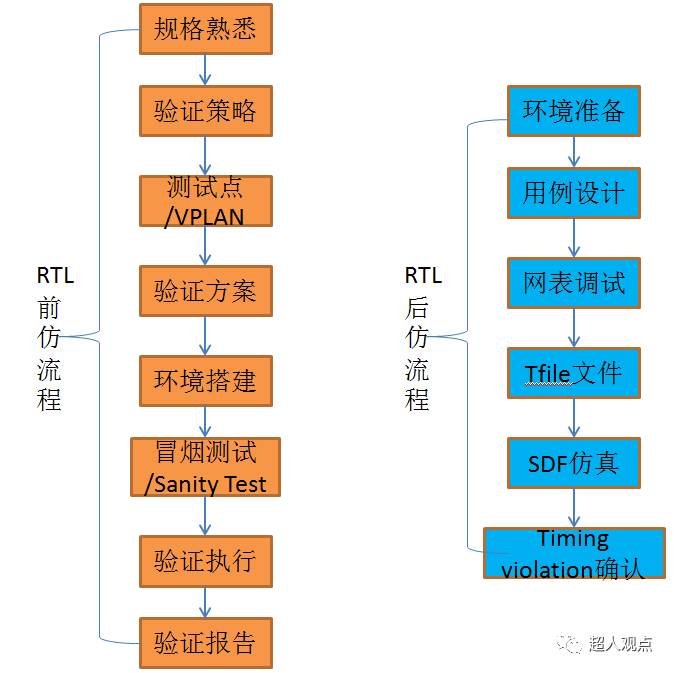
**IC验证流程及质量管理**

下图既是一副比较完整的芯片端到端验证流程图，大多数公司的验证流程是大同小异的，差异点在于各个流程中具体做的事情。



**规格熟悉**

这是属于项目刚立项后验证人员要做的事情，主要是学习各种相关的文档材料，包括但不限于协议、需求、规格、功能说明、历史芯片文档、重用环境评估与恢复。

我有一个习惯，就是会将学习内容整理到一个PPT中，主要是为了提炼、归纳、整理下所学知识，弄的简洁有条理一些，方便自己理解、记忆和日后查阅。

大多数公司都会有这一步，无论多忙。个别公司刚进项目就让调试测试用例，或许是因为模块重用度较高，风险不大，所以才这么做的。但我个人始终认为这是有风险的：我认为规格熟悉这一步是无论如何不能省略的，对所验证对象都不熟悉何以采用切实有效的方法进行验证？即使验证工作做完又何以保证验证完备没有bug遗漏？另外重用代码也不一定是没有问题的，哥不止一次在成熟重用代码中发现bug。

**验证策略**

这项工作主要由验证SE或验证经理完成，主要是进行BT(Block Test)、IT(Integrated Test)、ST(System Test)的划分、集成策略（外企一般叫Module Test和Chip Test），专项验证规划，FPGA测试策略，重用策略，质量计划等工作。

无论设计还是验证，模块划分都应该遵从“**高内聚、低耦合**”的原则，即尽量将相关功能放在一个模块或子系统实现，尽量减少模块间功能耦合、交互以及减少信号连线。过多的信号连线意味着更多的工作量，更大的潜在风险，更复杂的验证环境实现。

是否进行模块验证的基本原则是：除非该模块高度重用，则不做模块验证，仅在IT验证阶段对模块修改点及其影响点做重点测试；否则所有模块都必须做模块验证。

      关于模块验证范围的通用原则是：该模块所有功能必须在模块验证阶段完成，不允许将BT/IT混为一谈，一般不允许将BT功能放到IT/ST测试。IT/ST验证只关注应用场景、模块子系统配合、功能耦合、信号连接、子系统性能等。

专项验证主要在ST层次完成，包括但不限于寄存器专项、中断专项、时钟复位专项、DFX专项、RAM专项（分散在各BT完成）、Function  Mbist专项（网表、gatesim阶段完成）等等。

