输出是什么，状态转移的条件等。具体描述时方法各种各样，最常见的有三种描述方式：

（1）一段式：整个状态机写到一个always模块里面，在该模块中既描述状态转移，又描述状态的输入和输出；

（2）二段式：用两个always模块来描述状态机，其中一个always模块采用同步时序描述状态转移；另一个模块采用组合逻辑判断状态转移条件，描述状态转移规律以及输出；

（3）三段式：在两个always模块描述方法基础上，使用三个always模块，一个always模块采用同步时序描述状态转移，一个always采用组合逻辑判断状态转移条件，描述状态转移规律，另一个always模块描述状态输出(可以用组合[电路](http://bbs.elecfans.com/zhuti_dianlu_1.html)输出，也可以时序电路输出)。

## 对于建立时间违例的解决办法按优先级有

1、首先是修改代码，找到关键路径上的源寄存器和目的寄存器，拆分它们之间的组合逻辑。插入寄存器是最简单粗暴的办法，实际上在设计之初，如果我们精心设计的话，是可以在算法级对组合逻辑进行分解的，好的时序是设计出来的，不是约束出来的；

2、如果修改代码实在解决不了，在使用约束关键路径的办法；

3、应该还有其他很多办法，不过我暂时还不知道；

如果上面都解决不了，我们还可以：

4、买更好更快的芯片，更好的芯片意味着更低的Tsu要求，更小的Tpd、Tcomb；

5、降低工作频率，即提高时钟周期。

## 对于保持时间违例的解决办法按优先级有

1、增加Tdata = (Tcomb+T逻辑组合布线延迟），具体操作是在data line上(两个触发器之间的组合逻辑块部分)插入buffer，反相器或者delay cell去增加Tdata；

2、增加前级D触发器的驱动延迟；

3、减小Tskew，减小时钟偏移；

# 综合问题

## 3.1 IC验证的流程

1、搞清楚要验证的东西

对设计spec进行阅读和理解，把DUT的结构、功能，时序弄清楚。

2、编写验证计划，指导性的文件

（1）提取验证功能点

（2）明确DUT接口信号（所有信号的名字，位宽，功能，时序关系等）

（3）TB的架构（能够描述每一个组件的功能）

（4）检查点（check point）

（5）功能覆盖率（覆盖点）

（6）测试用例的规划（testcase尽可能的规划完整）

（7）结束标准

3、搭建TB&Debug&调通第一个最基本的testcase，然后再编写测试用例debug,进行主要验证。

4、Regression（回归测试，一天一次，有随机测试）

5、分析代码/功能覆盖率，增加新的测试用例

6、测试报告，测试结果：测试用例（pass/fail），覆盖率报告

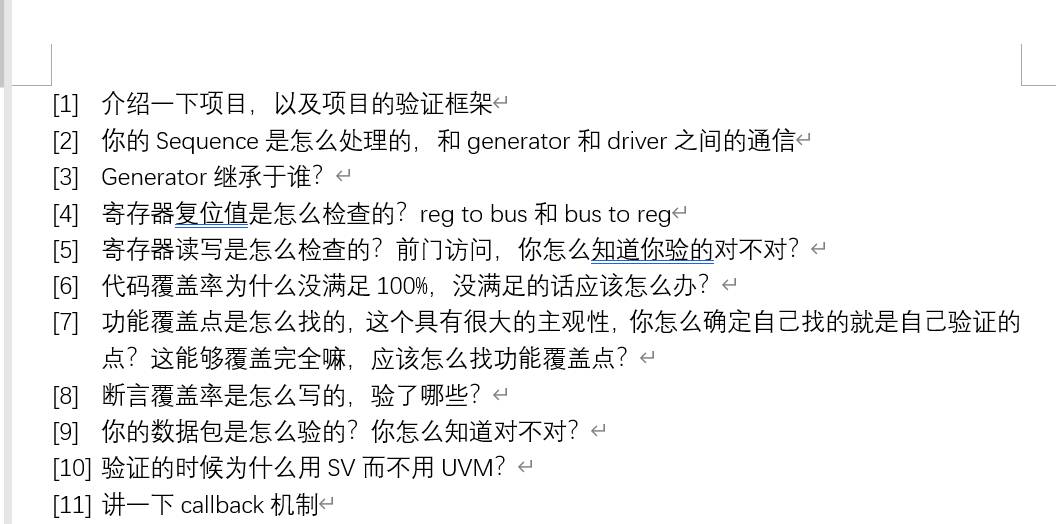
## 3.2验证过程中怎么和设计人员沟通,如果设计人员脾气不好你会怎样做

首先做好本职工作，看spec的功能描述文档，在设计的过程中和设计人员以及有经验的人一起制定测试点以及验证计划，在完成自己工作的同时，要经常和设计人员进行沟通，了解DUT的功能，验证过程中如果出现问题且设计人员脾气不好的话，应该在对自己的验证非常有信心的情况下，注意方法，语气和设计人员沟通，拿出关键点和其沟通，且大家也都受过高等教育，不至于强词夺理，一意孤行，毕竟芯片的设计及验证还是比较重要的，在设计和验证中我们都有比较大的责任，应该谨慎一些。

