基于UVM的UART验证环境开发

作者姓名_____夏晓芸_____

学校导师姓名、职称___吴振宇 副教授____

企业导师姓名、职称___朱思良 高工_____

申请学位类别_____工程硕士_____

学校代码 10701 学 号 1311122937
分 类 号 TN4 密 级 公开

西安电子科技大学

硕士学位论文

基于 UVM 的 UART 验证环境开发

作者姓名：夏晓芸

领　　域：软件工程

学位类别：工程硕士

学校导师姓名、职称：吴镇宇 副教授

企业导师姓名、职称：朱思良 高工

学　　院：微电子学院

提交日期：2015 年 12 月

UART ENVIRONMENT RESEARCH AND DEVELOPMENT BASED ON UVM ARCHITECTURE

A thesis submitted to

XIDIAN UNIVERSITY

in partial fulfillment of the requirements

for the degree of Master

in Software Engineering

By

Xia xiaoyun

Supervisor: Wu zhenyu Associate Professor Zhu siliang

Senior Engineer

December 2015

西安电子科技大学
学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同事对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文若有不实之处，本人承担一切法律责任。

本人签名：_____　　日　　期：_____

西安电子科技大学
关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权属于西安电子科技大学。学校有权保留送交论文的复印件，允许查阅、借阅论文；学校可以公布论文的全部或部分内容，允许采用影印、缩印或其它复制手段保存论文。同时本人保证，结合学位论文研究成果完成的论文、发明专利等成果，署名单位为西安电子科技大学。

保密的学位论文在___年解密后适用本授权书。

本人签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘要

随着集成电路已经进入后摩尔时代，电路的复杂性日益提高，因此验证工作的难度也越来越高，随着验证要求的提高，在实际生产中，传统的直接验证已经越来越难以满足设计的需求。为了使产品更加具有竞争力，在激烈的市场竞争中获取有利地位，在确保功能正确的前提下，提高代码复用率，缩短验证周期成了各个公司缩短产品上市周期的重要方法。为此，就需要采用更有效的验证方法，使得验证工作更加有效。

在市场的驱动下，验证工作日益成熟，渐渐独成一体。基于 System Verilog 的 UVM 验证具有很高的抽象性，可以通过产生受约束的随机激励来提高代码覆盖率，使得验证效率大大提高。此外，由于 UVM 各个模块功能划分明确，使得各个模块具有一定的独立性，这样可以很好的提高代码复用率，从而在长远上，大大缩短后期新开发的产品上市周期。由于以上各项优点，基于 System Verilog 的 UVM 验证已经成为多家 IC 公司验证的主流，并且还在不断发展。

本文的验证对象是在华虹集成电路有限责任公司实习时所参加的 TPCM 项目中的一款 MCU 中的由公司设计人员自主设计的 UART(通用异步收发传输器)模块。此款 UART 是在通用 UART 的基本功能上加以增强而来的，相对于通用 UART，多了多机通讯以及错误位检测功能，并且多了模式 0，在数据帧结构上有所不同。通过对 UVM 验证平台的基本组成，常用的标准库进行学习研究，并对各个组成模块的功能以及各个模块间的联系进行分析研究，利用 UVM 验证方法学，搭建适合 UART 验证的环境，并且为后期系统级验证做准备，在搭建验证平台时采用 AMBA2.0 总线架构，虽然这会增加验证平台的搭建难度，但是提高了代码复用率，是本文的重点。通过对 UART 功能进行研究分析，分析 UART 的工作特点，利用 UVM 验证方法学的特点，由 Sequence 模块产生相应的受约束的随机激励，通过 Monitor 模块监视需要验证的 UART 模块的输出，并且由 Scoreboard 模块进行结果自动比较。在验证平台搭建完成后，为了排除由于验证环境的问题而导致的结果不正确，需要对验证平台进行调试。在这之后，加入待验证的 UART。为了确保验证结果的正确性，对于最后的输出波形，选取具有代表性的波形进行分析。

在企业导师的带领下，使得验证方案有计划变为现实，功能覆盖率达到 100%，课题顺利完成。

关 键 词：System Verilog，UVM 验证方法学，UART，AMBA2.0，覆盖率

ABSTRACT

As integrated circuits has entered the era of super-mole, the increasing complexity of the circuit, and therefore the difficulty of verification are also increasing, with the improvement of verification required, in the actual production, traditional direct verification has become increasingly difficult to meet design needs.In order to make our products more competitive, to obtain an advantageous position in the fierce competition in the market, to ensure proper function under the premise of improving code reuse rate, it has become an important method of verification cycles each company to shorten time to market cycle is shortened. To do this, we need to adopt a more effective verification methods, making verification more effective.

In the market, driven by the increasing maturity of verification, only gradually into one. Has a high abstract UVM based verification of System Verilog, can improve code coverage generated by constrained random stimulus, such verification efficiency is greatly improved. In addition, since each module functions into UVM clear so that each module has a certain independence, so can good way to improve code reuse rate, which in the long run, greatly reducing the late development of new products to market. Due to the above advantages, UVM-based verification System Verilog has become more than the company verification IC mainstream, and still evolving.

Verify that the object of this paper is TPCM project at Huahong Integrated Circuit Co., Ltd. participated in the placement of a MCU in use by a company independent designer designed UART modules. Through the basic components of the UVM verification platform, commonly used standard library study and research, and to analyze the function of each component modules to study the association and individual modules, use UVM verification methodology, build verification environment for UART, and for the latter system-level verification preparing improve code reuse rate, is the focus of this article. Through UART research and analysis, analysis of the characteristics of UART using UVM verification methodology characteristics generated by the corresponding module Sequence constrained random stimulus, monitoring Reference Model and the need to verify the output module through UART Monitor module, and by the Scoreboard module automatically compares the results. To verify the results to ensure the correctness of the final output waveform, the

waveform selected representative analysis.

Thanks to the guide of enterprise menter, the verification platform became ture ultimately, achieve reasonable value of coverage, making the task completed successfully.

Keywords: System Verilog, UVM, UART, AMBA2.0, Coverage

插图索引

表格索引

符号对照表

符号	符号名称

缩略语对照表

缩略语	英文全称	中文对照
SSI	Small Scale Integration	小规模集成电路
MSI	Middle Scale Integration	中规模集成电路
LSI	Large Scale Integration	大规模集成电路
VLSI	Very Large Scale Integration	超大规模集成电路
ULSI	Ultra Large Scale Integrated	巨大规模集成电路
GSI	Grand Scale Integrated	超巨大规模集成电路
SOC	System On Chip	片上系统
UART	Uinversal Asy Receiver/Transmitter	通用异步收发传输器
DUT	Design Under Test	被测试的设计

目录

第一章　绪论

1.1 课题背景

在二十世纪五十年代，首块集成电路问世后，集成电路的规模由最初的 SSI(小规模集成电路)，MSI(中规模集成电路)，LSI(大规模集成电路)，按照摩尔定律不断发展至 VLSI(超大规模集成电路)，ULSI(巨大规模集成电路)，现如今，已经进入超摩尔时代，集成度越来越高，现在已经出现了 SOC 和 GSI(超巨大规模集成电路)。这意味着对于验证工程师验证工作变得日益复杂，无论是单个 IP 验证还是 SOC 验证，都对验证有了更高的要求。尤其对于 SOC，为了缩短设计周期，将各个 IP 集成起来，使得设计工程师的工作量减少，但是，这却更加加大了验证的难度。在实际生产中，经常会出现由于验证不充分，导致芯片生产后需要重新对设计进行返工，这对生产成本产生巨大影响。因此，验证需要不断提高，使得整个设计周期及成本下降。