**测试点/VPLAN**

在外企这个验证活动叫VPLAN/VO，中企叫测试点（Test Point）分解或验证计划，我觉得叫测试点比较贴切一些。测试点一般分解过程如下，首先根据各种文档梳理出验证特性，然后根据验证特性细化出测试点。



我印象中比较好的测试点分为**场景类、功能类、性能类、白盒测试点(设计人员提供)、接口类、异常类**6个维度，全面、明确、细致，无歧义的将所有验证特性细化为一个个不可分割的小点，每个点明确采用directtest还是coverage覆盖还是assertion覆盖。功能覆盖需要细化覆盖范围，比如典型值、边界值、异常值、cross。当然测试点也不是一次分解完成，在整个验证过程中会进行多次分解和review，直至完善。

在这里我需要强调的一点是场景类、性能类、异常类测试点一定要尽早思考和分解，SE也要多参与进来提供支持，比如各种随机配置、异常配置、动态配置、power反复开关，开关过程中各种操作等等，不要总是关注于常用的功能，不能总停留在舒适区，只有经得起反复考验的代码才是好代码，受得起各种蹂躏的芯片才是好芯片。

如果可能，测试点分解完后建议做一下测试点反标，即需要将测试点映射到所有相关文档，主要目的在于保证测试点分解完备、无遗漏。

经过这么多年的验证工作以及经历过几家大公司的洗礼，我深刻地认识到：测试点分解绝对算是芯片验证工作中的最重要的一环，是充分体现验证人员经验、能力、智慧、价值的一项工作。因为只要你想到的点，你肯定会有办法把它验证到，无论是采用什么验证语言或是验证方法。芯片中的bug往往都是没有想到的点或者没有覆盖到的点。所以测试点分解一定要追求完备、细致、无歧义，要做到测试点分解完成后，无论哪个验证人员测试，验证质量都是有保证的。过于粗糙的测试点会导致不同的验证人员在测试用例设计时有不同的理解和实现，或许就会遗漏掉一些corner点。另外粗糙的测试点也会造成工作量评估不准确，导致后期突发任务增多，造成项目延期。

该省的时间可以省，但测试点分解的时间我认为绝对不能省太多，尤其是首次测试点分解。所谓磨刀不误砍柴工，只有你非常清楚地知道你要测试什么怎么测，后面的验证活动都好说。怕就怕还没把模块搞太清楚就囫囵吞枣开始搭环境、写case，我认为这不是明智之举。

**验证的最终目的是为了保证芯片成功无bug，不是为了匆匆把事情做完**。

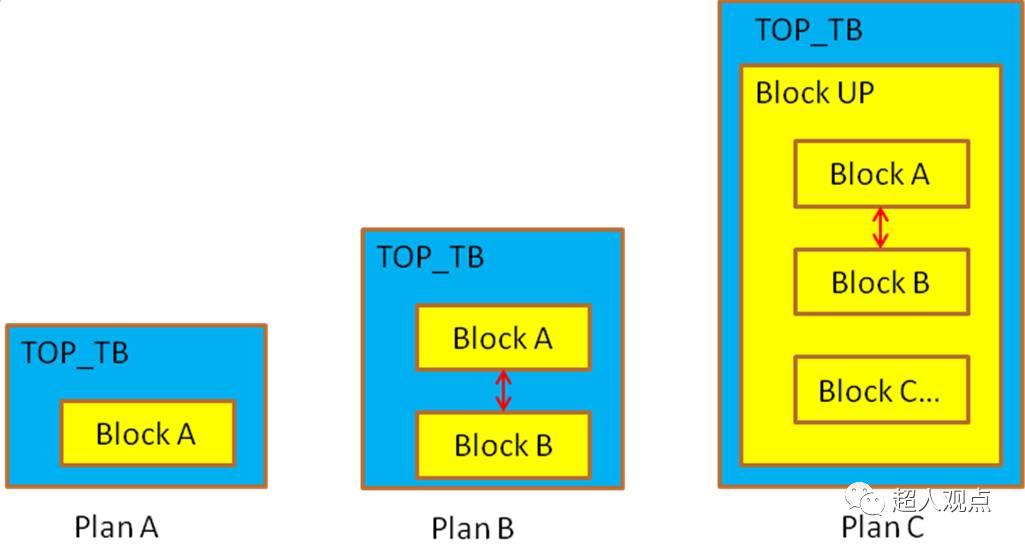
芯片研发有一个10倍代价说法：前一个流程遗留下来的问题在后一个流程去发现解决会付出10倍的代价。我深表赞同。