* + 1. clock gating 门控时钟技术。时钟信号的翻转率是比较高的，它的功耗约占整个芯片功耗的20-30%。传统的设计方法是时钟信号一直是常开的，门控时钟技术就是根据设计，将暂时不用的模块的时钟信号通过一个控制信号gating住，降低这个模块的时钟信号翻转率，从而降低芯片功耗的一种技术
    2. power gating：在数字IC后端设计中，经常采用这个策略降低功耗。在后端实现过程中，加入MTCMOS来控制标准单元的开关。
    3. Multi vt cells ：在数字IC后端设计实现过程中，将某些不是critical path的地方尽量用HVT或者RVT，降低leakage。
    4. DVFS技术。DVFS（Dynamic Voltage and Frequency Scaling）动态电压频率调节本质上是一种低功耗技术，目的是根据的芯片当时的实际功耗需要设定工作电压和时钟频率，这样可以保证提供的功率既满足要求又不会过剩，从而可以降低功耗
    5. Well bias。这个方法可以动态调整偏置电压，从而实现降低功耗的目的。

# 4 各个公司的面试问题

## 4.1深圳中微电一面



## 4.2鼎信提前批

问的都是我自己的项目，主要在于算法方面，因此给我的是芯片算法工程师。

## 4.3地平线（原型验证）

因为HR的失误，投的是上海，面的是杭州，没问技术，也不符合岗位要求，一顿闲聊……

## 4.4 鼎信终面

参加什么社团活动，成绩怎么样之类的，10分钟结束

## 4.5 合肥宏晶微电子

技术面：

1. 你为什么转验证
2. 三个线程应该怎么办？fork join有哪三种？如果加上wait fork 之后运行情况有啥变化？
3. 简述UVM的工厂机制
4. 你的功能覆盖率为啥只有95%,功能点是自己写的嘛，都验了哪些功能？
5. 状态寄存器，如果状态是2，你采样的是1，怎么判断这个值是否正确？
6. 你这个soreboard怎么检查数据，如果出来一个包，怎么检查。。。问了一堆，你的scoreboard写了多久，考虑了什么问题没
7. 优先级的检查，如果三个上行的优先级是123,1为高优先级，2为低优先级，从formater中出来应该是什么样子的，如果连续发送又是什么样子的
8. 如果你的上行数据比较快，下行数据慢，会怎么样，应该怎么办？我：用FIFO吧，那你的FIFO的空满状态怎么呈现到接口上？
9. 看你这个断言覆盖率100%，描述一下你写的比较复杂的断言
10. SV中的interface的clock blocking的功能，如果clock blocking定义在下沿，最后的结果应该是什么样？
11. 动态数组和联合数组的区别？

HR面：

优点、缺点、做过最有成就感的事儿……

## 4.6中感微电子

问的自己的项目以及实习经历，并且让画项目框图并讲解

MCDF方面问写过SV没，用啥看的波形，以及写过模块，在VCS跑过没，还有就是几个数组的区别

## 4.7 飞腾公司

1. 看你参加了很多比赛，介绍一下你的IEEE全球极限编程
2. 介绍一下MCDF，你的DUT的功能，搭建的验证平台以及怎么测的
3. 说一下你的reference model和scoreboard是怎么实现的，reference model是实现了DUT的全部功能嘛，scoreboard 具体的怎么收数据，怎么比对数据?reference model 的包的速度和DUT的一样嘛？
4. reference model的实现

用tlm\_fifo进行数据传输，用一个取一个

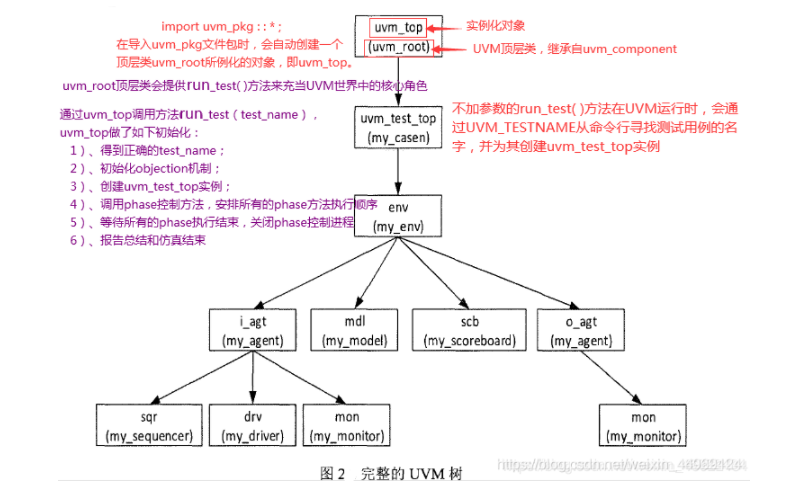
1. scoreboard 的实现

model通过tlm传递给scoreboard,monitor也传递一个句柄。就可以比较了

1. UVM从哪里启动，接口怎么传递到环境中

UVM的启动

总结：1）在导入uvm\_pkg文件时，会自动创建UVM\_root所例化的对象UVM\_top，UVM顶层的类会提供run\_test()方法充当UVM世界的核心角色，通过UVM\_top调用run\_test()方法。 2）在环境中输入run\_test来启动UVM验证平台，run\_test语句会创建一个my\_case0的实例，得到正确的test\_name 2）依次执行uvm\_test容器中的各个component组件中的phase机制，按照顺序， 1.build-phase（子顶向下构建UVM 树） 2.connet\_phase(子低向上连接各个组件) 3.end\_of\_elaboration\_phase 4.start\_of\_simulation\_phase 5.run\_phase() objection机制仿真挂起，通过start启动sequence（每个sequence都有一个body任务。当一个sequence启动后，会自动执行sequence的body任务）,等到sequence发送完毕则关闭objection，结束run\_phase()（UVM\_objection提供component和sequence共享的计数器，当所有参与到objection机制中的组件都落下objection时，计数器counter才会清零，才满足run\_phase()退出的条件（UVM入门P45）） 5.执行后面的phase