1.2 国内外研究现状

如图 1.1 所示，在进行产品研发时，会有一份需求说明书，根据这份说明书，验证人员和设计人员会得到一份特性列表，研究人员会根据此列表的要求，编写代码，满足客户需求；根据列表上的测试点，验证工程师会拟定测试规划，确保设计符合设计要求。

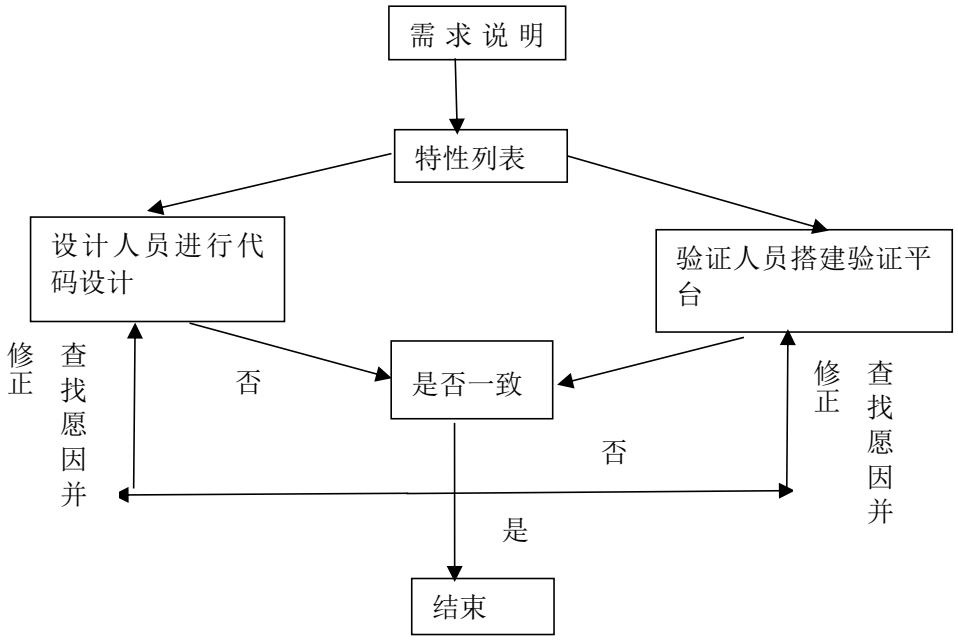

图 1.1　　验证与设计流程简介

在集成电路兴起时，验证平台的搭建是用 verilog 编写的，这使得设计和验证之间的连接很容易实现。这种验证方法对于小规模的电路设计而言，十分便捷，并且在早期，电路功能比较简单，测试点并不多，可行性比较高。验证时只要对设计代码进行实例化，并施加相应的激励即可，但代码的可再次利用率比较低，只要设计发生更改，测试平台可能就需要重新编写。但随着集成电路的规模不断变大，电路功能日益复杂，测试点越来越多，如果还使用 verilog 编写验证平台，由于 verilog 语言的抽象性比较低，层次性也不够强，不能够像 C 语言等高级语言那么灵活，很难覆盖到所有测试点，此外，由于代码的可再次利用率较低，修改验证平台将花费较长时间。

现如今，对于 SOC 级别的验证，很多公司由于技术和资金等方面的原因，采用直接验证(定向验证)的方法。即用 Verilog 语言编写一个 testbench，用 C 语言或 C++编写出针对不同测试点的 test_case，然后将产生的 hex 文件进行转换，对产品进行验证。由于 C、C++的抽象性较好，使得测试平台变得相对容易。但是，要想提高测试的代码覆盖率和功能覆盖率，就必须编写大量的测试样例使得每种可能性都被考虑，这需要投入大量人力和时间，并且收效甚微。但是，通过编写 MAKEFILE，使得测试能够按照设定的方式自动执行，减轻了验证人员的负担，但是代码的可再次利用率依旧不是很高，这在验证大规模的设计时，还是会花费不少时间，才能使覆盖率到达要求。

随着验证的需求逐步增多，越来越多的人从事验证开发工作，这使得验证领域发展迅速，现如今，出现了比较先进的方法，即使用随机化激励的测试方法。由于 System Verilog 语言具有一定的抽象性，所使用的类比较复杂，与直接验证所使用的 C 语言有很大的区别，这使得建立这种测试平台所花费的时间比直接验证(定向验证)要长很多。但是由于测试平台的各个模块有各自的独立性，代码的可重复利用率很高，避免了一旦测试要求发生变化，就需要验证人员重新编写验证代码的麻烦。由于是采用受约束的随机测试，查找 bug 的效率比直接测试要高出很多，这使得产品的返工率下降。如图 1.2，我们可以清楚得看到受约束的随机测试与直接验证(定向验证)在功能覆盖率上的优越性。

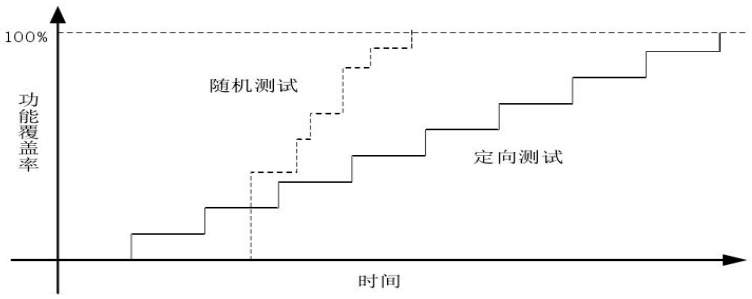

图 1.2　　定向测试与随机测试的功能覆盖率

随着验证要求不断提高，验证方法学也不断发展，验证方法学逐步形成一套完备的验证体系，出现了诸如 VMM，OVM 以及刚出现的 UVM 验证方法学，验证的方法形成体系，并将用到的基本模块构造成类似单元库的类单元，供使用者进行调用，并可以使用一些宏定义，使得验证硬件验证工作能取得如 C 语言等软件语言的灵活性，使整个验证工作更有条理可循，在进行 debug 时更加高效，并且使得代码的重复利用率提高，并且，由于这些高级验证方法有专门的收集覆盖率的方法，使得收集功能覆盖率变得简单，我们只需要按照验证方法的步骤，并调用相应的函数即可。

1.3 课题的内容及意义

本论文是在公司实习期间，根据项目要求，本小组需要对基于 AMBA2.0 架构的一款芯片进行 SOC 系统验证，本人需要对 UART 以及几个安全模块进行 UVM 验证。由于 UVM 验证方法学是基于 System Verilog 语言的，需要首先对 System Verilog 语言进行系统学习，对 System Verilog 语言的熟悉之后，进修 UVM 验证基本理论知识。在对验证有了一定了解之后，开始学习 ARMB2.0 协议以及 UART 通信协议，了解各个安全模块的功能。在进行验证时，对 UART 的通信方式，数据帧结构，以及接口方式和工作状态进行重点研究。根据通讯协议，以及 AMBA2.0 协议，基于产品的整体架构和工作方式，对产品进制定测试计划，剖析需要测试的功能点，对测试样例进行规划，确定设计平台架构，搭建 UVM 测试平台。在熟悉 UVM 库的基础上，对库中的模块根据测试要求进行扩展，并通过 TLM 接口进行各个模块的连接，完成各摸间的数据通讯，通过使用 virtual interface 将 DUT 接进测试环境，利用 UVM 验证方法中特有的 phase 机制，使整个测试平台各个模块间能有效的进行通讯交流。在受约束的随机激励这个条件下，完成各个测试点的测试，达到测试要求。本课题的研究在于学习 UVM 验证方法学，利用 UVM 搭建 AMBA2.0 总线架构的测试平台，使整个验证环境和现实情况更接近，满足验证计划。

