目录

1 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)

phase	函数/任务	执行顺序	功　能	典型应用
build	函数	自顶向下	创建和配置测试平台的结构	创建组件和寄存器模型，设置或获取配置
connect	函数	自底向上	建立组件之间的连接	连接TLM/TLM2的端口，连接寄存器模型和adapter
end_of_elaboration	函数	自底向上	测试环境的微调	显示环境结构，打开文件，为组件添加额外配置
start_of_simulation	函数	自底向上	准备测试环境的仿真	显示环境结构，设置断点，设置初始运行的配置值
run	任务	自底向上	激励设计	提供激励、采集数据和数据比较，与OVM兼容
extract	函数	自底向上	从测试环境中收集数据	从测试平台提取剩余数据，从设计观察最终状态
check	函数	自底向上	检查任何不期望的行为	检查不期望的数据
report	函数	自底向上	报告测试结果	报告测试结果，将结果写入到文件中
final	函数	自顶向下	完成测试活动结束仿真	关闭文件，结束联合仿真引擎

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

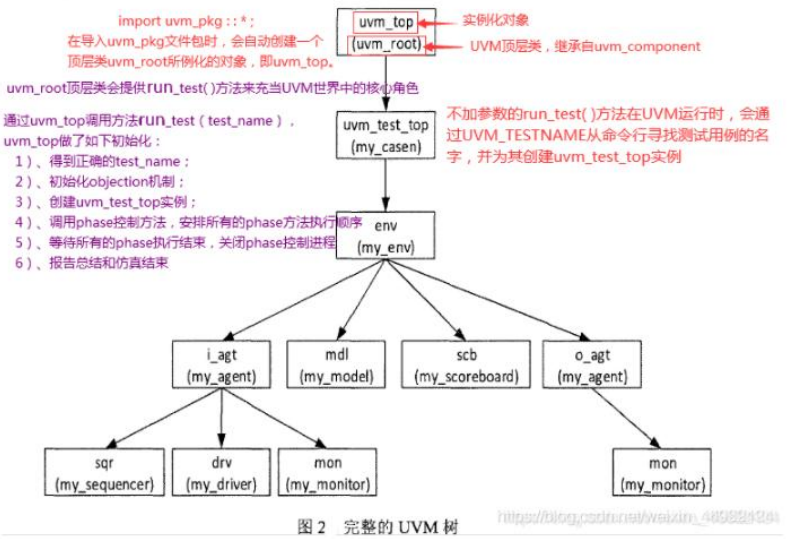


图 2 完整的 UVM 树

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；

1) 传递 virtual interface 到环境中；

a) 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。

b) 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：

c) 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。

d) 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。

2) 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。

3) 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

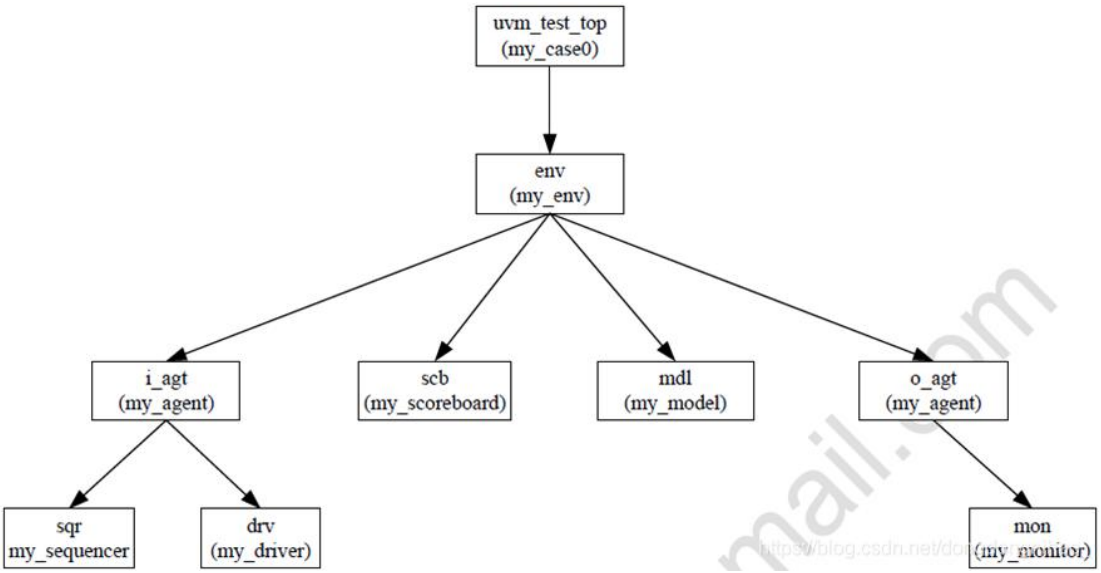
2) 在任务里可以修改变量而且修改结果对调用它的函数随时可见。

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。

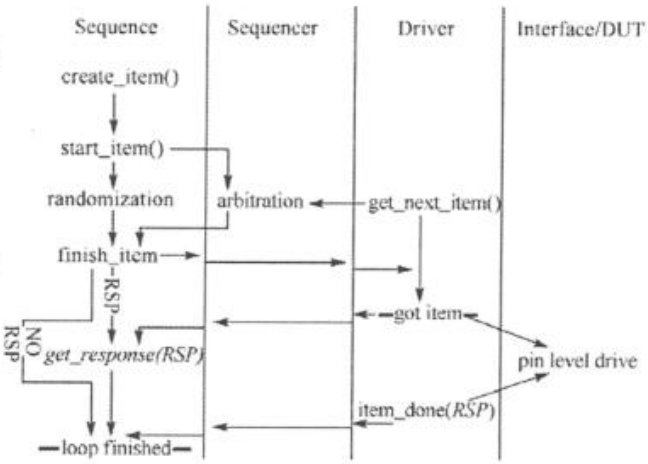
Env: 则相当于是一个特大的容器，将所有成员包含进去。