接口怎么传递到验证环境中（uvm\_config\_db）

1. 传递virtual interface到环境中；
2. 配置单一变量值，例如int、string、enum等；
3. 传递配置对象（config\_object）到环境；
4. 传递virtual interface到环境中；
5. 虽然SV可以通过层次化的interface的索引完成传递，但是这种传递方式不利于软件环境的封装和复用。通过使用uvm\_config\_db配置机制来传递接口，可以将接口的传递与获取彻底分离开。
6. 接口传递从硬件世界到UVM环境可以通过uvm\_config\_db来实现，在实现过程中应当注意：
7. 接口传递应发生在run\_test()之前。这保证了在进入build\_phase之前，virtual interface已经被传递到uvm\_config\_db中。
8. 用户应当把interface与virtual interface区分开来，在传递过程中的类型应当为virtual interface，即实际接口的句柄。
9. 配置单一变量值，例如int、string、enum等；

在各个test中，可以在build\_phase阶段对底层组件的各个变量加以配置，进而在环境例化之前完成配置，使得环境可以按照预期运行。

1. 传递配置对象（config\_object）到环境；

在test配置中，需要配置的参数不只是数量多，可能还分属于不同的组件。对这么多层次的变量做出类似上边的单一变量传递，需要更多的代码，容易出错且不易复用。如果整合各个组件中的变量，将其放置在一个uvm\_object中，再对中心化的配置对象进行传递，将有利于整体环境的修改维护，体改代码的复用性。

1. UVM的优势，为什么要用UVM

UVM 其实就是 SV 的一个封装，将我们在搭建测试平台过程中的一些重复性和重要的工作进行封装，从而使我们能够快速的搭建一个需要的测试平台，并且可重用性还高。但是UVM又不仅仅是封装。

1. 说一下ref类型，你用到过嘛

Ref参数类型是引用

1. 向子程序传递数组时应尽量使用ref获取最佳性能，如果不希望子程序改变数组的值，可以使用const ref类型
2. 在任务里可以修改变量而且修改结果对调用它的函数随时可见。
3. 还有代码覆盖率和功能覆盖率90%多，为什么没有100%，你是怎么实现的
4. 请说一下APB的读写操作以及AHB的Hburst

AHB基本信号

HADDR:32位系统地址总线。

HTRANS:M指示传输状态，NONSEQ、SEQ、IDLE、BUSY。。HWRITE:传输方向1-写，0-读。

HSIZE:传输单位。

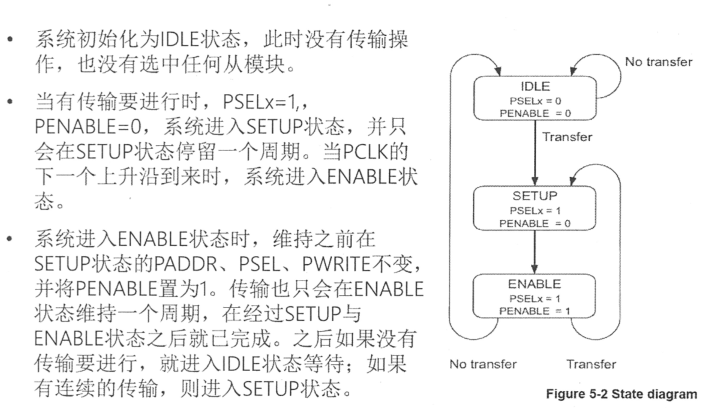
HBURST:传输的burst类型，SINGLE、INCR、WRAP4、INCR4等。

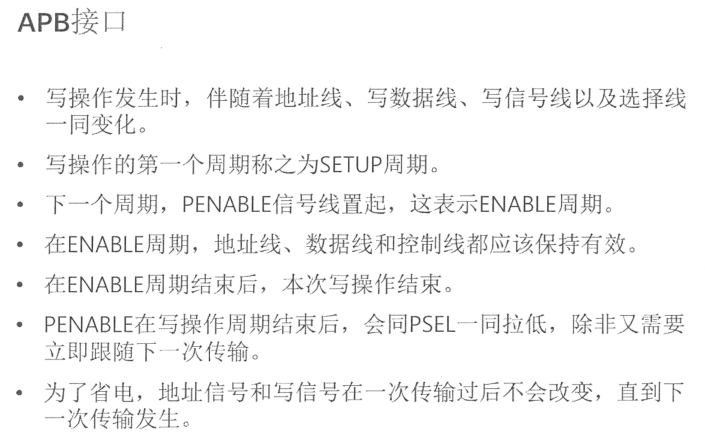
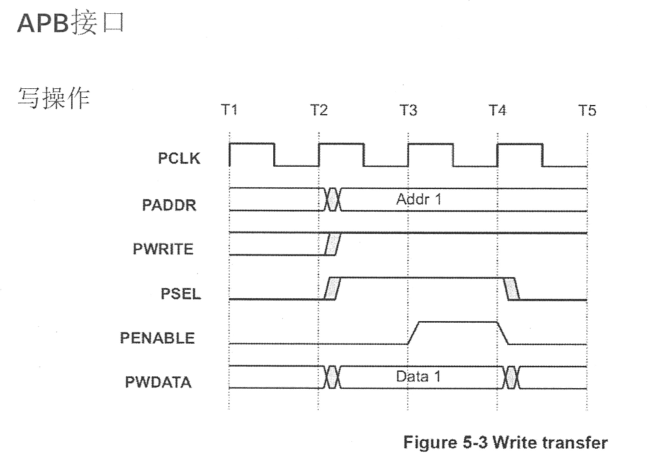
HWDATA:写数据总线，从M写到S。