1.4 本论文的结构简介

本论文所涉及的项目来源于实习时，公司正在研发的的一款 MCU 芯片，此芯片为一款低功耗芯片。本论文主要研究 ARMB2.0 协议，以 System Verilog 语言为基础的 UVM 验证方法学，UART，如何搭建 AMBA2.0 总线架构的 UVM 验证环境。章节安排如下：

第一章：粗略介绍验证的历史状况后，阐述研究是有价值的，并对个章节的大致内容进行简介。

第二章：主要介绍 UVM 验证平台的组成，所使用的接口以及常用的标准函数库，

验证的机制，并将 UVM 验证与 System Verilog 的验证方法行比较。

第三章：对需要被验证的 UART 进行功能研究。之后，对需要进行的工作拟定计划。

第四章：对验证平台的架构进行合理规划，分析数据流向，研究分析验证平台的各个模块功能。并简单介绍本项目在验证时所用的仿真工具和验证时所采用的流程，重点对仿真得到的重要仿真波形进行分析，最后获得覆盖率报告，表明实验的正确性。

第五章：在工作完成，对工作中的不足进行反思和归纳。

第二章　UVM 验证方法学基础

UVM(Universal Verification Methodology) 起源于 OVM 即 Open Verification Methodology，主要用于数字逻辑电路的正确性的验证。它与 System Verilog 有着密切的关联，在 System Verilog 的基础上，UVM 为数字电路验证提供了一系列的接口以及库函数，使我们能像在 C 语言中调用如 stdio.h 函数编写程序一样方便的编写验证平台。

2.1 UVM 验证平台

2.1.1　UVM 验证平台的组成

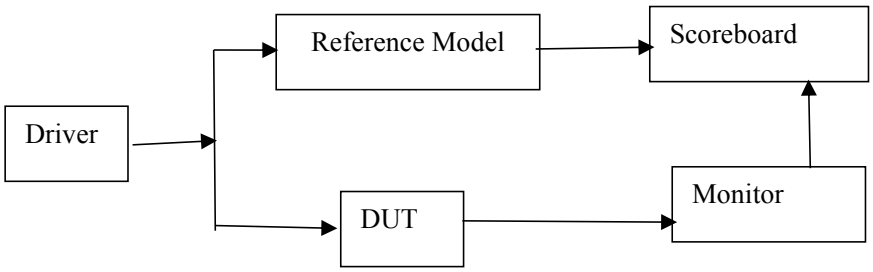

图 2.1　　验证平台组成

之所以需要对 DUT(design under test 待测设计)进行测试，就是为了在进行后期的布局布线时，没有功能上的问题。因此我们需要根据 DUT 的功能，对 DUT 施加不同的激励，再将结果与期望值进行比较的方法。

如图 2.1 所示，其中的 Reference Model 是一个与我们的 DUT 具有相同功能的模型。由于它接收到的激励与 DUT 相同，都是由 Driver 施加的，因此，我们只要在 Scoreboard 中比较 Reference Model 的输出和 monitor 监测到的 DUT 的输出即可。UVM 是建立在 System Verilog 语言上的，因此，也继承了 System Verilog 语言的特点，即采用类（class）来实现 Driver，Monitor，Scoreboard 等组成部分。下图 2.2，是 UVM 验证平台的树形结构。

1. transaction：对于 DTU 需要的事物级激励进行定义，通过施加约束，可以产生大量符合要求的数据，一旦设计增加新功能，修改时不需要重新搭建平台，只需要在 transaction 中增加或删去相应变量和约束即可。

2. Sequence：根据测试的要求对 transaction 进行进一步约束和例化，通过构造 sequence_lib 产生大量测试场景，使得测试具有完备性。并将例化得到的 transaction。

此外，Sequence 还可以控制测试平台的关闭。

3. Sequencer：通过 Sequencer 的 phase 开启 Sequence，获取来自 Sequence 的 transaction，再通过 sequencer 传递到 Driver。

4. Driver: 其主要功能是向 Sequencer 索要 sequencer_item(transaction)再将获得的数据给 DUT。完成由 transaction 级别的数据向 DUT 可以接受的 pin 级别的数据的转换。

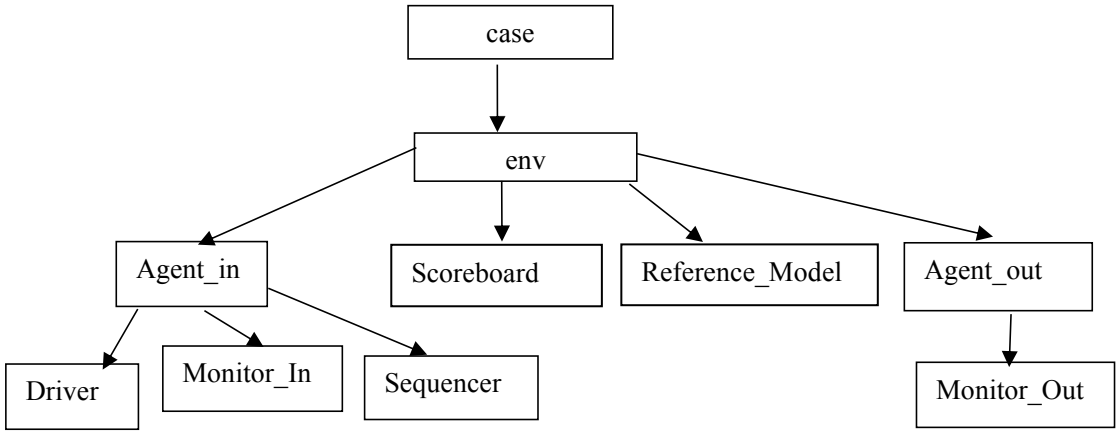

图 2.2　　验证平台树形图

5. Monitor_In: 其作用主要用于监测 DUT 的 interface，从 DUT 的 pin 上接收数据，并对接收到的数据进行转化，让数据转变成 TLM(Transaction Level Modeling)类的数据类型,这样才可以在整个验证平台实现数据流通。

6. Agent_In：其主要作用是将 Monitor_In，Sequencer，Driver 封装在一起，所以，一个 Agent_In 的主要行为由 Monitor_In，Sequencer，Driver 决定，DUT 的数据从宏观上可以说是来源于 Agent_In。由于需要对 Driver 进行实例化，Agent_In 是 UVM_ACTIVE 的。由于有了 Agent_In，就只需要 Agent_In 与 DUT 的物理组件进行通讯。

7. Agent_Out：它的主要作用是监测 DUT 的输出，因此，不需要有 Sequencer，Driver，只要有 Monitor_Out 即可。所以，它是 UVM_PASSIVE 的。

8. Monitor_Out：监测 DUT 是否有输出，当检测到有数据输出时，对数据进行解析提取，并发送给 Scoreboard。在 DUT 与 Reference Model 有通讯交流时，Monitor_Out 还需要将 DUT 输出的数据提取、解析后，通过 TLM 接口给 Monitor_In，传送给 Reference Model，使两者之间相互通讯。

9. Scoreboard：它的主要作用是比较 DUT 和 Reference Model 送来的数据是否一致，由此，判断 DUT 的功能是否正确。

10. Reference Model：完成与 DUT 相同的功能。与 DUT 不同的是，它不是采用 verilog 描述，而是用 System Verilog，并且，通过 DPI 等接口，可以调用其他语言实现 DUT 的功能。在测试时，经常使用如 SYNOPSYS 公司提供的 VIP(虚拟 IP)。