**验证方案**

验证方案没有什么特别的，所有公司都一样，就是设计验证架构来保证所有测试点都能在该验证环境覆盖到。当然标准化、参数化的方案设计是必须的，要考虑到该模块的后期集成和重用，环境比较大时还要考虑到验证环境的运行效率问题。

注意设计参考模型或checker的时候一定不要过多参考设计方案或代码，理论上设计和验证从spec开始就是分开的，是平行独立的两条线，过多的参考设计是非常不可取的，造成的后果就是RM/checker和设计代码实现一样，case总是一跑就过完全测试不出设计bug，这样的失败案例比比皆是。

另外DUT顶层TOP\_TB包封的时候经常会遇到一个问题：就是被测试模块如果跟其它某个或某些模块交互较多，这时候应该怎么办？



如上图所示：假如Block A就是我们的测试对象，跟Block B模块交互比较多，顶层TOP\_TB包封就会出现上面3种方案。

Plan A：仅将Block A包封在TOP\_TB。这样做的优点是TOP\_TB包封简单，case跑的快点。缺点也是显而易见的，就是输入输出信号太多，环境中interface、driver、monitor、RM设计会变得复杂，前期编码和调试工作量及难度会增加，后期环境稳定性较差，维护工作量也会增加。

Plan B：将Block A和Block B一起包封在TOP\_TB，A B之间的信号参考RTL连接。这样做的优点当然是解决了DUT A B模块间信号交互问题，跟验证环境交互的信号减少，环境实现也简单多了。但是不可忽略的一点就是BlockA B之间连线是验证人员连接的，不是真正的DUT，可能会遗漏某些问题，不是特别放心，这种等效做法还是曾经出现过问题的；另外一点就是如果后期DUT不稳定经常变化，那么A B之间的连线也要经常更新，无形中又增加了工作量和出错概率。

Plan C：选用Block A和Block B共同的上层模块（例如Block UP）做DUT，将该Block UP包封在TOP\_TB，Block UP模块接口上其它无关信号在TOP\_TB接合理值，环境输入激励接Block UP输入，输出激励从Block A或Block UP获取。这种做法带来的好处是显而易见的：除了兼容了Plan B的优点外，就是DUT包封始终都是完整的RTL代码，界限比较清楚，debug也方便。当然代价就是DUT包封的有些大仿真速度可能慢点，但BT测试5分钟和10分钟的仿真时间对工作效率真的没有多大影响，时间主要是花费在测试用例设计和debug上面；另外一点就是前期TOP\_TB信号连接会复杂点，我认为这都是可以接受的。

有人会问那Plan C不就相当于一个小的IT了吗？我想说的是：IT/BT的划分不在于你的验证环境包封了哪个DUT层次，而在于你的测试对象。无论你采用哪种方式，只要你能够完整、准确、方便、高效地将被测模块的所有测试点测试完成，那就是一个好的BT验证环境。

当然这里也能体现出前面10倍代价说法的威力：假如前面测试点分解的时候比较粗糙遗漏掉一些功能点，然后验证方案设计和环境搭建时就不会考虑这些，等后面开展质量活动的时候发现有些功能点没覆盖，这个时候再反过来修改环境可能会非常麻烦，不仅会影响自己的计划执行还可能导致整个顶层环境集成都得动一遍，某些已有case回归也可能受影响又要修改，得不偿失。因此整个验证流程是一个环环相扣不断迭代的过程，每个环节都不能马虎。

