目录

[1、 顶层验证环境 2](#_Toc76743874)

[2、 通信机制 3](#_Toc76743875)

[3、 UVM验证平台搭建 4](#_Toc76743876)

[3.1、Chnl\_pkg的内容： 5](#_Toc76743877)

[3.2、Arb\_pkg的内容 5](#_Toc76743878)

[3.3、Fmt\_pkg的内容 6](#_Toc76743879)

[3.4、Reg\_pkg的内容 6](#_Toc76743880)

[3.5、Mcdf\_pkg的内容(refmodel和scoreboard都在这个里边) 7](#_Toc76743881)

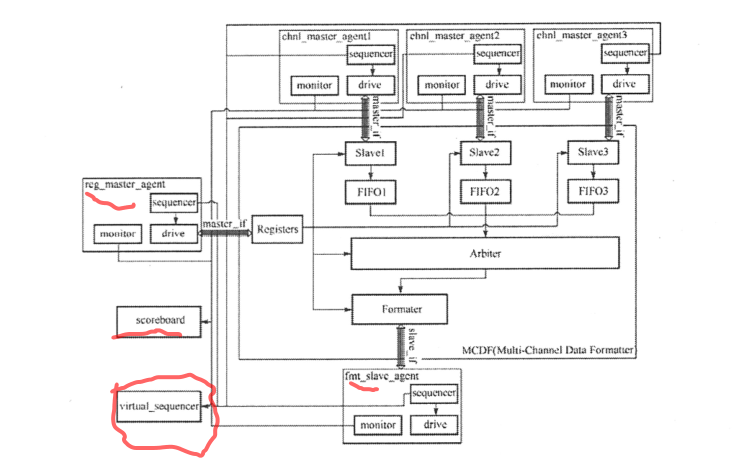
[3.6、mcdf\_rgm\_pkg 8](#_Toc76743882)

[3.7、tb 9](#_Toc76743883)

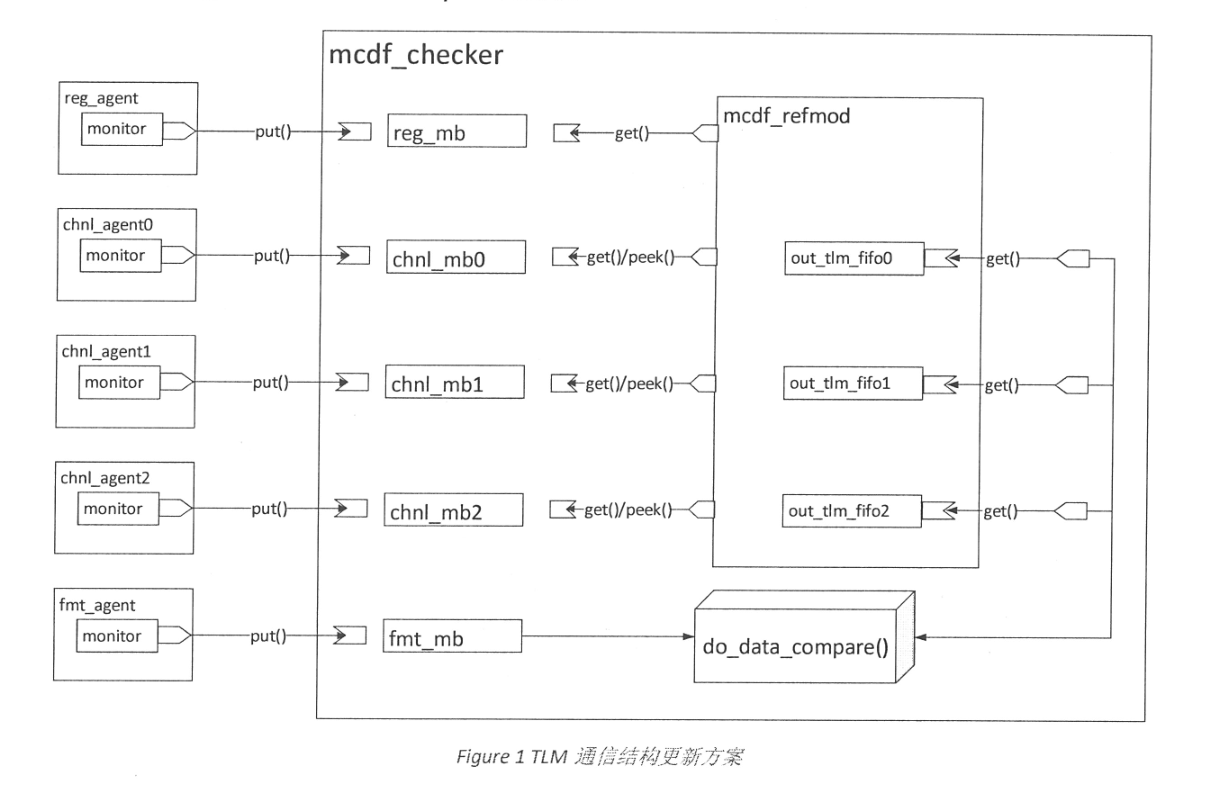
[3.8、参数param\_def.v 9](#_Toc76743884)