11. env：它将整个验证平台上所用到的不发生变化的组件都封装到一起，这样，在跑不同的 case 时，只要在每个 case 中实例化相应的 env 就可以了。

12. case：在 case 中需要对 env 进行实例化，只有这样，才可以把数据正常的发送到 DUT，并且，正常的接收来自 DUT 的数据。

2.1.2 UVM 验证平台的接口

1. 与 DUT 的接口——virtual interface

我们的测试环境的激励是 System Verilog 所编写的，还可以通过 DPI 调用 C 语言所编写的函数，再通过 task 任务施加给 DUT，但是，DUT 是由纯粹的 Verilog 语言构建的，在这种情况下，怎样才能将 System Verilog 语言的(class)类函数或者(task)任务中的激励传递给 DUT，并进行检测、接收呢？这就需要使用 System Verilog 中的 virtual interface 来解决。通过 virtual interface，可以便捷地将 DUT 的 input，output 和用 System Verilog 语言编写的相应模块连接起来。这样，DUT 就能接收 System Verilog 所编写的模块所产生的受约束的随机激励，而且，DUT 的 output 所发出的数据也能被相应的 UVM 模块检测到，并接收所需要的数据从而进行数据分析。在使用 virtual interface 前，必须先定义一个由 DUT 设计中存在的信号以及由 UVM 模块的通讯信号组成的 interface，即将 DUT 中的信号在 interface 中进行定义，也可以将时钟信号定义在其中，并将 UVM 的相应 model，例如 Driver 和 Monitor 与 DUT 通信时的信号定义在 interface 中。

2. 各个模块间的接口——TLM

在 UVM 中，要想实现各个模块间的通讯，可以有多种方法，例如 Driver 和 Monitor 间的通讯，可以在 Monitor 中定义一个全局变量，使 Driver 对这个全局变量进行赋值操作，实现与 Monitor 的数据交流。但是，在进行复杂的数据交流时就需要定义多个全局变量，这对验证平台而言，可能会造成不必要的影响；或者通过使用指针，使 Driver 能够访问 Monitor 中的非 local 变量。但是，这会使 Driver 对 Monitor 中的所有非 local 进行操作，在测试平台较大较复杂时，Monitor 模块与 Driver 模块是由不同人员编写的，如果 Driver 模块的开发人员不小心对 Monitor 模块的某些变量进行了操作，这会对整个平台的搭建带来不必要的麻烦。因此，在 UVM 中，一般使用 TLM(transaction level modeling)，供 UVM 中的模块相互通信。

在 TLM 中，有三种类型的 port，分别为：用于发起操作的 PORT；被 PORT 操控的对象——EXPORT；以及优先级最低，只能作为动作的承担者——IMP。因此，

EXPORT 可以向 IMP 发起操作。

通过调用 connect 函数，将有数据通信的各个端口相互连接起来，但连接时必须注意的是，函数的调用者必须是动作的控制者，被控制者只能作为函数参数。所以，PORT 可以对 EXPORT 和 IMP 调用 connect 函数，EXPORT 可以对 IMP 调用 connect 函数，但是，IMP 不能成为 connect 函数的调用者。

2.1.3 UVM 验证平台的类的标准库

在 UVM 中，有许多标准类库，通过对这些库进行合理派生，就能产生我们需要的子类，并且，在缩短设计周期的同时，也能保证准确性。其中，最最基本的类就是 uvm_object，对于像 uvm_sequence，uvm_sequence_item，config，uvm_phase 等都是从 uvm_object 中派生出来的。另一个比较重要的类就是 uvm_component，诸如 uvm_agent、uvm_driver 等 UVM 测试平台中的 model 大多是由它派生出来的，再通过这些派生的 model 库，可以构造出我们所需要的 model。

但令人惊讶的是，这两个基本类并非对立关系，uvm_component 派生于 uvm_object，因此，包含了 uvm_object 的所有特性。但是，uvm_object 却不包含 uvm_component 的所有特性。例如，在 uvm 测试平台中，每次构建 class 时，都会使用 super.new()这个函数来指明它的 parent，并利用 phase 机制来自动执行。而这两个特性只有在 uvm_component 中所具有，uvm_object 没有这两种特性。我们所用的如 agent，driver，monitor 等都是 uvm_component 派生出来的。如下图所示，我们可以看到在 UVM 测试平台中一些基本的派生关系。

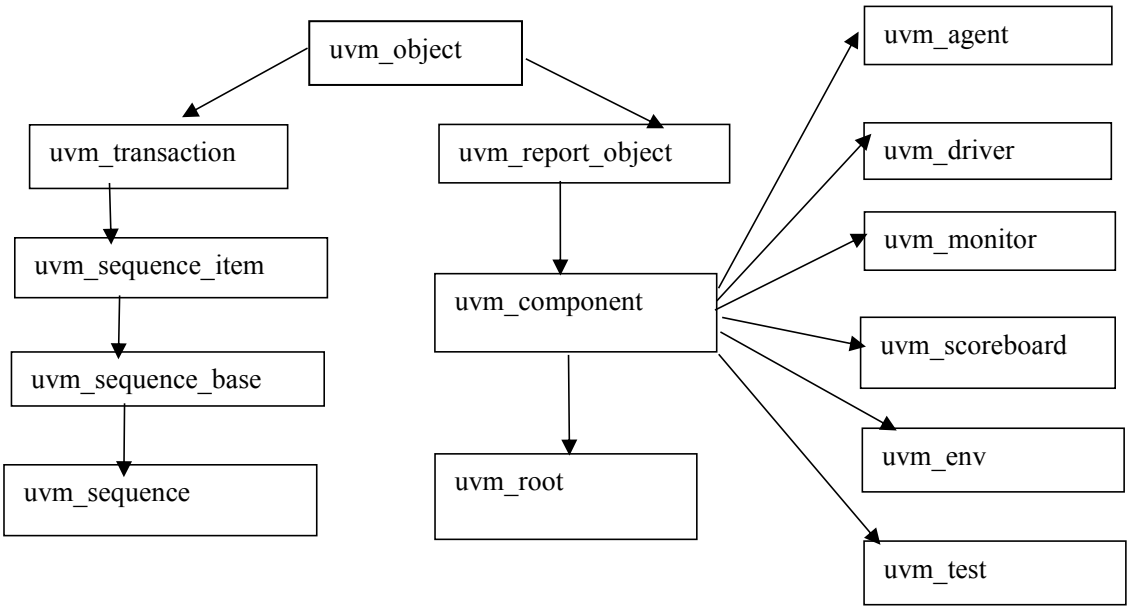

图 2.3　　UVM 中的基本关系继承图

2.2 UVM 验证的机制研究

在 UVM 中，由于存在多种机制，例如 field_automation 机制、factory 机制、sequence 机制，这些机制使得 UVM 验证平台能够方便，灵活地处理许多问题。

2.2.1 field_automation 机制研究

在 UVM 测试平台中，为了测试需要，需要有一些类似 print，compare 的函数来帮助验证人员进行 debug。虽然可以自己编写 print、compare 函数，但是，当需要打印或比较的内容发生变化时，函数就需要重写，这无疑会增加工作量，而且代码的继承性很低。这时，我们如果使用 field_automation 机制完成这些事，一切将变得相当便捷。