有些公司会要求验证人员输出非常详细的验证方案文档，要求细化到每个transaction、driver、monitor、RM、scoreboard、coverage、interface、agent、ENV集成、DUT包封等。个人认为这个文档可以适当省略或不写，但前提是验证人员自己必须要首先想清楚整个环境架构和每个组件实现细节。新员工输出该文档还是有些必要性，对个人成长比较有帮助。

**环境搭建**

这是验证人员的基本功，一个足够完备灵活自动化的验证环境能节省后面测试用例实现的很多工作量，测试用例会变得很简单，不同的测试用例只需要开关某些配置和修改一些约束；反之一个糟糕的验证环境会让测试用例变得冗长、复杂、低效。

**冒烟测试/sanity test**

顾名思义，就是一跑就冒烟挂了。这是设计代码和验证环境都刚刚完成后的测试，目的就是确保寄存器读写OK和打通基本数据流。这个过程设计人员和验证人员高度配合，发现bug会立即修改。这个过程会发现很多代码问题，这个阶段发现的bug一般不提问题单。

**验证执行**

冒烟测试完成后，DUT已经基本可以正常工作，这时候就正式开始进入验证执行阶段，按照测试点一个一个进行覆盖，写测试用例、debug，后期进行代码覆盖率和功能覆盖率的分析、用例增加以及最后用例检视。这个阶段发现的所有RTL问题都必须提问题单。

有些新员工对测试点和测试用例的覆盖关系没有概念，这里讲一下业界通用做法以及我自己的一些经验：

1）  
测试用例是用来覆盖测试点的，一个测试用例可以覆盖多个测试点，在考虑用例复杂度和仿真时间的前提下单个测试用例应覆盖尽量多的测试点；测试用例可以适当复杂一些，就如我前面所说，经得起反复考验的代码才是好代码。我发现的那些corner bug大多都是复杂用例发现的。

2）  
单个测试点在一个用例中必须被测试覆盖，不可出现多个测试用例测试通过后，某个测试点才覆盖的情况。如果有，必定是测试点分解太粗，需要重新细化该测试点；

3）  
直接用例和随机用例有机结合，在保证典型应用场景和典型配置用直接用例覆盖或遍历前提下，多用随机用例来覆盖更多验证空间，另外随机用例一定要配合功能覆盖率才能保证覆盖效果。

注意上面所说的第3点，有时不要过度迷信于随机用例而忽视直接用例的价值，曾经出现过BT测试从头到尾使用大量随机用例，代码覆盖率和功能覆盖率都很高，但是在IT阶段刚一测试就出现问题，最后发现是DUTbug。原来BT测试时最典型的链路配置都没测试，而是采用大量随机用例，而bug就恰好出现在最典型配置上。

另外上面的第2点，也是一些公司我认为做的不妥的一点。有些公司经常以测试用例执行情况作为工作进度评估标准。要知道测试用例的难易程度差别很大，并不能真实反映测试工作量，比如一个时钟用例仅检测时钟频率，很快就做完；而一个所谓all\_ip\_accessbility的case，实际上需要至少半个月的时间。所以我们关注的应该是测试点而不是测试用例，测试点体现的才是工作量才是验证对象，当然前提是测试点要分解的足够细致。

另外就是一些通用的操作建议尽量包封成函数，一个函数尽量在一页写完。这样不但测试用例会简洁清楚，更重要的是减少重复劳动。

**验证报告**

验证执行做完后，一些公司会要求每个验证人员输出验证报告，我见过做的比较全面的验证报告包括以下几个维度：

1）  
应用场景分析；

2）  
专项分析：

* 寄存器、中断；
* 随机性分析：接口输入随机、配置随机；
* 异常分析：接口输入异常、激励异常、配置异常、处理异常；
* 低功耗验证分析；
* 接口配合分析；
* 反压分析；
* 性能分析；
* 计数器；
* 告警；
* 时钟复位；
* 异步；
* RAM；
* 验证薄弱点；
* 白盒测试点；
* Bug list；
* IP/BB分析；
* FPGA和EDA差异分析；