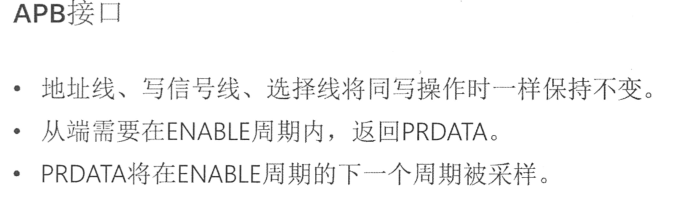
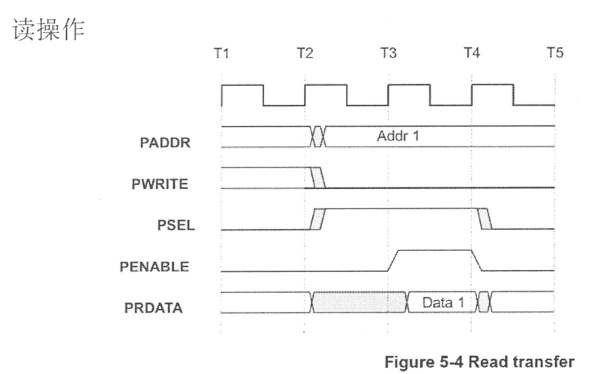
HREADY:S应答M是否读写操作传输完成，1-传输完成，0-需延长传输周期。

HRESP:S应答当前传输状态，OKAY、ERROR、RETRY、SPLIT。

HRDATA:读数据总线，从S读到M。







## 4.8 中科芯58所

1. 介绍一下自己，介绍一下项目
2. FIFO的深度是多少？位宽是多少？仲裁的算法是什么？
3. 参考模型是自己写的还是提供的，仲裁的话怎么在参考模型实现。参考模型怎么获取寄存器的值？
4. 怎么配置RTL寄存器，通过平台的寄存器模型配置寄存器，寄存器模型里边的方法，写一个寄存器或读一个寄存器怎么实现
5. 寄存器怎么实现前门访问和后门访问，怎么后门访问的路径怎么配置
6. 寄存器的总线协议是什么，说一下APB都有哪些信号，怎么实现读操作
7. 参考模型的monitor是怎么实现的，是人家提供的嘛，你做了哪些部分
8. 断言覆盖率是怎么写的，断言的分类，断言的实现是在哪一个模块实现的
9. 代码覆盖率和功能覆盖率，说一下对他们的理解，分析代码覆盖率用什么工具
10. 分析功能覆盖率是怎么分析的，怎么收集的，怎么调用covergroup
11. 说一下component和object的区别，item是component还是object
12. Virtual sequencer 和sequencer区别
13. 平台往里边输入数据的话怎么输入sequence， sequence，sequencer，driver之间的通信
14. 做软件为什么想转验证

## 4.9乐鑫提前批

1. 举例VCS的基本命令
2. APB的写操作、读操作，什么情况下APB开始写
3. APB的monitor是怎么写的
4. AHB是流水的嘛，是几级流水，如果这一拍是地址，且有数据，那么下一拍的地址能改嘛
5. 写一个100MHz的时钟
6. Reg和wire的区别，并写代码让判断值应该是啥
7. Logic和wire的区别，两者可以转换嘛
8. 用你擅长的原因找出1:100的质数

## 4.10中兴通讯

1. 介绍一下自己
2. APB协议
3. AHB TO APB你都做了哪些
4. 覆盖率都有哪些，代码覆盖率分为哪几种

代码覆盖率、功能覆盖率、断言覆盖率

代码覆盖率分为行覆盖率、分支覆盖率、状态机覆盖率、翻转覆盖率

1. Scoreboard是干嘛用的

是进行比较，通过从monitor里边采集的数据包，然后将APB的pkt和AHB的pkt进行比较，主要比较地址、数据、读写、优先级、传输字节数目、以及hresp和slverr

1. 为什么选验证

## 4.11集创北方

1. 介绍一下自己，介绍项目
2. APB协议分为哪些操作，介绍一下，你那个burst是怎么写的
3. 这个burst可以随机嘛，你是怎么随机的
4. 你的功能覆盖率是怎么写的，有哪些点，怎么检查你的功能覆盖率

我的功能覆盖率是通过设置covergroup和各个bin,对于交叉部分设置交叉覆盖组来定义的，目前的点主要是根据AHB TO apb的功能设置的，

1. 功能覆盖率和断言覆盖率的区别

功能覆盖率---与spec比较来发现，design是否行为正确，需要按verification plan来比较进度。用来衡量哪些设计特征已经被测试程序测试过的一个指标，首要的选择是使用更多的种子来运行现有的测试程序；其次是建立新的约束，只有在确实需要的时候才会求助于定向测试，改进功能覆盖率最简单的方法是仅仅增加仿真时间或者尝试新的随机种子。验证的目的就是确保设计在实际环境中的行为正确。设计规范里详细说明了设备应该如何运行，而验证计划里则列出了相应的功能应该如何激励、验证和测量

断言覆盖率---用于检查几个信号之间的关系，常用在查找错误，主要是检查时序上的错误，测量断言被触发的频繁程度。

1. 你的断言覆盖率写了哪些？（AHB TO apb Bridge的流程）
2. 写了PACTIVE无效时。没有APB传输发生
3. PACTIVE有效时，必定有APB传输发生
4. 你那个Arbiter是怎么写的

我的Arbiter本身是没有写什么具体的东西的，就是一个pkg，里边包括trans,sequencer,monitor,driver,agt等，我对仲裁的实现重点是在reference model和scoreboard里边，这个部分通过设置chanl\_id实现仲裁。