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

- 无论是sequence还是driver，它们通话的对象都是sequencer。当多个sequence试图要挂载到同一个sequencer上时，涉及sequencer的仲裁功能。
- 重点分析sequencer作为sequence与driver之间握手的桥梁，是如何扮演好这一角色的。
- 我们将抽取去这三个类的主要方法，利用时间箭头演示出完整的TLM通信过程。



- 对于sequence而言，无论是flat sequence还是hierarchical sequence，进一步切分的话，流向sequencer的都是sequence item，所以就每个item的"成长周期"来看，它起始于create_item()，继而通过start_item()尝试从sequencer获取可以通过的权限。
- 对于sequencer的仲裁机制和使用方法我们暂且略过，而driver一侧将一直处于"吃不饱"的状态，如果它没有了item可以使用，将调用get_next_item()来尝试从sequencer一侧获取item。
- 在sequencer将通过权限交给某一个底层的sequence前，目标sequence中的item应该完成随机化，继而在获取sequencer的通过权限后，执行finish_item()。
- 接下来sequence中的item将穿过sequencer到达driver一侧，这个重要节点标志着sequencer第一次充当通信桥梁的角色已经完成。
- driver在得到新的item之后，会提取有效的数据信息，将其驱动到与DUT连接的接口上面。
- 在完成驱动后，driver应当通过item_done()来告知sequence已经完成数据传送，而sequence在获取该消息后，则表示driver与sequence双方完成了这一次item的握手传输。
- 在这次传递中，driver可以选择将RSP作为状态返回值传递给sequence，而sequence也可以选择调用get_response(RSP)等待从driver一侧获取返回的数据对象。

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

```
a_high_then_b_high:assert property();          //a和b同时为高
property a_high_then_b_high;
    @(posedge clk)
        a|->b;
endproperty

property a_high_then_b_high;                   //a为高，下一个周期b为高
    @(posedge clk)
        a|=>b;
endproperty
```

