验证通识3

1. 计划的概述：

验证计划：不断更新

计划步骤：

* 创建验证计划
* 选择验证方法
* 人力资源调配
* 构建验证平台和环境组件
* 开发测试用例

概述：收集的材料：

* 结构**功能**描述 Hardware design
* 设计的各种操作**使用**模式 Function design
* 在正常输入和错误输入情形下设计的**行为** action
* 设计的**接口**
* 在**边界**情况下的设计行为 corner case
* 设计在**实际**使用中的场景描述

计划概述**优点**：

* 使设计和验证人员对功能描述文档的**理解和翻译保持一致**
* 将自然语言描述的功能通过**可测试的语言**描述（提取测试功能点）
* 更**合理评估**出工作量，人力安排和进度节点
* 为验证人员提供清晰的**验证目标**、**任务**和**进度安排**
* 为功能文档**提供反馈**，修改文档中不明确、有歧义的描述

影响计划的因素：

* 会有**不同人员**更新验证计划：系统、设计、验证、软件人员给出意见共同参与
* 需要更新上百上千的测试用例，与计划中的**待测功能映射**
* 考虑选择不同的**验证方法**。不同的设计选择不同的验证方法
* **更新计划**
* **多组协调**

相较于设计计划，验证计划更需要动态的调整

1. 计划的内容

技术部分&项目部分

**技术**：验证功能点、验证层次、测试用例、验证方法、覆盖率

**项目**：工具、人力安排、进度安排、风险评估

验证**功能**点：

**基本**功能：clock、power、reset、寄存器访问和基本特性，在模块级完成验证

**互动**功能：一些需要同其他模块互动的特性，需要在更高层次的子系统级或者芯片级完成验证

**次要**功能：通常会在项目后期完成验证，如：性能验证、效能验证。即时没有通过验证要求，也没有致命影响。

验证**层次**：

了解功能点是否可以在**较低层次**完成验证

从**验证效率和激励自由度**来看，尽量在较低层次验证更多的功能点

在较高的层次，如芯片级，侧重于**系统集成测试**

验证的**方法**：

采用何种，动态仿真、形式验证or硬件加速

采取何种透明度，黑盒、白盒or灰盒

采用**定向**测试or**随机**约束激励

验证**用例**：

**更随机**的方法，尽可能**遍历**各种状态空间

**适中的随机约束**，倾向于更**贴近实际场景**的随机激励

采用**定向**测试，一些**边界情况**可以更有效地完善覆盖率

覆盖率的**要求**：

覆盖率是衡量**激励**生成种类和**功能点**验证的**量化**指标

覆盖率分为：**代码**覆盖率、**功能**覆盖率、**断言**覆盖率

除了**合法**激励之外，也需要考虑给出一些错误的激励，来测试设计的**稳定性**和纠错能力

**工具**的选择：

仿真工具:Questa, VCS, Veridi

形式验证工具:

验证IP

断言IP

调试器

硬件加速器

HVL, high-level verification language : System C

人力安排：

确定方法后，考虑投入的人力

不同的验证方法，除了考虑个人的实际经验外，也需要考虑他们是否熟悉该模块。

在一个完整项目周期，让**固定的人员跟踪同一个设计模块**，从搭建环境到模块级，子系统级，芯片系统级，人员成长更快，项目风险低

进度安排：

大致的时间表：工作量=人力X时间

考虑如何恰当的安排人力，做好动态人力分配，实现高效的资源配置

在计划中细分量化：设计的交付时间，验证的验收时间，不同模块的集成时间等拆分

风险评估：

芯片结构不稳定：结构变化

工具不稳定：替换工具版本，影响验证性能和环境，仿真编译和运行出现问题

人力的不稳定：模块验证人员临时替换活跃，一人任多职（解决：验证文档）

模块交付时间的不稳定因素：明确交付时间。

1. 计划的实现

概述：

细致的计划包括**项目动向**、**更新内容**和**工程进度**，清晰的计划保证**时间和人力的平衡**。

验证计划，需要时常**保持更新**，给出**合理的安排**，过程蕴含**从计划到实践再到反馈**，最后到**修改计划**周期。

如何制定验证计划：

邀请相关人员参加会议：设计人员，验证人员，硅后测试人员，软件开发人员，系统人员，验证经理（项目经理）



开会讨论：验证计划的**内容组成** & 需要确定的验证**功能点**

验证计划的模板（组织结构）：

* 设计功能简要**描述**
* 硬件实现**框图**
* 待验证的**功能点**
* 验证**环境**搭建
* 测试**用例**构成
* 编译**脚本**和**回归**测试
* **覆盖率**分析

有功能点之后如何**确定测试场景**

针对功能点，给出特定的测试场景。

在构建测试用例前，心中已经**模拟出测试序列**，**参与进来的模块**，以及**如何配置寄存器**、**等待某些状态信号完成下一步功能设置**，直到最后**完成整个功能测试**；

如果涉及**电源开关**，需要考虑在PA（power aware）场景中完成测试；

如果与**性能**相关，需要考虑如何发送大**规模的数据量实现压力数据传输**；

针对不同的功能点，我们需要考虑选择**合适的验证层次**，以及**对应的验证方法**。

**创建验证环境**：

* 针对设计模块的**接口信号**需要实现对应的激励发生组件，通过控制**协调不同的激励组件**来构建场景；
* 尽量复用验证资源，节省构建时间；
* 收集数据和对比结果；
* 监事信号组件将分析整理好的数据发送给检查组件，由检查组件进行数据比较，给出比较信息和报告，最终判定测试是否成功；

1. 计划的进程评估

从各项参数综合评估验证的完备性：

* 回归测试通过率（regression pass rate）
* 代码覆盖率（code coverage）
* 断言覆盖率（assertion coverage）
* 功能覆盖率（function coverage）
* 缺陷曲线（bug curve）

回归测试通过率：

**回归测试表**将测试设计所有功能点的用例合并为一个测试集

* 回归测试表**主要功能**：用来在设计经过缺陷修复或者性能提高后测试原有的所有功能点，确保设计仍然可以正常工作。
* 这种往复的测试方式不仅在于**确保新的设计变化不会影响之前的功能**，也可以用来**避免修改后的设计对于别的模块造成的功能失效**。
* 设计的维护不仅在于按照设计需求提供新的功能，也需要保证新功能不会影响原有的功能。

回归测试表中的测试用例需要确保可以**重现**激励场景。

对于定向测试方法容易实现，对于随机约束测试而言，需要显示每次测试中使用到的随机种子（random seed），通过特定的种子，**跟踪调试**失败的用例

在某一个层次的回归测试通过后，向上迁移到新的验证层次

或在设计需求发生变化时，重新从模块级开始递交测试表

不同层次的回归测试表，每个测试用例的仿真时间消耗也不一样。

**不同层次、不同设计规模、不同测试场景复杂度**，都会影响测试用例的仿真时间。

对验证平台的优化和运算资源会有更高的要求。只有更快的往复递交和得出结果，才能更快的得知新的设计变动是否可靠。

**代码覆盖率**：