在需要使用 field_automation 机制的代码中，需要对变量使用 uvm_field_*宏，并在使用宏的开始加上`uvm_field_utils_begin，在结束时加上`uvm_field_utils_end，具体用法可以参考如下代码，否则，系统会报错。因为，field_automation 是建立于 factory 机制的基础上的。

```
class test extends uvm_sequence_item;
  rand bit [15:0] a;
  rand bit [15:0]b;
   ....
`uvm_object_utils_begin
`uvm_field_int(a,UVM_ALL_ON)
   ......
`uvm_object_utils_end
endclass
```

uvm_field_*系列宏多种，可根据实际情况使用，例如,需要使用动态数组，就可以使用 uvm_field_array_*，*为这个动态数组中将要存储的数据的类型。

除了可以直接调用 compare 和 print 外，还可以对数据进行 pack 和 unpack 处理。例如，在发送数据时每笔数据是以 byte 的形式发送的，如果没有 pack 操作，则需要将数据按照一 bit 发送，每次发送 8 次，并且，一旦数据的结构发生变化，就需要重写。但是使用 pack_byte 就可以很好的解决这一系列问题。它可以将数据的字段按照 byte 形式，放到一个数组中去。这样，在发送数据时，就不必考虑数据的具体定义，只要将数据的具体定义放到 transaction 中即可，这样一旦数据结构发生变化，也只需要将 transaction 中的定义修改即可。但是需要注意的是，在 UVM 中，pack_byte 函数返回的是数组中所有 bit 的和，并不是 bytes 的具体数值。在发送多 byte 时，要将

pack_byte 函数返回的值除以 8 后得到具体的 byte 个数，再通过 for 循环，对 byte 进行处理即可。同理可得 unpack 的用法，它的操作过程与 pack 恰恰相反。需要注意的是，在定义 transaction 时，一定要注意变量在调用宏时的顺序，因为在对数据进行 pack 和 unpack 操作时是按照 transaction 定义中的顺序进行的。

2.2.2 factory 机制研究

factory 机制更多地作用是体现在 UVM 的内部编码上的，是其他众多机制的基础，要是不使用 factory 机制，诸如 field automation 等机制就不能正常适用。

factory 可以被认为是一张表格，里面有许多宏定义。我们在编写 UVM 模块时，经常会使用 uvm_component_utils 或者 uvm_object_utils。这经常使刚刚接触 UVM 的人感到困惑，不明白这两个宏的作用。这两个宏的作用是对我们定义的类进行注册，这样，在创建类的实例时，就可以使用 type_id::create()了，并且能使用 UVM 的众多功能，要是不使用 uvm_component_utils 或者 uvm_object_utils 进行注册，则只能使用 new()来进行实例化，并且，很多 UVM 的功能不能使用，只能使用 System Verilog 中的功能。因此，可以说 factory 在一定程度上是对 new 函数进行重写，并使得原来的 new()函数的功能比 System Verilog 中多了很多。一个比较重要的功能就是 override 功能。

我们在编写 UVM 的一个 model 时，在跑大部分 case 时，是适用的，但对于一小部分这个 case 变得不适用，这时，应该如何处理呢？如果之前适用的 factory 机制进行注册，并使用 UVM 专门的实例化方法，这时，就可以使用 override 功能处理。这样，就可以从原来的基本类中衍生新的类。然后，对于具体的 case，在它的 build_phase 中，使用 set_type_override_by_type(A::get_type(),new_A::get_type()),就可 override 相关函数。这样，在跑到原来的 class(类)不适用的 case 时，就会调用新的类。这样，就不必将这几个特殊的 case 单独拿出来，重新搭建相应的 model，甚至重新搭建整个测试平台。

在 UVM 中，factory 机制的另一个重要的功能就是可以根据类名创建实例名。这或许看上去觉得并无特殊之处，但是，在进行 UVM 仿真时，这一功能就显示了它的强大之处。在进行 UVM 仿真时，系统之所以能根据输入的+UVM_TESTNAME=name 创建一个以 name 命名的类，就是因为 factory 机制中有 create_component_by_name 函数，它可以根据输入来创建以输入的字符串命名的类的实例。

2.2.3 Sequence 机制研究

在 UVM 验证方法学中，Sequence 机制是非常重要的一种机制，因为它不仅可以根据需要产生相应的 transaction，也可以控制整个验证平台的运行，在验证结束时，

关闭验证平台。在 UVM 中，代码再次利用率比传统的直接验证要高的原因就是在 UVM 中将不同的运行过程进行划分，使得在验证过程中，在特定的时间做特定的事，使得验证过程更加规范。这也是 UVM 验证的核心思想。

在使用了 Sequence 机制后，使得功能分工明确。因为在 UVM 机制中，Sequence 会产生 transaction，我们在不同的 case 中，将相应的 Sequence 设置成 Sequencer 的默认的 Sequence。这样，在程序执行到 main_phase 时，就会使用默认的 Sequence，并对它进行实例化，然后在调用其中的 start 函数，开始启动。default_sequence 的设置与执行如下图所示：

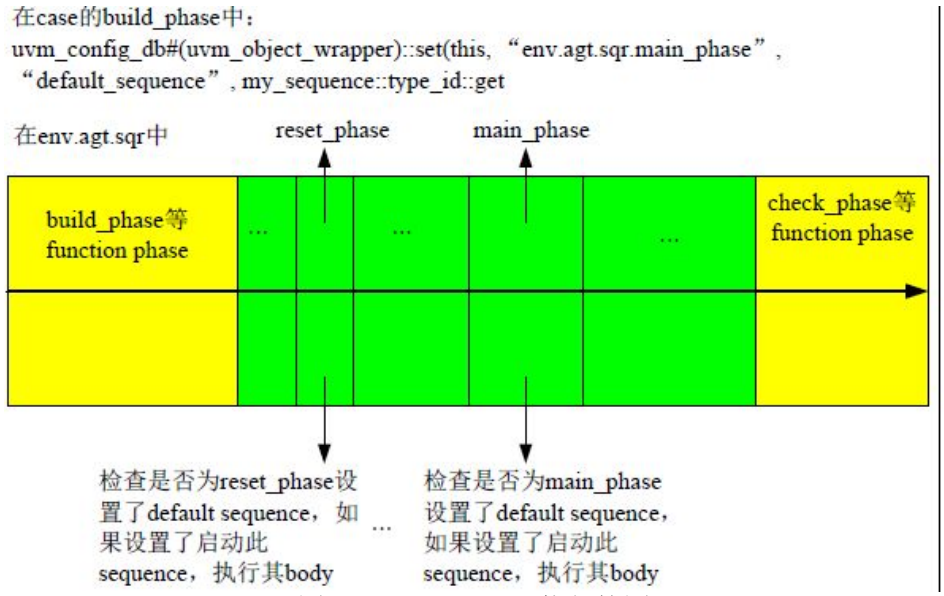

图 2.4 Sequence 执行简图

```
task body();
if (starting_phase!=null)
starting_phase.raise_object(this);
.....
if(starting_phase!=null)
starting_phase.drop_object(this);
endtask
```

由于 Sequence 是派生于 uvm_object 的，因此它不具备 uvm_component 中 phase 的功能。所以，我们需要有一个例如 starting_phase 的指针，在每次使用时判断是否为空，并让它指向 phase。这样，当 sequencer 启动 sequence 时，就可以把 main_phase 的 phase 值赋值给 starting_phase 这个指针了。通过此方法，控制验证平台的运行。