1. 寄存器是自己写的嘛

自己写了一部分，是按照一个模板写的。

1. MCDF的验证怎么知道你自己的数据对不对
2. 如果我这三个chnl的数据的优先级高，低，低，是不是一直发送高，那我低的就不发送了？
3. 为什么选验证，IC设计流程也即ASIC设计流程

芯片架构-RTL设计-功能仿真-综合&扫描链的插入（DFT）-等价性检查-形式验证-静态时序分析（STA）-布局规划-布局布线-布线图和原理图比较-设计规则检查-GDII

1. 一个最简单的八位加法器应该怎么验证？才有完备性？
2. 首先拿到这个加法器时先看基本逻辑，比如有没有复位信号，有复位信号，验证复位功能是否正常
3. 然后从功能角度，这个加法器是只用来做正数加正数还是负数的运算也可以，所以应验证其正数+正数，正数+负数以及负数+负数
4. 从数据随机的角度来说00001111这种边界情况需要测，还有随机出现数据
5. 最后还要考虑模块出错的情况，如果我们的数据源出现了X值，这个加法器模块会怎么处理
6. 加法器如果没有带时序逻辑进行采样，就是一个组合逻辑电路，还可以验证是否出现毛刺（待定）
7. 在一个CPU系统中，有2个master通过一个2\*1的AXI总线访问一个Slave，简述如何构造验证场景来进行验证，并保证验证的完备性。
8. 第一个就是最基本的测试，就是单master访问slave用master0访问slave，然后用master1访问slave，然后每个master访问slave的时候要对axi的所有协议都要进行测试，这是最基本的测试。
9. 第二个的话可以测一下他那个地址边界，如果你这个slave是memory的话，那我访问你这个memory地址范围以外的话，你是不能正常读写的。
10. 第三个就是仲裁访问测试，两个master同时访问slave的时候，要看谁的优先级更高，这个要测一下。
11. 最后一个测试要测的是数据的一致性完整性，Master写进去的数和读出来的数必须要一样，而且master0写进去的数，然后master1读出来也必须要一样。
12. FIFO作为一个通用的逻辑单元模块，应该怎么测试？

FIFO深度检查

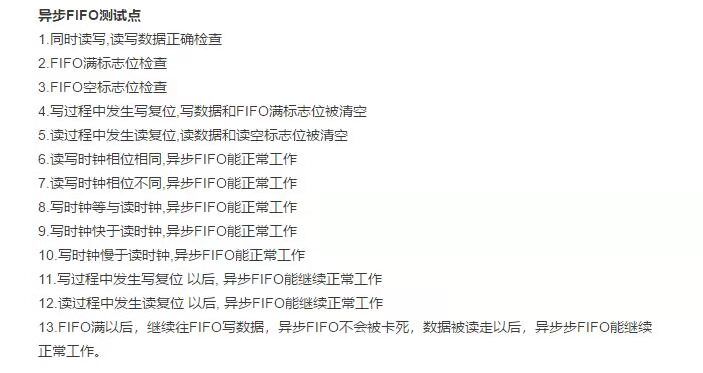
FIFO位宽检查

FIFO空满判断

FIFO满后再继续写数据，或者是FIFO空后再继续读数据，会发生什么

FIFO清空检查，软件复位后可以清空FIFO嘛

1. 异步FIFO的测试点



## 4.12星宸科技线下：一面技术面+HR面

1. 刚开始就是问了一下基本情况
2. 拿着线下的笔试题，让我用matlab写那个 maxpiol，简单的写出来了，复杂的不大会
3. 然后就闲聊了一下，都是开放性问题，可以自由发挥
4. HR面也很细,HR很好，很认真，问的比较全，但基本上都是常见的几点，接下来还有个技术二面，应该比较复杂

## 4.13地平线（上海）线上（一面）

1. 介绍一下MCDF，你是怎么打包的

根据写入寄存器的值，及时更新reference model中寄存器的值。然后根据reference model中的寄存器中决定长度的值，开辟出一个相应长度的空间。