立即断言和并行断言

可以将断言分为两种常见的类型：

- 立即断言（immediate assertion）。
  - 非时序的。
  - 执行时如同过程语句。
  - 可以在initial/always过程块或者task/function中使用。
- 并行断言（concurrent assertion）
  - 时序性的。
  - 关键词property用来区分立即断言和并行断言。
  - 之所以称之为并行，是因为它们与设计模块一同并行执行。

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

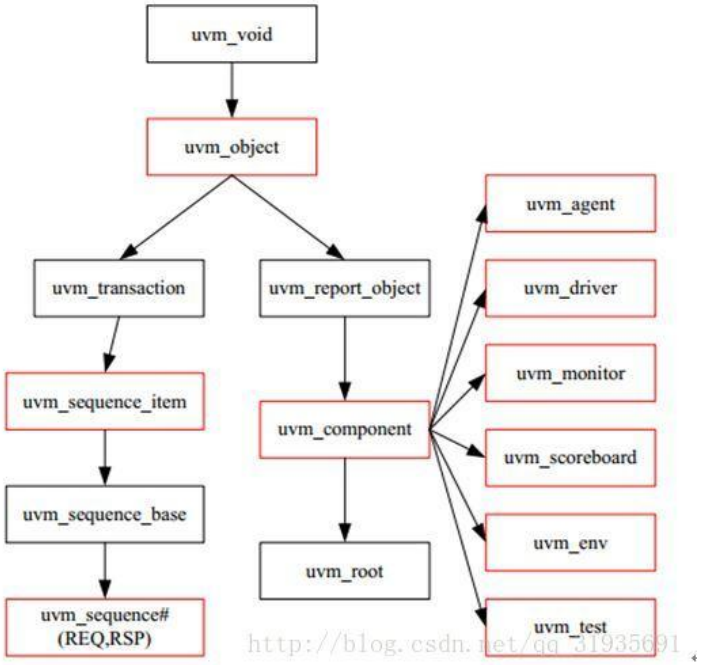
随机化是 SV 中极其重要的一个知识点，通过设定随机化和相关约束，我们可以自动随机想要的数据。

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



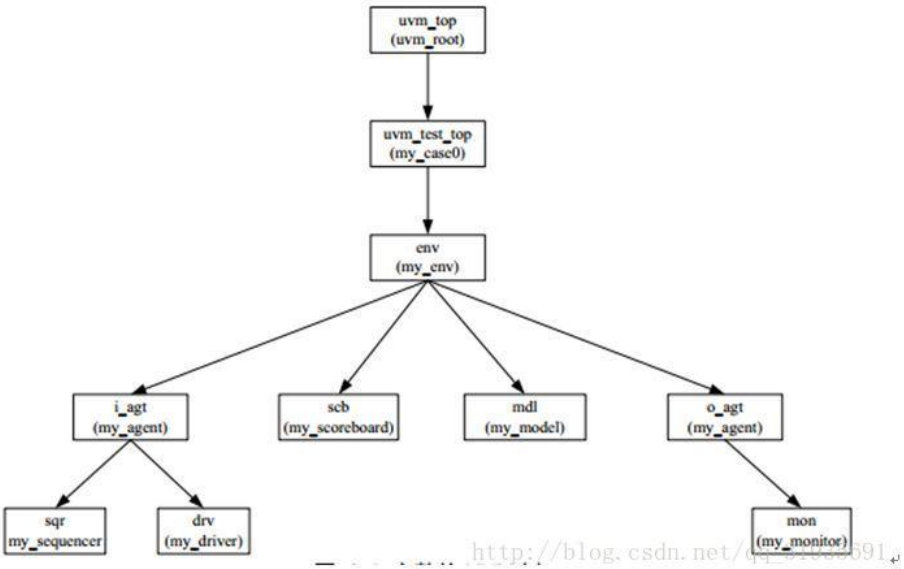
除了 driver、monitor、agent、model、scoreboard、env、test 之外全部用 uvm_object。