UVM的缺点：代码冗余，工作量大，运行速度有损失

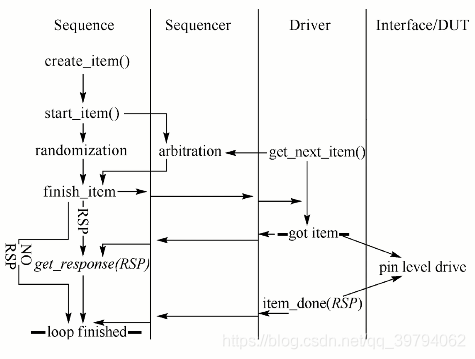
# 顶层验证环境

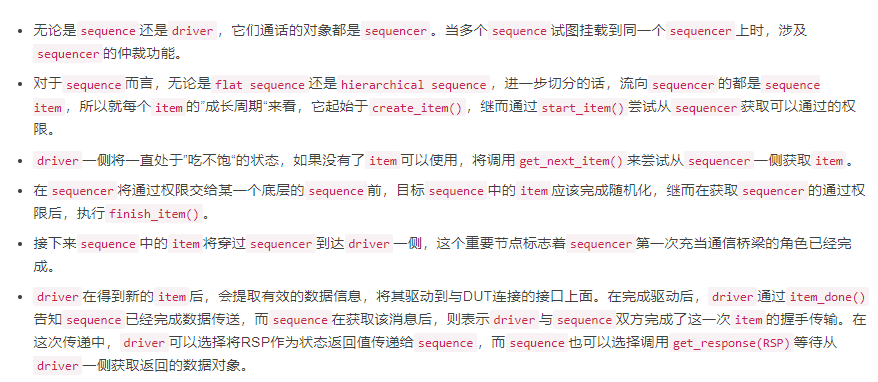


# 通信机制









# UVM验证平台搭建

重点：通过uvm\_config\_db的set和get方法完成各个接口从TB（硬件一侧）到验证环境mcdf\_env一侧的传递，TB：set, mcdf：get 激励是怎么产生的

Reference model是怎么实现的

Scoreboard 是怎么实现的

寄存器模型是怎么实现的

在定义sequence item时，应当伴随域的自动化声明，利用·uvm\_object\_utils\_begin和·uvm\_object\_utils\_end完成

在不同的phase阶段完成组件的例化和连接

MCDF的UVM验证平台主要有Chnl\_pkg/Arb\_pkg/Fmt\_pkg/Reg\_pkg/Mcdf\_pkg

## 3.1、Chnl\_pkg的内容：

包括chnl\_trans-chnl\_driver-chnl\_sequencer-chnl\_sequence-chnl\_monitor-chnl\_agent，还有一个数据打包的结构体，里边是32位的数据和2位的ID 。