Sequence 机制的另一个强大的功能就是 virtual sequence。在验证中，我们需要对

DUT 进行初始化，这就要求只有在某个负责初始化任务的 Sequence 完成工作后，才能让其他的 sequence 发送数据。虽然我们可以通过定义一个全局的事件来控制，但是，在验证平台较大的情况下，不同的功能模块往往是由不同的验证人员进行编写，这就可能出现这个定义的全局事件在别的 model 中被意外触发，导致意想不到的错误产生，在进行 debug 时不易进行处理。此外，如果需要多个 Sequence 之间实现同步，使用定义全局变量的方式就会使验证平台的搭建变得更加繁琐。如果使用 virtual Sequence，就可以很好的解决同步问题。由于可以通过使用一系列 uvm_do 宏，可以将这些依次执行的模块按顺序打开。这样在 Sequence 中就可以按照次序执行。需要注意的是，在 Sequence 的代码编写中，如果需要开启多个进程，需要尽量避免使用 fork_join_none。由于 fork_join_none 的特性，不管前面的进程是否执行完毕，它都会执行，因此，我们以为已经将 sequence 执行完毕，将 transaction 传递出去，实际这个进程还没有执行就已经结束了。这在 debug 时，不易被察觉，对工作进程造成影响。为了解决这个问题，可以使用 wait fork 或 fork join 来解决。

在使用 virtual sequence 管理多个 Sequence 时，验证平台的关闭就由 virtual Sequence 进行控制，并不在每个 Sequence 中使用 starting_phase.raise_objection。因为在使用 virtual Sequence 时，我们使用 uvm_do 来控制这些 Sequence 的打开，因此，这时 Sequence 中指向 sequencer 的 phase 的指针为 null，而若是使用 starting_phase.raise_objection 则会产生问题。为了解决这个问题，我们在 virtual Sequence 中使用 starting_phase.raise_objection 来对验证平台进行控制。

2.2.4 report 机制研究

由于现在设计规模非常大，需要分析的信号也很多，工程师通过查看一个个信号波形来判读功能是否正确，变得不切实际。在 UVM 中，通过 report 机制，可以直接将我们需要的信息显示出来，这就减少了工程师的工作量，提高 debug 的效率。

在我们编写的每个 model 中，我们可以在代码中加入 uvm_error 宏来显示错误信息。通过设定 report 的冗余度，决定最后 report 显示的结果如何。Report 有 MEDIUM，LOW，HIGH 等几种，可根据实际需要决定报告的冗余程度。一个简单的 uvm report 如下图所示：

```
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :  568
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[conv_env]    19
[conv_mdl]   256
[conv_scb]   256
[demo]     4
[demo_seq]    32
```

图 2.5　UVM Report

2.3 UVM 验证与 System Verilog 验证

System Verilog 验证的重要突破在于使用了受约束的随机激励，面向对象的类，创造性地对接口进行抽象定义，使 DUT 与整个验证环境连接在一起，并且有专门收集功能覆盖率的模块，大幅度提高了工作效率。System Verilog 验证平台组成由下图所示。

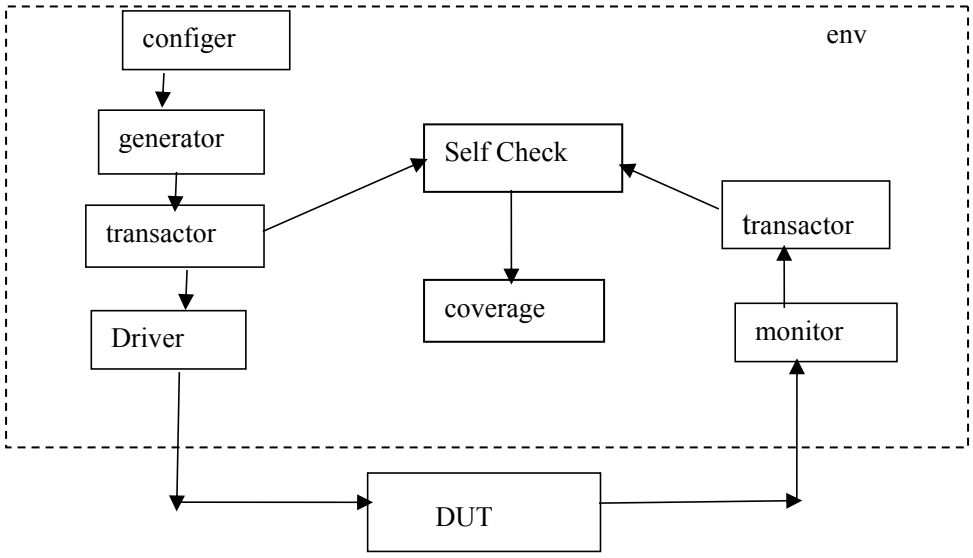

图 2.6　System Verilog 验证平台

在 System Verilog 中，transactor 的作用是将来自场景层的 generator 的命令进行分解。以 Flash 为例，对 Flash 进行读、写或擦除，分解后，传送给 Driver 和 Self Check，Self Check 会将 transactor 送来的命令保存在 Scoreboard 中，作为检查 DUT 输出的依照。Driver 接收来自代理 transactor 的命令，将命令进行分解，产生相应的激励给 DUT。

这类似于 UVM 中 Driver 的作用。System Verilog 验证中的 Monitor 的作用类似于 Monitor_Out，检测 DUT 的输出结果，并将结果给 transactor，通过 transactor 进行解析，送往 Self Check 模块，检查结果是否正确。Coverage 模块进行 function coverage 的检查。在 System Verilog 中，Self Check modle 包含了 UVM 验证方法中 Scoreboard 的功能，并有类似于 UVM 中 reference_model 的功能。在 UVM 验证中，有时可以看到在 Scoreboard 中包含类似 reference_model 的功能，这在 SOC 验证中对 Flash_Control 验证时，很常见。因为 Flash 主要作用是读，写以及擦除，没有必要通过一个 reference_model 模仿其行为来判断结果是否正确，我们只需要将 Driver 施加给 DUT 的激励分析出来需要的信息给 Scoreboard 作为评判 DUT 是否正常的标准，将它与 DUT 最后的输出对比即可。

System Verilog 验证的层次性与定向验证(直接验证)相比，已经清晰很多，UVM 验证方法学在 System Verilog 验证的基础上，各个模块进行了更加详细的功能划分，使得层次更加清晰。尤其是对于规模较大的设计，分工更加明确，在 debug 时效率更高。它将功能类似于 generator 的 sequencer，driver 以及用于检测的 DUT 输入的 Monitor_In 封装到 Agent_In 中，它们之间的通讯由专门的 TLM 端口连接，一旦有模块的因测试要求需要进行修改，不会出现牵一发而动全身的情况，这使代码的可再次使用率大大提高。此外，UVM 在继承 System Verilog 验证的优点的基础上，又开发出更多更加强大的库，例如 field_automation、factory 等，使得验证更加高效。这些都成为采用 UVM 的原因。

2.4 本章小结

本章主要研究 UVM 的测试平台的构建，首先研究测试平台中各个模块的功能与作用。然后对 UVM 测试平台的各个模块以及与 DUT 间的数据信息通讯分别进行研究。之后研究了 UVM 中常用的几种机制，如 field_automation，factory，Sequence，report，并对使用方法进行分析。最后通过和其他验证方法进行比较，对验证方法学有了更进一步的认识，这些都将为后续的工作提供理论基础。

第三章　UART 工作环境简介

本论文所涉及的项目来源于公司的 SOC 组设计的一款低功耗 MCU 芯片，目前仍在研发之中。该芯片主要用于保证电脑的数据安全性，防止数据被窃。本次需要被验证的 UART(通用异步数据收发器)，具有把获取的并行数据处理成串行方式后，给其他设备的功能。