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

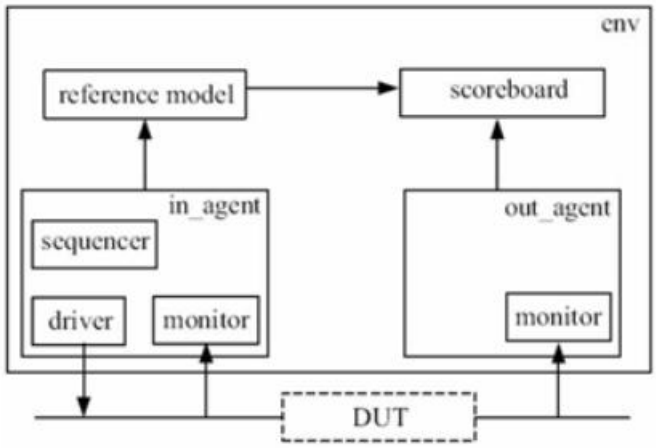
触发器：时钟触发，受时钟控制，只有在时钟触发时才采样当前的输入，产生输出。

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

一个典型的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

语法之$past构造

SVA提供了一个内嵌的系统任务"$past", 它可以得到信号在几个时钟周期之前的值。在默认情况下，它提供信号在前一个时钟周期的值。
结构的基本语法如下:
$past (signal name, number of clock cycles)
这个任务能够有效地验证设计到达当前时钟周期的状态所采用的通路是正确的。

```
property p20;
    @(posedge clk) (c && d) |->
    ($past((a&&b),  2,  e) == 1'b1);
endproperty

a20:  assert property(p20);
```

只有当控制信号"e"在任意给定的时钟上升沿有效时检验才被激活。

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:

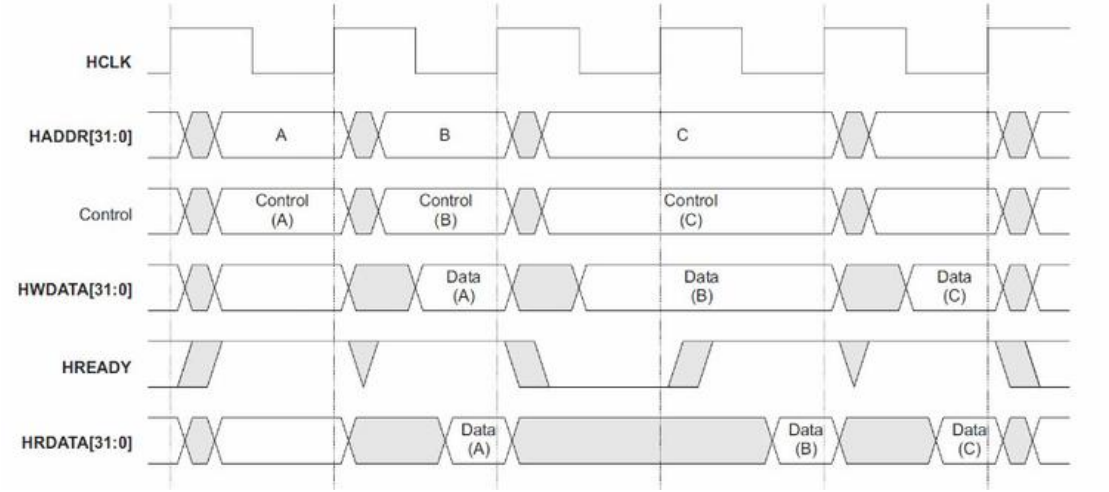
4) request & response 通信管道

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



扩展数据周期的一个负效应是必需延长相应的下一笔传输的地址周期。A和C为零等待传输，B加入了一个等待周期，因此相应的C地址周期要进行扩展。

第一个周期，master发起一个操作A，并驱动地址和控制信号；

第二个周期，slave收到了来自总线的请求，将HREADY信号拉高；

第二个周期上升沿后，master发现有操作B需要执行，并且检查到上一周期的HREADY为高，则发起第二个操作B；

第三个周期，master获取HREADY信号为高，表示操作A已经完成；

第三个周期上升沿后，master发现有操作C需要执行，并且检查到上一周期的HREADY为高，则发起第三个操作C；

第三个周期上升沿后，slave由于繁忙插入了一个等待状态，将HREADY拉低；

第四个周期，master获取HREADY信号为低，知道slave希望等待，于是master保持和上一拍一样的信号；

第四个周期，slave处理完了事务，将HREADY信号拉高，表示可以继续处理；

第五个周期，master获取HREADY信号为高，知道slave已经可以处理B操作；

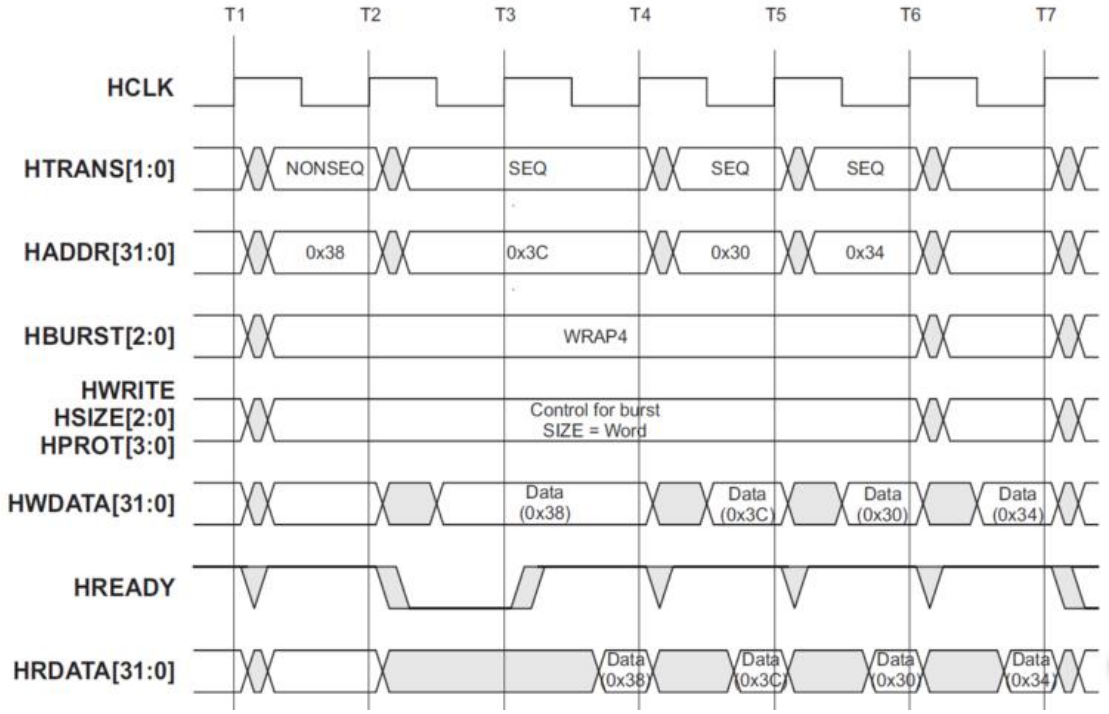
第五个周期上升沿后，B操作完成；

第六个周期上升沿后，C操作完成。

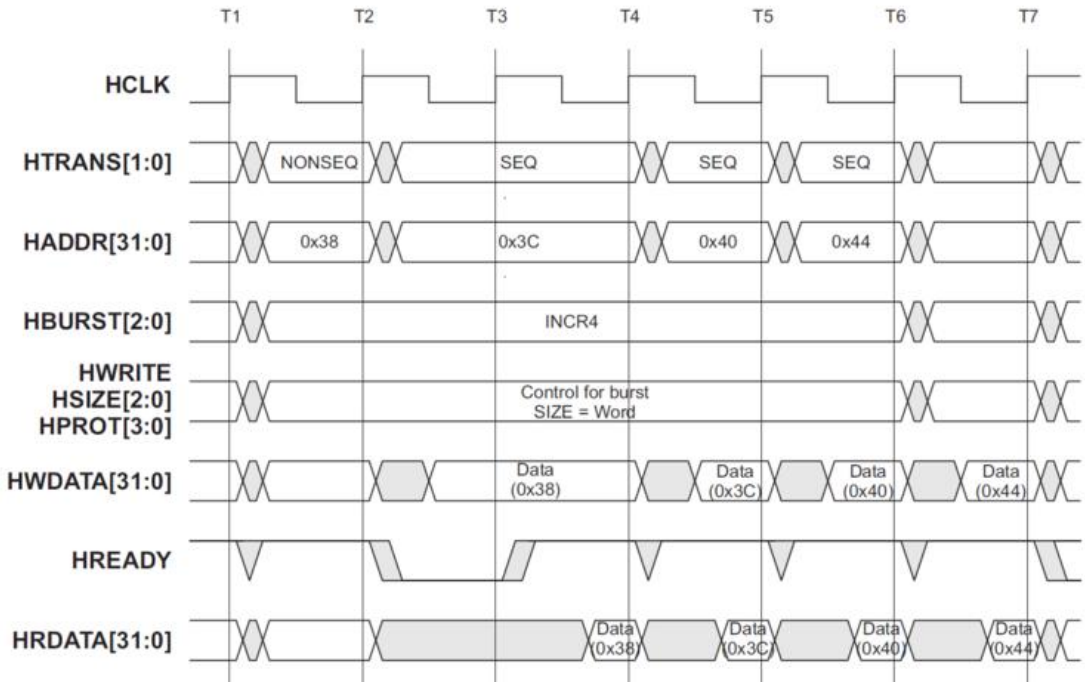
需要注意几点：

HREADY在一定程度上表示了slave的pipeline能力，在AHB中是2个pipe，也就是总线上最多存在2个未处理完的transfer。只有当总线上未完成的transfer少于2个时，master才能发起操作。

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

2 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

```
for i = 2 : 100  %外层循环，i的初值为2，终值为100
    for j = 2:100  %内层循环，j的初值为2，终值为100
        if(~mod(i,j))  % i除以j取余后再取反
            break; % 跳出循环
        end
    end
    if(j > (i/j)) %检查是否有其他除数
        fprintf('%d is prime \n',i); %输出素数
    end
end
```

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。

2.9 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

2.10 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

异步FIFO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.写时钟等与读时钟,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能继续正常工作
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，异步步FIFO能继续正常工作。

2.11 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。

2.12 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。

2.13 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(可以用组合电路输出，也可以时序电路输出)。

2.14 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。

2.15 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；

3 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

[1] 介绍一下项目，以及项目的验证框架
[2] 你的 Sequence 是怎么处理的，和 generator 和 driver 之间的通信
[3] Generator 继承于谁？
[4] 寄存器复位值是怎么检查的？reg to bus 和 bus to reg
[5] 寄存器读写是怎么检查的？前门访问，你怎么知道你验的对不对？
[6] 代码覆盖率为什么没满足 100%，没满足的话应该怎么办？
[7] 功能覆盖点是怎么找的，这个具有很大的主观性，你怎么确定自己找的就是自己验证的点？这能够覆盖完全嘛，应该怎么找功能覆盖点？
[8] 断言覆盖率是怎么写的，验了哪些？
[9] 你的数据包是怎么验的？你怎么知道对不对？
[10] 验证的时候为什么用 SV 而不用 UVM？
[11] 讲一下 callback 机制

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?