1. 定义chnl\_trans类继承于uvm\_sequence\_item类。定义随机的变量，主要有data,ch\_id,pkt\_id,data\_nidles,pkt\_nidles,并对其进行constraint，soft为随机变量指定默认值和分布，然后域的自动化的声明（克隆、打印、比较），构建函数function（new）
2. 定义chnl\_driver类，vif，注册，构建函数，然后set\_interface函数，判断接口是否传递，run\_phase里边两个函数，do\_reset复位（valid和data）和do\_drive （进行sequencer和deriver之间数据的传递）

seq\_item\_port.get\_next\_item(req);

this.chnl\_write(req);

void'($cast(rsp, req.clone()));

rsp.rsp = 1;

rsp.set\_sequence\_id(req.get\_sequence\_id());

seq\_item\_port.item\_done(rsp);

里边有个chnl\_write函数：当时钟到来时，把数据写进去，如果ready

为1，数据已经写进去了，加入data\_nidles，这个数据发送完毕，加入pkt\_nidles

1. 定义chnl\_sequencer继承于uvm\_sequencer，注册，构建函数
2. 定义chnl\_data\_sequence继承于uvm\_sequence,定义数据的值，有10个trans,域的自动化，运用了uvm\_declare\_p\_sequencer，构建函数，用task body()任务来发送数据：采用uvm\_do\_with，之后pkt\_id++,然后get\_response返回rsp,再用断言检查assert(rsp.rsp)，可能随机化之前会调用post\_randomize
3. 定义chnl\_monitor：vif，还有TLM通信端口uvm\_blocking\_put\_port，注册，构建函数（创建对象mon\_bp\_port），set\_interface将接口传递进来，在run\_phase里边将监测数据，当chnl的valid和ready都为1的时候，把chnl的数据传给monitor
4. 定义chnl\_agent，vif，把driver，monitor，sequencer放进来，注册，构建函数，build\_phase里边config\_db::get接口，创建三个对象，在connect\_phase里边连接

driver.seq\_item\_port.connect(sequencer.seq\_item\_export);

## 3.2、Arb\_pkg的内容

包括arb\_trans-arb\_driver-arb\_sequencer-arb\_monitor-arb\_agent

仲裁器的算法实现：就是创建和例化这些类

* 1. arb\_trans：继承于uvm\_sequence\_item，注册，构建函数
  2. arb\_driver：继承于uvm\_driver，注册，构建函数
  3. arb\_sequencer：继承uvm\_sequencer,注册，构建函数
  4. arb\_monitor：继承uvm\_monitor,注册，构建函数
  5. arb\_agent：继承uvm\_agent,注册，构建函数

## 3.3、Fmt\_pkg的内容（重点是driver和monitor）

包括：fmt\_trans-fmt\_driver-fmt\_sequencer-fmt\_config\_sequence-fmt\_monitor-fmt\_agent,定义两个枚举类型，fmt\_fifo\_t和fmt\_bandwidth\_t

1. fmt\_trans:定义各个参数，constraint soft,域的自动化，构建函数
2. fmt\_driver：vif，信箱，构建函数，创建fifo，对象，设置fifo\_bound ，data\_consum\_peroid 的值，build\_phase,中get 接口，在run\_phase中有四个函数， 1、do\_receive：当req为1，时，判断FIFO深度是否够用，然后让grant信号拉高，然后开始传送数据

2、do\_consume：调用fifo.try\_get拿数据，然后重复数据消耗周期

3、do\_config:设置fmt\_trans：req和rsp,首先get\_next\_item（req）,然后设置FIFO的深度和位宽，void将req的类型转换为rsp，令rsp=1，设置sequence\_id，调用item\_done