3.1 UART 简介

本设计中所用的 UART 是根据实际用途而设计的一款同时具有异步、全双工串口，并且具有多种波特率可供用户进行配置。此外，支持多种工作模式，并且，这款 UART 可以作为从机，支持地址自动匹配。

与通用 UART 相比，项目中的 UART 多了模式 0，此工作模式在通用 UART 中是没有的。这个工作模式是为了在一开始工作时，测试 UART 是否正常上电，实际应用中基本不用，这也是为了安全考虑而加的功能。此外这个模式的数据帧结构与其他通用的 UART 不同，没有起始位和结束位，只有 8 位数据位。另外还多了数据结束位错误检测功能。这是为了当检测到接收的数据有错时，进行丢弃，并给出错误警告。因为此款 MCU 是为了保护数据的安全性，所以在数据安全性方面要求较高，UART 的功能也是根据此加以改进的。

3.1.1　基本工作状态分析

1. 模式 0 工作状态分析

将 SCON 寄存器的 SM0 位与 SM1 置“0”，可使 UART 工作在模式 0。在此模式下，UART 工作在同步移位半双工模式，其波特率为系统钟频率的十二分频。串口数据由 RXD 管脚输入或输出，同步移位时钟由 TXD 管脚输出。无论是发送还是接收操作，其数据长度都为 8 位（无起始位与结束位）。

在输出数据的时候，SCON.REN 位清“0”，并将需要输出的信息写入 SBUF 寄存器；数据将从 RXD 管脚移出（最低位先出，最高位后出），同步移位时钟由 TXD 输出；完成发送数据任务后，SCON 寄存器的写操作中断标志位 TI 将被置“1”，在中断请求被允许时，将会产生中断操作，即对 SBUF 进行写数据。

接收数据时，将 SCON.REN 位置“1”，并将 SCON.RI 位清“0”。数据经过 RXD 输入（最低位先进，最高位后进），同步移位时钟通过 TXD 输出。完成接收数据任务后，SCON 寄存器的读操作中断标志位 RI 变成高电平，在发出的请求被许可的条件

下，将产生中断操作，此时读 SBUF 寄存器将会得到 UART 接收到的具体数据。

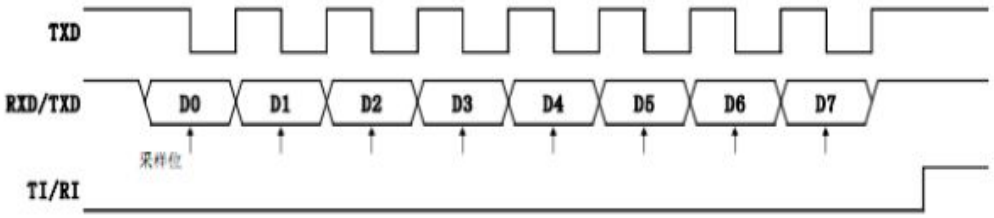

图 3.1　　模式 0 接收/发送简图

2. 模式 1 工作状态分析

UART 工作在异步全双工模式(全双工应该使 SCON.REN=1，如果 SCON.REN=0，只能发送不能接收)，其波特率可由寄存器 TM 配置。数据由 RXD 管脚输入，由 TXD 管脚输出。将 SCON.SM0 清“0”与 SCOM.SM1 置“1”，可工作在模式 1 下。

在进行发送操作时，必须把数据存入 SBUF 寄存器；数据将被从 TXD 管脚移出（最低位先出，最高位后出），完成发送数据任务后，SCON 寄存器中表示写操作中断的标志位 TI 将呈现高电平，在发出的请求被许可的条件下，对 SBUF 进行写数据。

在进行接收操作时，数据通过 RXD 输入（最低位先进，最高位后进）。完成接收任务后，SCON 寄存器中表示读操作中断的标志位 RI 变成高电平，在发出的请求被许可的条件下，以读的形式访问 SBUF 寄存器，会得到 UART 接收到的具体数据。

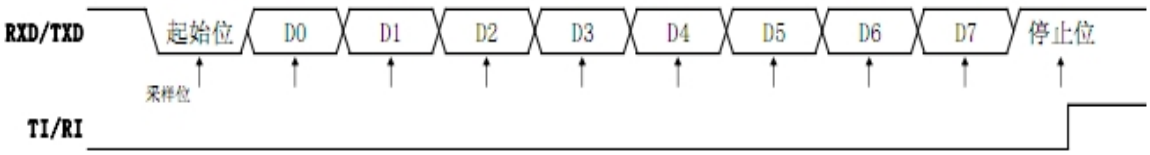

图 3.2　　模式 1 接收/发送简图

3. 模式 2 工作状态分析

UART 工作在异步全双工模式(全双工应该使 SCON.REN=1，如果 SCON.REN=0，只能发送不能接收)，其波特率是固定的系统时钟/32，或者系统时钟/64。需要转换的数据通过 RXD 进入，然后通过 TXD 移出。在此模式下，无论是发送操作还是接收操作，其数据长度都是 11bit，根据工作需求，通过设定可编程位的值，使 UART 对数据进行奇偶校验，或者进行多机通信操作。将 SCON.SM0 置“1”与 SCOM.SM1 清“0”，可工作在模式 2 下。

在进行发送操作时，必须把数据存入 SBUF 寄存器；TXD 管脚移出数据（最低位先出，最高位后出），完成发送数据任务后，SCON 寄存器的写操作中断标志位 TI

将被置“1”，在中断请求被允许时，将会产生中断操作，即对 SBUF 进行写数据。

在进行接收操作时，RXD 管脚输入数据（最低位先进，最高位后进），在完成接收任务后，SCON 寄存器中表示读操作中断的标志位 RI 呈现高电平，在发出的请求被许可的条件下，以读的形式访问 SBUF 寄存器，会得到 UART 接收到的具体数据。

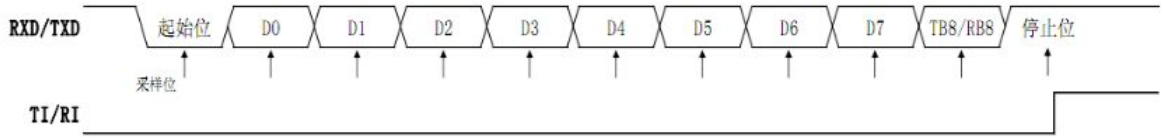

图 3.3　　模式 2 接收/发送简图

4. 模式 3 工作状态分析

将 SCON.SM0 置“1”与 SCOM.SM1 置“1”，可工作在模式 3 下。通过配置 TM 寄存器，可以改变 UART 的波特率，这是模式 3 与模式 2 唯一的不同，发送/接收数据与模式 2 相同。

5. 数据停止位错误检测功能分析

数据停止位错误检测功能可在模式 1,2,3 下使用，将 PCON.SMODx0(x=0, 1)位置“1”将启用该功能。接收数据时会检测有效停止位，若未发现有效停止位，则将 SCON.FE 位置“1”。软件在每次接收数据后检测 SCON.FE 位，以检查是否有数据传输错误。硬件置“1”后，只有通过软件清“0”或者复位键将 SCON.FE 位清“0”。如果未经过软件清“0”,那后续而来的具有有效停止位的数据帧也不能将 FE 位清“0”。

3.1.2　多机通信模式分析

只有模式 2 以及模式 3 的数据位中有可编程位，因此只有这两种模式有多机通信功能。当一台设备有数据需要发送给多个其他设备时，首先需要确定发送的地址字节，此时要求可编程位设定为逻辑“1”，并且接收数据的设备的 SCON.SM2 位被置”1”。接收设备在收到有效停止位和地址后，产生中断，从而使接收到的数据字节有效，直到消息发送完毕，此接收设备的 SCON.SM2 重新置“1”。未被寻址的设备则不会产生中断，主机发送的数据对它无效，不会被接收。地址和设备不需要一一对应，即从机的地址可以不唯一，也可以使多个从机共享一个地址，从而达到一台设备在不同功能下接收不同信息，多台设备接收到相同信息的目的。