1) reference model 的实现
   用 tlm_fifo 进行数据传输，用一个取一个
2) 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

[4] 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

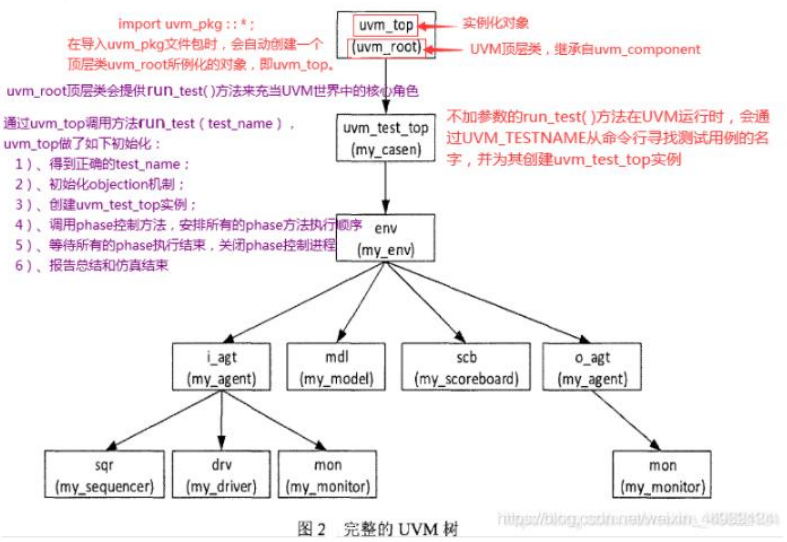


图 2　完整的 UVM 树

接口怎么传递到验证环境中（uvm_config_db）

4)　传递 virtual interface 到环境中；

5)　配置单一变量值，例如 int、string、enum 等；

6)　传递配置对象（config_object）到环境；

3)　传递 virtual interface 到环境中；

e)　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。

f)　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：

g)　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。

h)　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。

4)　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。

4)　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。

[5]　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。

[6]　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。

[7] 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

[8] 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。

- 系统初始化为IDLE状态，此时没有传输操作，也没有选中任何从模块。
- 当有传输要进行时，PSELx=1,，PENABLE=0，系统进入SETUP状态，并只会在SETUP状态停留一个周期。当PCLK的下一个上升沿到来时，系统进入ENABLE状态。
- 系统进入ENABLE状态时，维持之前在SETUP状态的PADDR、PSEL、PWRITE不变，并将PENABLE置为1。传输也只会在ENABLE状态维持一个周期，在经过SETUP与ENABLE状态之后就已完成。之后如果没有传输要进行，就进入IDLE状态等待；如果有连续的传输，则进入SETUP状态。

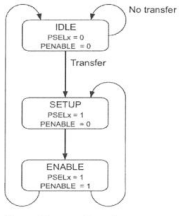


Figure 5-2 State diagram

APB接口

写操作

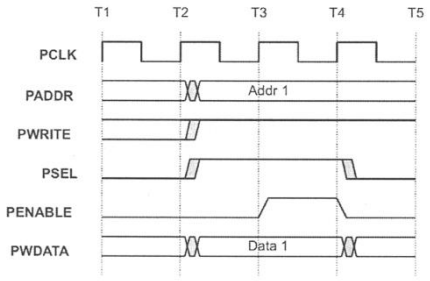


Figure 5-3 Write transfer

APB接口

- 写操作发生时，伴随着地址线、写数据线、写信号线以及选择线一同变化。
- 写操作的第一个周期称之为SETUP周期。
- 下一个周期，PENABLE信号线置起，这表示ENABLE周期。
- 在ENABLE周期，地址线、数据线和控制线都应该保持有效。
- 在ENABLE周期结束后，本次写操作结束。
- PENABLE在写操作周期结束后，会同PSEL一同拉低，除非又需要立即跟随下一次传输。
- 为了省电，地址信号和写信号在一次传输过后不会改变，直到下一次传输发生。

读操作

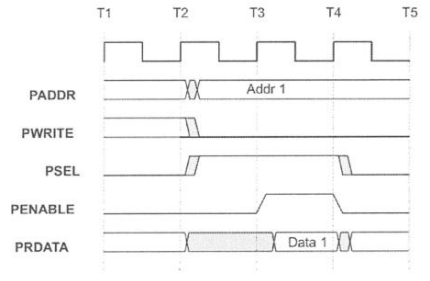


Figure 5-4 Read transfer

APB接口

- 地址线、写信号线、选择线将同写操作时一样保持不变。
- 从端需要在ENABLE周期内，返回PRDATA。
- PRDATA将在ENABLE周期的下一个周期被采样。