4、do\_reset：就是复位的时候，让fmt\_grant=0；

1. fmt\_sequencer：注册，构建函数
2. fmt\_config\_sequence：定义FIFO的深度和宽度，约束，域的自动化，构建函数，通过task body发送trans：定义fmt\_trans req, rsp；uvm\_do\_with，get\_response(rsp),然后assert（rsp.rsp）,最后调用post\_randomize 4
3. fmt\_monitor:定义string，vif，uvm\_blocking\_put\_port,注册，构建函数，build\_phase里边：：get接口，run\_phase里边进行数据的检测，mon\_trans函数：定义一个fmt\_trans m 对象，new例化，然后把接口上的数据赋值给m，并通过mon\_bp\_port.put(m),再打印
4. fmt\_agent：声明driver，monitor，sequencer，vif，注册，构建函数，build\_phase里边type\_id：：create对象，config\_db#()::get接口，connect\_phase里边连接

driver.seq\_item\_port.connect(sequencer.seq\_item\_export);

## 3.4、Reg\_pkg的内容

reg\_trans/reg\_driver/reg\_sequencer/reg\_base\_sequence/idle\_reg\_sequence/write\_reg\_sequence/read\_reg\_sequence/reg\_monitor/reg\_agent

1. reg\_trans:addr,cmd,data,约束，域的自动化，构建函数
2. reg\_driver：vif,注册，构建函数，build\_phase，get接口，run\_phase里边有do\_reset和do\_drive（get\_next\_item和item\_done）里边有个reg\_write函数，有write/read/idle把对象的数据写到接口上边
3. reg\_sequencer：继承于uvm\_sequencer，注册，构建函数
4. reg\_base\_sequence：设置参数addr,cmd,data，约束，域的自动化，构建函数，task body（）里边发送数据，有个send trans函数uvm\_do\_with,get\_response,数据赋值，断言检查，最后还有个post\_randomize
5. idle\_reg\_sequence：约束，注册，构建函数
6. write\_reg\_sequence：约束，注册，构建函数
7. read\_reg\_sequence：约束，注册，构建函数
8. reg\_monitor：vif,以及进行TLM通信的uvm\_blocking\_put\_port和analysis\_port的对象，注册，构建函数，build\_phase中::get if，run\_phase中，将接口的数据传到寄存器，地址和指令传过去，如果是写的话，传递写的数据，如果是读的话，就是读数据
9. reg\_agent：声明driver，monitor，sequencer，vif，注册，构建函数，build\_phase里边type\_id：：create对象，config\_db#()::get接口，connect\_phase里边连接

driver.seq\_item\_port.connect(sequencer.seq\_item\_export);

## 3.5、Mcdf\_pkg的内容(refmodel和scoreboard都在这个里边)

首先把chnl\_pkg,reg\_pkg,arb\_pkg,fmt\_pkg,mcdf\_rgm\_pkg几个包import进去，然后定义了packed的结构体，按照len,prio,en,avail等进行，并根据寄存器的设置定义枚举类型

1. Mcdf\_refmod:首先定义vif，声明TLM端口，寄存器和monitor的端口，reg\_bg\_port和in\_bgpk\_ports，还有三个TLM FIFO，注册，构建函数（例化对象）

* Run\_phase里边完成复位，寄存器的更新还有打包，do\_reset,do\_reg\_update,do\_packet(0),do\_packet(1),do\_packet(2)，do\_reg\_update是通过reg\_bg\_port.get端口获得寄存器的值，并且分为读写模式
* Do\_packet，打包，定义两个对象fmt\_trans ot和mon\_data\_t it，通过in\_bgpk\_ports[id].peek(it)端口将it 的数据赋给ot，再通过out\_tlm\_fifos[id].put把ot的数据传出去，其中有个函数 get\_field\_value获得寄存器域的值

1. mcdf\_checker：即是scoreboard,首先声明err\_count，total\_count,chnl\_count[3]等变量，还有chnl\_vif,arb\_vif,mcdf\_vif以及refmod,通过TLM端口将monitor、fmt和reg的数据和checker连接起来，还有信箱chnl\_mbs[3],fmt\_mb,reg\_mb