3）  
重用分析：

* 修改点（规格变化、接口、配合、RAM替换、MPI时序、器件替换）；
* 影响点。

4）  
系统配合分析：

* 信号传递配合；
* 系统耦合点分析；

5）  
覆盖率分析：

* 代码覆盖率（行、条件、FSM、toggle）；
* 功能覆盖率；
* 断言覆盖率；

6）  
风险评估；

7）  
验证结论。

**后仿真/Gatesim**

后仿真（gatesim）流程我见过的基本都一样，关注重点不是功能，而是检查在特定工艺下电路物理实现的setup/hold时间是否满足工艺需求。

后仿真测试用例的设计主要覆盖异步路径、外部接口、时钟复位、主要工作模式、数据流、power开关、频点、Mbist等等。测试用例在后仿应该尽量精简，宜少不宜多，要充分考虑工作站资源和仿真时间问题，最好控制在一天内跑完。测试用例pass不是后仿真关注重点，Timingviolation确认才是后仿的工作重点。

网表调试主要是为了调试后仿环境、用例，消除不定态。后仿出现的不定态一般有以下几种类型：

* 输入不定态，输入悬空；
* 时钟、复位不合理；
* RAM没做初始化，不定态传递；
* 个别寄存器没做复位；
* DFT信号悬空；
* 管脚没有power on/power down;
* 接口多驱动。

Tfile文件一般要求设计人员提前准备好加载进环境中，因为只有设计人员最清楚哪些电路是异步路径。

Timing violation出现的原因主要有以下一些：

* 异步路径；
* 上电复位拉起（复位撤离）之前，不定态传播导致上报timing violation;
* DFT引入的timing violation，与功能无关，需与后端和设计人员确认；
* 工作模式、时钟切换、power开关过程中出现的timing violation；
* 异步复位同步撤离，一般会在开始上报一两次timing violation;
* 打两拍做异步处理的第一级reg;
* Latch的EN端和CLK端，在开始时可能上报timing violation.

我见过芯片质量比较好点的公司后仿用例一般较少，后仿release版本也较少，只要前面网表调试阶段消除不定态后，后面的SDF网表基本都能跑通，后仿极少发现过RTL的问题。

**验证质量保证**

**质量活动**

一些公司在做完验证执行之后RTL前仿工作就结束了，试问这时验证人员会有多少信心说我验证的模块没问题了？恐怕没多少人敢拍胸脯吧。

有人说我代码覆盖率和功能覆盖了都100%了，还没验完吗？

代码覆盖率100%并不能保证所有的功能都覆盖到了，电路在乱序状态下的覆盖率也会被记录下来，而且即使覆盖到的代码，也并不一定在环境中就做了check。代码覆盖率的最大价值在我看来就在于它能够指出我们没覆盖到的那些点，这些点是肯定没测试到的或测试不到的（比如冗余代码），然后我们再补充测试将那些没测试到的点覆盖，将确定测试不到的点过滤掉。