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

[5] 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

[6] 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，

[5] 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。

[6] 你的断言覆盖率写了哪些？（AHB TO apb Bridge 的流程）

1、 写了 PACTIVE 无效时。没有 APB 传输发生

2、 PACTIVE 有效时，必定有 APB 传输发生

[7] 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。

[8] 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。

[9] MCDF 的验证怎么知道你自己的数据对不对

[10] 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？

[11] 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

[12] 一个最简单的八位加法器应该怎么验证？才有完备性？

6) 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
7) 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
8) 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
9) 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
10) 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）

[13] 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。

[14] 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

[15] 异步 FIFO 的测试点

异步FIFO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.写时钟等与读时钟,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能继续正常工作

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，异步步FIFO能继续正常工作。

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。

[2] 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）

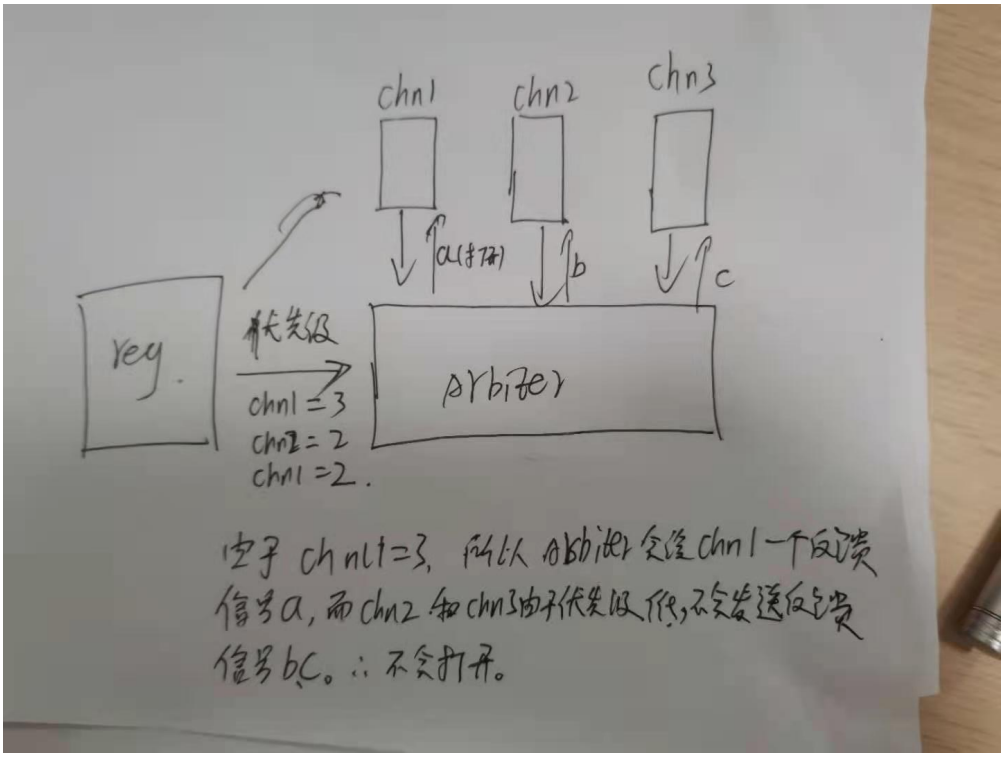
当出现高低低时，高优先级先通过，然后两个低优先级的会采用轮询机制（DUT 本身

功能），轮询时是 chn1 > chn2 > chn3

[3] 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

[4] 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

[5] 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

[6] 代码覆盖率和功能覆盖率有什么区别？

[7] 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？

[8] 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。

[9] 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。

[10] 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)

[11] 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。

[12] 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

[13] 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?

[14] 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。

[15] 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

[16] 你的数据是什么样子的

[17] 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