* 注册，构建函数（例化对象），在build\_phase里边例化信箱还有创建对象，可以通过config\_db进行get接口
* 在connect\_phase里边将这些端口进行连接，主要是将chnl和reg的和mcdf的连接
* 在run\_phase里边检查，do\_channel\_disable\_check，do\_arbiter\_priority\_check和do\_data\_compare，以及refmod.run。
* 其中do\_data\_compare是将fmt\_trans类型的expt和mont进行对比，运用TLM通信获得数据，然后调用compare函数进行比较。
* do\_channel\_disable\_check：当复位信号为1，使能为0时，若valid，ready为1则检查错误
* do\_arbiter\_priority\_check：如果arbiter接收到req信号为1且chnl的某个端口的有较高的优先级，但是不被arbiter所允许，即为错误
* 在report\_phase里边报告结果，打印消息，后边还有很多task，通过TLM通信peek，或get数据

1. mcdf\_coverage：定义chnl/arb/mcdf/reg/fmt的vif，以及delay\_req\_to\_grant变量，注册，定义几个covergroup：cg\_mcdf\_reg\_write\_read，cg\_mcdf\_reg\_illegal\_access，cg\_channel\_disable，cg\_arbiter\_priority，cg\_formatter\_length，cg\_formatter\_grant，定义很多bins,并通过cross定义交叉的部分，binsof指定覆盖点

* 构建函数，例化覆盖组，在run\_phase里边做采样do\_reg\_sample，do\_channel\_sample，do\_arbiter\_sample，do\_formater\_sample
* 在report\_phase里边打印消息
* 在set\_interface里边get接口

1. mcdf\_virtual\_sequencer：声明reg\_sequencer，fmt\_sequencer， chnl\_sequencer，mcdf\_rgm，注册，构建函数，set\_interface
2. mcdf\_env：声明chnl\_agent,reg\_agent,fmt\_agent,mcdf\_checker,mcdf\_coverage, mcdf\_virtual\_sequencer, mcdf\_rgm, reg2mcdf\_adapter, uvm\_reg\_predictor,注册，构建函数，在build\_phase里边创建对象（type\_id::create）,在connect\_phase里边进行TLM端口的连接以及寄存器模型的配置

rgm.map.set\_sequencer(reg\_agt.sequencer, adapter);

reg\_agt.monitor.mon\_ana\_port.connect(predictor.bus\_in);

predictor.map = rgm.map;

predictor.adapter = adapter;

1. mcdf\_base\_virtual\_sequence，声明几个sequence：idle\_reg\_sequence，write\_reg\_sequence，read\_reg\_sequence，chnl\_data\_sequence，fmt\_config\_sequence，注册，构建函数，在task body()里边p\_sequencer连接寄存器句柄,并进行do\_reg,do\_formatter,do\_data操作，并通过diff\_value进行数据的比较
2. mcdf\_base\_test：声明env，各个vif，注册，构建函数，在build\_phase里边创建env对象，并把各个vif通过config\_db#()::get进来,在connect\_phase里边进行vif接口的连接，end\_of\_elaboration\_phase进行测试环境的微调，在run\_phase里边运行sequence
3. mcdf\_data\_consistence\_basic\_virtual\_sequence：继承于mcdf\_base\_virtual\_sequence，注册，构建函数，然后写do\_reg,do\_formatter,do\_data三个任务，任务里边使用uvm\_do\_on或者是uvm\_do\_on\_with
4. mcdf\_data\_consistence\_basic\_test：继承于mcdf\_base\_test，注册，构建函数，还有这个任务run\_top\_virtual\_sequence