1. chnl的优先级怎么验证，如果高，低，低出来之后是什么样子？只送优先级为高的数据嘛

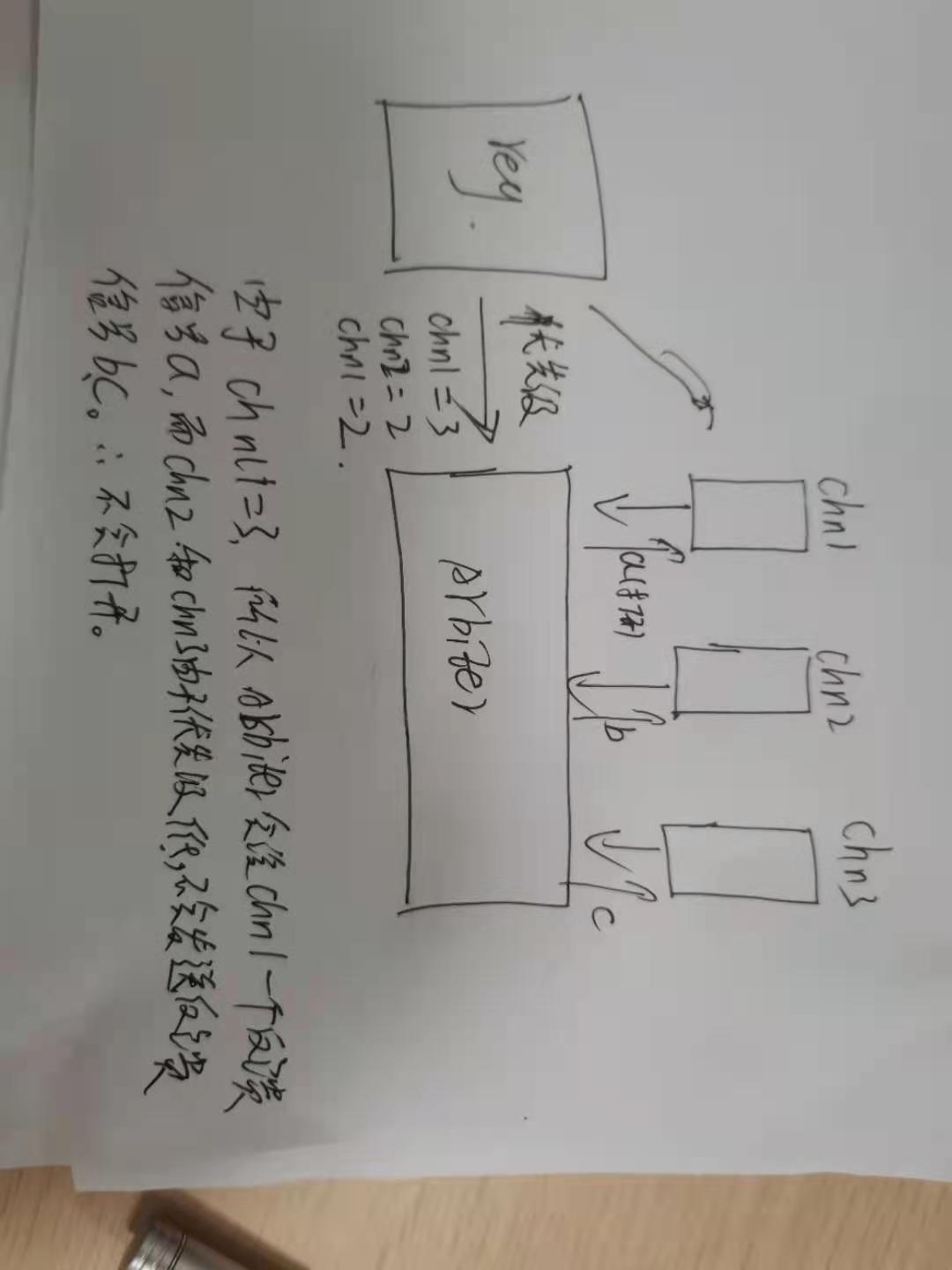
三个chnl同时请求的话，先发送优先级高的chnl的数据，后发送优先级低的数据，，如果优先级一样，就轮询。优先级检查，是通过scoarboard中的do\_arbiter\_priority\_check函数检查的，每一个id对应一个优先级数据大小，优先级高的会先打开abiter的通道，仲裁通过。（可以看看那个优先级检查函数）

当出现高低低时，高优先级先通过，然后两个低优先级的会采用轮询机制（DUT本身功能），轮询时是chn1＞chn2＞chn3

1. 怎么检查优先级是否正确呢？你怎么知道你的数据对不对

写个参考模型，直接比较，将参考模型出来的数据和DUT出来的数据进行比较，只要参考模型写得好，全部都可以直接比较，例如可以模拟三个fifo，再模拟一个仲裁fifo，这个仲裁fifo根据寄存器配置的优先级，选择性的从三个fifo中拿数据，然后按照dut发送出来数据的格式发送出来，验证环境里边模拟FIFO的方法有很多，例如队列，mailbox或者是tlm fifo

1. 你的arbiter是怎么写的？



根据寄存器写入的优先级的值，把优先级大的通道打开，然后从通道内拿数据，Arbiter根据谁的优先级大，给谁回应信号让其打开，优先级低的收不到回应信号。这个回应信号相当于rsp告诉chnl我要接收你的信号

1. 如果打包的时候，只有一部分chnl1的数据，还会不会补chnl2的数据？这个包里的数据会不会是chnl1+chnl2+chnl3？

不会补，整形器必须完整发送某一个数据通道的数据包后，才可以转而准备发送下一个数据包，在发送数据包期间，fmt\_chnl\_id和fmt\_length应该保持不变，直到数据包发送完毕

1. 代码覆盖率和功能覆盖率有什么区别？
2. 请描述一下AHB协议，你是怎么写的，AHB的第一拍是做什么，第二拍做什么？Hready为高是什么操作？
3. 描述一下APB协议，APB2和APB3有什么区别?除了多一个信号还有别的区别嘛？

APB的读写操作，重点

APB2.0和APB3.0的差别：APB3.0提供了一个低功耗的接口，并降低了接口的复杂性。且APB3比APB2增加了两个信号：

PREADY：来扩展APB传输，主要是增加延时；

错误信号PSLVERR： 来指示传输失败

APB3和APB4的差别：

增加了PROT和PRSTB两个信号。

PPROT一种保护信号，用于支持APB上的非安全交易和安全交易。

PSTRB一个写选通信号，用于在写数据总线上进行sparse data transfer（稀疏数据传输）。APB4用的比较少。

1. OPP的特性？多态看的是基类句柄还是对象？

封装、继承和多态

封装:通过将一些数据和使用这些数据的方法封装在一个集合里，成为一个类。

继承:允许通过现有类去得到一个新的类，且其可以共享现有类的属性和方法。现有类叫做基类，新类叫做派生类或扩展类。

多态:得到扩展类后，有时我们会使用基类句柄去调用扩展类对象，这时候调用的方法如何准确去判断是想要调用的方法呢?通过对类中方法进行 virtual声明，这样当调用基类句柄指向扩展类时，方法会根据对象去识别，调用扩展类的方法，而不是基类中的。而基类和扩展类中方法有着同样的名字，但能够准确调用，叫做多态。

1. Sequencer和driver之间的通信，是什么端口，怎么通信？

如果driver需要item，会使用get\_next\_item,拿到之后再使用item\_done，是TLM通信端口，具体通信是：driver是port，sequencer是export

drv\_i.seq\_item\_port.connect(sqr\_i.seq\_item\_export)