功能覆盖率的价值主要在于确保我们随机用例的效果，100%仅表明我们确定的那些功能点都覆盖到了，不确定的呢？而且有些功能点也是不适合用功能覆盖率来保证的。

其实这时候只是最基本的验证工作做完了，接下来要做的事情才是全方位保证验证质量和增强验证信心的手段。

不同的公司会在验证执行中后期根据自身业务特点开展丰富多彩的验证质量活动，包括但不限于以下所列：

1）  
专项验证（寄存器、中断、时钟复位、RAM、DFX、Function Mbist等等）；

2）  
Bug review, 错误案例分析；

3）  
波形确认；

4）  
验证发散，比如随机约束放开；

5）  
不常见的组合分析；

6）  
修改点、影响点、耦合点分析；

7）  
代码、文档一致性检查；

8）  
上下游配合检视；

9）  
FPGA测试检视；

10）  
验证代码检视（约束、配置、接口、coverage、断言、force、TC等）；

11）  
Corner点确认；

12）  
End\_of\_check（case执行完毕，检查所有中断和状态寄存器符合预期）；

13）  
初始化流程检查；

14）  
异步、最小时钟、最大复位；

15）  
  
遍历测试；

16）  
环回测试；

17）  
低功耗验证分析；

18）  
文档、寄存器反标；

19）  
粘连检视（时钟、复位、数据、告警、中断、配置等）；

20）  
等效、等价类；

21）  
设计、验证差异性检视（杜绝设计验证代码实现一致）；

22）  
头脑风暴。

认真做完上面的这么多质量活动之后，才能基本证明验证做完备了，才能差不多有信心说我的验证没问题了。

事实也证明了这点，以我个人为例，我工作过的其中一家公司质量活动开展较多，芯片一版成功率很高，我多年来负责的模块/子系统在100%网表后都是0 EC。

**标准化、规范化**

能力成熟度比较高的公司整个流程每个阶段均有规范的输入输出条件和明确的做事标准，绝不会出现没有规格就开始设计、验证编码，一定要走完每个流程输出各种交付件并评审通过后才能开始编码工作。

每个公司都会有详细的设计、验证编码规范，比如不同层次空几格，各种信号简写，注释的添加规范，各种参数前缀后缀，一行中不能出现太多条件组合，等等。这些规范不仅统一了格式，让代码整洁干净出错概率降低，而且减少了沟通成本，后面其他人参考和review也都会方便很多。

问题单/defect也应该提前制定合理明确的格式规范和提交原则。我认为一个比较好的问题单至少应该包含以下几项：问题模块层次、简介、详述、设计验证版本、用例名称、seed等信息。另外我认为验证执行阶段开始到芯片投片发现的bug都应该及时提单，这没有什么好商量的。

版本发布也要规范，一般要求跑完sanity test后没有问题再发布，否则匆匆发布后万一用例回归大量出错，不仅数据不好看，而且会浪费更多的debug时间。大的公司一般会有专人进行版本发布和维护，绝不会出现将修改代码增加进已发布版本的事情。

**完备性思维**

我认为验证活动本质上是一道证明题：就是运用各种方法手段，最终证明验证目标测试完备无bug。这才应该是芯片验证人员的价值和追求。

验证需要将完备性思维贯穿始终，争取一次把事情做对。测试点需要确认和证明分解完备；验证环境需要能覆盖所有测试点；验证执行阶段要保证所有测试点测试完备；完备性分析阶段要保证所有质量活动有效完全地执行。

**验证回归**

有些公司会要求由验证执行开始到芯片投片前每个验证人员都必须进行daily regression（后期会降低到每周2、3次）。回归结果验证经理会每日定时自动统计。失败用例需要及时回复原因并改正。

我认为这个做法是很有道理和成效的，我们做了那么多随机用例，就是为了寻找已知和未知的验证空间是否还存在bug，如果只进行有限的回归，那随机用例的价值就大打折扣。

另外有公司芯片投片标准之一是：验证质量活动完成后连续两周回归都没有再新发现任何RTL问题才允许投片。

**验证的定位**

我始终认为验证工作是一项科学严谨重要的研发活动，验证人员也是核心研发人员，绝不是设计人员的附属品，更不是产线操作工。

验证工作是芯片前端开发的最后一道保障，是保证芯片一版成功的重要基石，小的芯片或许重点依靠设计人员能够一版成功，大的SOC成功必然是设计验证紧密配合共同努力的结果。

那些把验证人员当designer附属品或产线操作工的公司，得到的也基本会是附属品或操作工的芯片质量；反之，把验证工作看做跟设计工作同等重要地位的公司，芯片质量往往会好很多。

**写在最后**

酝酿两月，历时两天，终于写完了。仅为一家之言，肯定有很多谬论和不足，敬请斧正。

不知不觉写了这么多字，废话多的毛病以后真得改改了

今天正好是感恩节，在这里首先感谢能坚持看到这里的读者朋友，你们都是最帅和最美的；其次感谢我曾经和现在工作的公司，让我学到了很多知识和技能，认识了很多朋友；最后感谢曾经和现在与我一起并肩战斗的同事，你们都是最棒的！