## 3.6、mcdf\_rgm\_pkg

首先把import reg\_pkg，然后包括ctrl\_reg/start\_reg/mcdf\_rgm/reg2mcdf\_adapter等几个类

1. ctrl\_reg：(读写寄存器)，注册，声明各个寄存器域，设置覆盖组，构建函数（里边设置覆盖率），在build函数里边创建对象，并设置各个域的参数，在sample函数里边进行采样
2. start\_reg:（只读寄存器），注册，声明保留域，定义覆盖组，在build函数里边例化对象，uvm\_reg.configure设置各个域的属性，在sample函数里边做采样
3. mcdf\_rgm:继承于uvm\_reg\_block，声明6个chanl的ctrl\_reg和start\_reg以及uvm\_reg\_map,在build函数里边例化对象，并设置各个寄存器的参数，通过add\_hdl\_path将寄存器模型关联到DUT一端，add\_hdl\_path\_slice完成寄存器模型各个寄存器成员与HDL一侧的地址映射，最后通过lock\_model结束地址映射关系，并保证模型不会被其它用户所修改
4. reg2mcdf\_adapter：继承于uvm\_reg\_adapter，注册，构建函数，实现reg2bus和bus2reg两个函数

## 3.7、tb

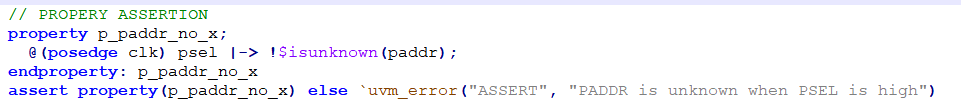
1. 定义几个接口chnl\_intf/reg\_intf/arb\_intf/fmt\_intf：定义各个接口的信号，以及drv\_ck和mon\_ck时钟块到来时的输入和输出
2. mcdf\_intf：定义chnl\_en信号，以及定义时钟和复位
3. tb里边对dut的例化， import pkg,mcdf的接口检测其端口还有信号，通过uvm\_config\_db#()::set interface ,这个必须在run\_test前做

## 3.8、参数param\_def.v

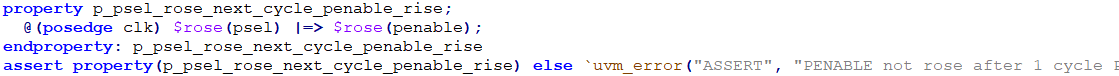
定义参数，断言写在interface或module里边，但是MCDF写在apb\_if