1. Monitor和scoreboard之间是怎么通信的，analysis port的特点

analysis port：一端对多端

* 除了端对端的传输，在一些情况下还有多个组件会对同一个数据进行运算处理。
* 如果这个数据是从同一个源的TLM端口发出到达不同组件，这就要求该种端口可以满足从一端到多端的需求。
* 如果数据源端发生变化需要通知跟它关联的多个组件时，我们可以利用软件的设计模式之一观察者模式(observer pattern）来实现。

广播observer pattern的核心在于:

第一，这是从一个initiator端到多个target端的方式。

第二，analysis port采取的是"push"模式，即从initiator端调用多个target端的write()函数来实现数据传输。

1. 说一下phase机制的特点

什么样的时间做什么样的事情，九大phase机制只有

1. Phase机制怎么同步？例如objection机制没有drop objection会发生什么情况？例如2个component。A和B，A啥事都没干，B没有drop objection，那么A会不会跳转到后边？
2. 二分频是怎么写的？

偶数分频：写一个0：（N-1）计数器，再写个分频器，在N/2-1和N-1的时候翻转。

奇数分频：写两个计数器和分频器（上升沿计数，下降沿计数，上升沿分频，下降沿分频），然后再将分频结果进行或运算。

1. 阻塞和非阻塞及其应用

阻塞赋值语句（“=”）和非阻塞赋值语句（“<=”）

阻塞：一般对应电路中的组合逻辑赋值，等号右端的结果会立刻赋值给左端。

非阻塞：一般对应电路中的时序逻辑赋值，等号右端的结果不会立刻赋值给左端。

在always语句中，阻塞赋值等号左端的参数如果参与该模块的其他运算，则按照赋值后的结果参与运算，而非阻塞赋值等号左端的参数依旧按照未赋值前的结果参与运算。

Assign语句中用阻塞赋值，

组合电路用阻塞赋值，时序电路用非阻塞赋值

1. 你的数据是什么样子的
2. Chnl的协议，formater的协议

## 4.14星宸科技二面

1. 你对UVM有什么了解
2. AHB的driver和monitor是怎么写的，你这个数据是从哪里来到哪里去
3. APB的monitor和AHB的monitor之间的区别
4. SV和verilog的区别
5. Sequence和sequencer之间的通信
6. Sequencer和driver之间的通信
7. 还有组件之间的通信
8. 寄存器的前门访问和后门访问
9. Opp的三个特性

## 4.15联发科一面

1. 一分钟英文自我介绍
2. 你那个桥是干啥的，桥的作用
3. 你那个AHB to APB Bridge如果一个master对多个slave应该怎么办？
4. MCDF验证环境里有几个agent
5. 因为笔试做的太好，还有C和python的题以及算法和数据结构（二叉树、栈）哭了
6. 讲一下component和object
7. 都有哪些组件
8. Factory机制

7.28集创北方二面

Questasim是哪家公司的？

## 4.16 zeku一面

1. 了解本科和研究生学过的课程
2. IC设计的流程和工具
3. 阻塞和非阻塞
4. 寄存器和锁存器的区别
5. 动态数组、队列、定宽数组、关联数组
6. 你的优先级是怎么测的
7. 硬件DUT和验证环境是怎么关联的
8. 对SV中约束的了解

## 4.17华为一面

1. 笔试题中异步电路的理解，怎么同步
2. 对形式验证的了解
3. 然后问了一下成绩，比赛之类的
4. 然后开始问项目，画项目框图，讲解
5. 寄存器读写怎么测试的
6. 写一个串并转换的reference model，输入是1bit，输出是256bit，在valid有效时输出

## 4.18 展锐一面（有一到三轮技术面）

1. 介绍一下自己
2. 把写的项目都问了一下，重点问的是MCDF
3. 寄存器的复位是怎么测的，寄存器的读写是怎么测的
4. 代码覆盖率为什么没达到100%，实际中应该是怎么样的
5. 功能覆盖率测了哪些功能点，是怎么写的，为什么没达到100%
6. 断言覆盖率写了哪些
7. 动态数组和关联数组的区别
8. 三个Fork join的区别
9. Task和function的区别
10. Phase机制有哪几个，build phase是自顶向下，对这个自顶向下有什么了解嘛

## 4.19商汤一面

1. 为什么转验证？
2. 你之前保研的时候怎么不选IC。想来北京还是上海，之前保研的时候怎么没来北京
3. SV里边find队列和find index队列应该返回什么?这个没敢懵，是不是返回数和索引啊？
4. 有没有用过zadx(好像是这个东西)?
5. 怎么验证一个东西
6. 用过断言嘛？写一个断言，a为高的时候，b为高，还有a为高的时候，下一个周期b为高
7. 对商汤的了解，对AI的了解

## 4.20联芸一面

1. 介绍自己，还有数字样机的项目，因为他们也是做雷达的，对我这个项目了解的多一些。
2. Scoreboard是什么时候比较
3. Reference model是怎么写的
4. 如果比较的时候rm有数据，而dut没有数据，那么你应该怎么比较
5. 你的reg用的是什么端口
6. Rm用的是什么端口
7. TLM通信有几种方式

## 4.21 展锐二面

1. 问电子侦察数字样机的项目
2. 事件触发，wait 和@的区别
3. Fork join的三种
4. 约束的几种形式
5. 哪些继承于component
6. 哪些继承于object