断言检查，主要是检查APB协议：

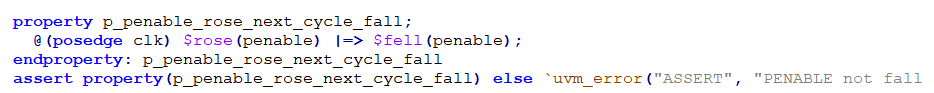
1. 在PSEL为高时，PADDR总线不可以为X值



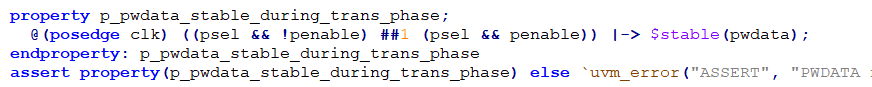
1. 在PSEL为高的下一个周期，PENABLE也应该拉高



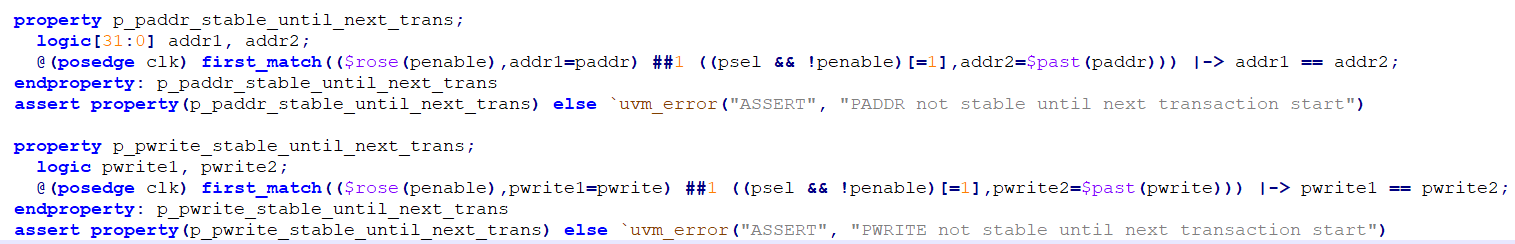
1. 在PENABLE拉高的下一个周期，PENABLE应该拉低



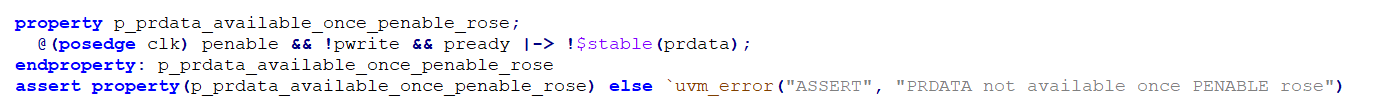
1. 在PSEL和PWRITE同时保持为高的阶段，PWDATA需要保持



1. 在下一次传输开始前，上一次的PADDR和WRITE信号应该保持不变

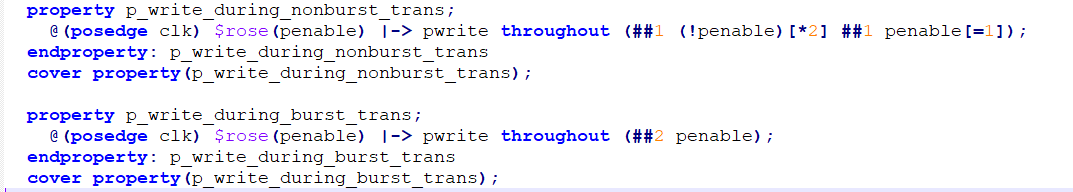


1. 在PENABLE拉高的同一个周期，PRDATA也应该发生变化

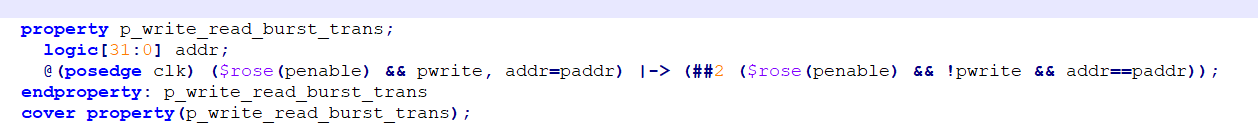


APB总线协议的断言覆盖率

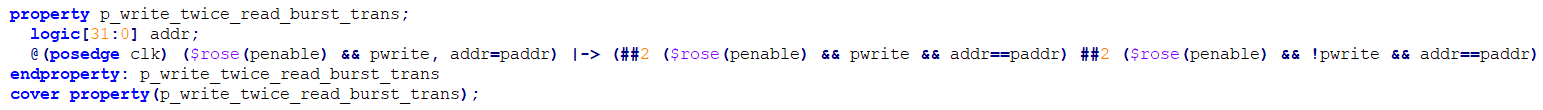
1. 在写操作，分别发生连续写和非连续写。



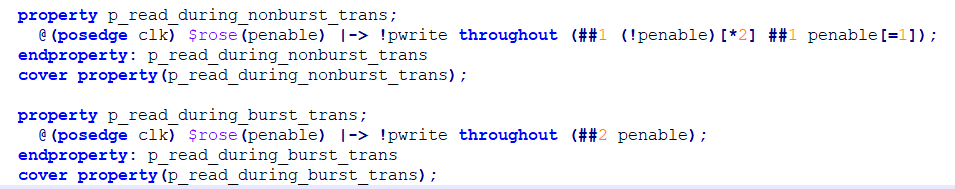
1. 对同一个地址先做写操作再不间隔做读操作。



1. 对同一个地址做连续两次写操作再从中读数据。



1. 在读操作时，分别发生连续读和非连续椟。



1. 发生对同一个地址的读操作再不间隔做写操作，再不间隔做读操作。