## 4.22联发科二面

聊天

## 4.23 商汤二面

【1】自我介绍

【2】项目中遇到什么问题是怎么解决的

【3】对SV中约束的理解

【4】phase机制的理解

【5】rand和randc的理解

【6】对进程的理解，我刚开始还以为是mailbox那几个，后来发现是fork join那三个的区别

【7】ref的用法

【8】寄存器模型的使用，例如前门访问是怎么用的，后门访问呢，adapter是干嘛的

【9】sequencer和driver之间的握手机制

## 4.24禾赛科技一面

【1】自我介绍

【2】你自己的项目是硬件的嘛，用什么语言写的，有没有设计一个模块

【3】学过数电，模电，半导体之类的嘛

【4】用过C或者是C++嘛，你以后有什么想法嘛

【5】MCDF和AHB都是网上的课程是嘛

【6】你AHB是怎么验的，验证环境是怎么搭的

【7】描述一下AHB协议

【8】给你一个模块应该怎么验证

【9】讲一下MCDF项目，你是怎么验证的，怎么打包

【10】用MOS管如何搭与非门和或门

## 4.25联发科三面

【1】将一下你那个AHB的trans

【2】AHB的burst

【3】举例说明这些操作相关的协议

【4】AHB和APB项目的测试点，你是怎么制定的

【5】你硕士期间的项目，遇到啥问题

【6】怎么解决的

## 4.26紫光同芯一面

【1】介绍一下你自己

【2】你的DUT的数据和UVM验证平台的数据会有时间差，这个你应该怎么办，你怎么知道比对的数据就是那个时间点的数据

【3】发各个chnl的数据的时候，可能chnl1发了几个trans，还有间断，然后再发chnl2的，这个你应该怎么仲裁

【4】AHB的协议和APB的协议特点，以及两者对比

【5】AHB总线的速率为什么比APB速率高？

## 4.27 奥比中光一面

1. 介绍一下寄存器怎么集成，寄存器怎么运用到环境中
2. 还有你这个reg2bus和bus2reg的区别
3. 只读寄存器怎么验证
4. Sequence的启动
5. 如何关闭约束
6. 队列的使用方法，以及push back和pop front的区别
7. Rand 和randc的区别
8. Uvm的phase机制，各个component之间的phase是怎么运行的
9. 组件之间的通信机制，analysis port和其它的区别
10. AHB的传输类型，说一下4回环突发的传输应该是怎么样的，这个回环边界怎么确定

## 4.28 寒武纪一面，正式批有三面，技术面，总监面（技术面少一些，定岗），HR面

【1】自我介绍，问提到的各个比赛情况，是什么内容，你在里边充当什么样的角色

【2】简述一下AHB都有哪些信号，Htrans是怎么用的

【3】你这个测试点怎么制定的，都制定了哪些

【4】介绍一下MCDF

【5】你这个是怎么验证的，寄存器模型有了解嘛

【6】寄存器都验了哪些？如果寄存器设置的数和你采集的数不一样怎么办

【7】只读寄存器应该怎么验证

【8】SV的优势，为什么要用SV进行验证

【9】深拷贝和浅拷贝

【10】为什么要用virtual

【11】为什么选验证

【12】你参与了很多学生工作，有遇到什么困难吗

【13】有没有压力特别大的时候，你是怎么解决的

## 4.29 汇顶一面

1. 介绍一下自己，讲讲你在做这些验证项目中对verilog、systemverilog以及UVM的了解
2. 这两个项目的driver有什么不同，你的数据怎么送进去的
3. AHB的写过程和读过程，简述一下是怎么读的
4. AHB的monitor是怎么写的
5. Interface是怎么传递的，config\_db的参数是什么样子的
6. DUT和interface的连接
7. Sequence是怎么启动的，你这有几个sequence，比如说3个通道的sequence怎么处理的，是有3个sequence嘛，有几个sequencer呢，怎么发送数据
8. 说一下你对流水线操作的理解，是怎么流水的
9. 说一下对burst的理解
10. 你的AHB to APB的reference model是怎么写的
11. 对TLM通信的了解，你sequencer和driver之间用的是什么端口
12. 什么是阻塞和非阻塞端口，那driver用的是阻塞还是非阻塞？
13. 对寄存器模型的了解

## 4.30 ARM一面

1. 介绍自己
2. APB和AHB的协议
3. 如何完成APB和AHB之间的握手
4. AHB怎么完成burst传输
5. AHB to APB Bridge的测试点
6. MCDF中的仲裁是怎么仲裁的
7. 如果优先级一样采用轮询机制的话，你对轮询机制的理解
8. 同步FIFO的测试点，FIFO的空满是怎么判断的
9. 收集覆盖率了嘛，都有哪些？代码覆盖率都有哪些
10. MCDF的测试点
11. 断言覆盖率都检查了哪些
12. 覆盖组怎么写，如果有128bit的数据怎么写covergroup，cross怎么写？触发有@和wait，这两种的区别
13. 寄存器和锁存器的区别
14. 什么是建立时间和保持时间，如果违例怎么办？
15. 跨时钟域应该怎么处理

## 4.31 晶晨半导体

1. SV中循环都有哪几种？
2. OOP是什么？
3. AHB协议和APB协议，AHB中Hready是怎么回事儿，APB中psel和penable的作用
4. 一个子类可以有多个父类嘛？一个父类可以有多个子类嘛？在C++中呢？
5. Struct的作用，union是什么？struct和类的区别，struct和union的区别
6. 知道DPI嘛
7. 对UVM的了解，什么是phase机制，你常用的phase机制，都是什么作用？
8. Sequence是在哪个phase跑的
9. Sequence中信号的随机，参数的定义写在哪个函数中？
10. Run\_test是启动UVM平台，应该怎么跑testcase，怎么启动
11. C语言中指针是什么意思，怎么写数据
12. 如果给你一个交通灯，怎么写测试点