对多机通信中的地址设定进行详细地解析如下所示：

只要从机地址寄存器 SADDR 或从机掩码 SADEN 寄存器中的数据发生变化，被寻址的从机就会发生变化。根据获取方式，从机的地址又分为具有独立性的指定地址和具有不确定性的广播地址。

指定地址：

每个设备有一个独立的地址，那就是在寄存器 SADDR 中设置的指定地址，而 SADEN 寄存器可以忽略掉一些 SADDR 中不关心的位（0 为忽略的无关位），而那些不关心的无关位提供了一个大范围的指定地址。在只对一个从机进行寻址时，SADEN 寄存器的值必须设置位 0xFF。

广播地址：

将 SADDR 以及 SADEN 中的数值进行或操作，就可以得到我们需要的广播地址。在广播地址中，为了具有更多灵活性，允许有无关位存在，但是在多数应用中，广播地址常常为 0xFF。

例如有下设置：

SADDR 被设置为： 01101001

SADEN 被设置为： 11111011

那么指定地址是 ：01101x01(x 表示无关位)，即从机的指定地址是 01101001 和 01101101。

广播地址是：11111x11(x 表示无关位)，广播地址是 11111011 和 11111111。换句话说，主机可以和从机通过 4 个地址通信 01101001 和 01101101（给定地址）11111011 和 11111111（广播地址）。

3.2 验证环境规划

在当代，时间与市场紧密挂钩，谁能在最短时间内开发出高性能的新产品，就能在市场中抢占先机，使利润最大化。在进行嵌入式微处理器开发时，可以采用 AMBA 总线架构来缩短开发周期。此外，ARM 公司开发的 AMBA 总线还提供测试接口环境，这使得验证的时间也缩短。

本论文所涉及的项目采用了 AMBA2.0 架构，为了方便日后进行系统级验证，避免在进行系统级验证时需要重新处理接口，提高代码复用率，所以对 IP 的验证也采用和系统级验证相似的环境，在验证中也引入 AMBA2.0 总线架构。因此，需要在验证平台中加入具有主机功能的 AHB 模块，用于读写待测设计以及对寄存器进行控制，需要有具有从机响应功能的 APB 模块，用于连接 APB 和 AHB 的 APB 桥，以及一个用于交互通讯的 UART。综合考虑后，决定采用 SYNOPSYS 的 AHB VIP 实现 AHB 主机功能，APB VIP 实现 APB 从机功能，UART VIP 实现交互通讯的功能。在验证时，需要模拟出 UART 复位，恢复正常工作，产生需要发送给 DUT 和 UART VIP 的信号等，并需要有模块对 DUT 发出的数据和 UART VIP 的数据进行比较和正误判断。

对于测试点可以分为概括成以下几点：

1. 先使进行环境调试，确保环境参数正确，然后用默认模式进行调试。这可以通过将待测模块先不接入，通过将单独对环境进行调试时的测试样例再跑一遍来实现。然后，对基本功能进行测试。在我们设计的 UART 中，模式 0 是最简单的收发数据功能，我们可以先测试模式 0 下功能是否正确。

2. 确保各个工作模式下，DUT 和 VIP 之间能进行数据传输，对于波特率可以改变的工作模式，需要确保数据传输时的波特率是配置值。查看波形时，重点关注表示处于数据发送状态的 starttx_r、处于数据接收状态的 startrx_r、用于对 sin 信号上数据打拍的 txtick、存储向 APB 总线写数据的 pwdata、接收到的来自 APB 总线传输过来的数据 prdata 信号。

3. 对于非 0 模式下的数据停止位错误检测，在产生激励时，必须将停止位设定成非 1，查看波形时，根据 txtick 判读数据帧格式是否正确、表示数据结束位是否有效的 framerr 信号是否正确。

4. 多机通信测试，查看波形时，需要查看广播地址信号 broadcast、从机特有地址 saddr、来自 UART_VIP 的寻址地址 given 信号、从机掩码地址 saden 这几个信号的值与期望是否一样。

如果报告显示有错误，需要根据报告内容以及相应的波形文件进行错误分析。

3.3 本章小结

本章首先对这次验证所需要的 UART 的功能进行简单介绍，并对相应的 5 种基本工作模式的功能以及数据帧结构进行分析，并对多机通讯模式进行研究分析。然后对本项目需要使用的验证平台进行初步的规划。

第四章　UART 的 UVM 验证技术研究

由于此次使用的 UART 是根据项目需求而设计的，因此，十分有必要对这个 UART 进行验证。为了能够全面的对 UART 进行检测，通过使用 SYNOPSYS 公司的 UART VIP 对我们设计的 UART 进行通讯，由此判断功能是否正确。

4.1 UART 验证平台架构

这次验证的 UART 的功能主要是数据通讯和多地址通讯。在数据通讯时有四种工作状态，有两种工作模式下，波特率可由用户进行配置，另外两种模式的波特率与系统时钟相关。为了检测 UART 的通讯功能是否正确，必须要一个功能相仿的外设对它进行数据通讯，因此，就需要有一个 reference model 作为参考。为了验证的方便，采用 SYNOPSYS 公司提供的 UART VIP 作为 reference model。为了在项目后期进行的工作需求，虽然只是对一个 IP 的功能验证，在验证平台上，还是采用 AMBA2.0 的总线架构，并使用 UVM 验证方法学。实际采用的验证平台如图 4.1 所示：

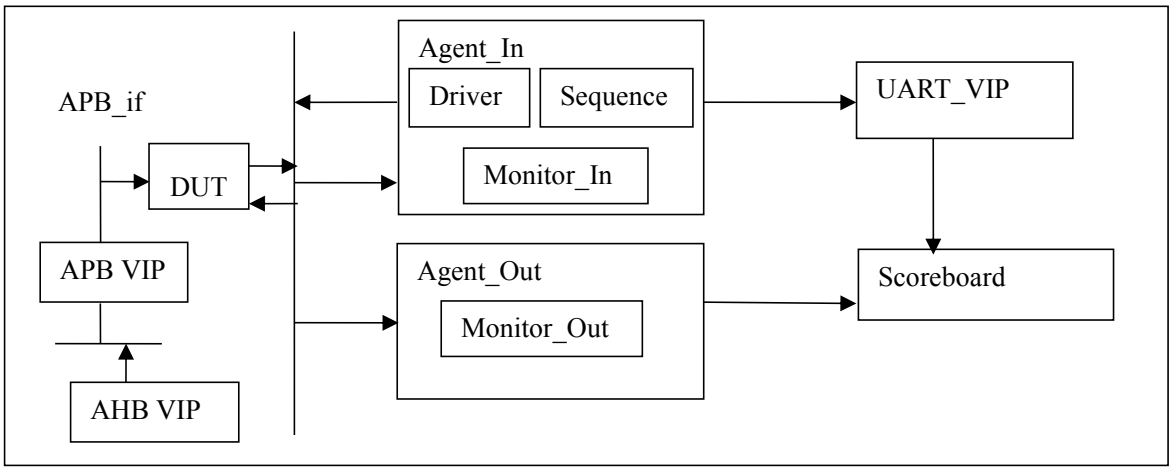

图 4.1　　实际 UVM 验证平台

1. AHB VIP：为需要被验证的UART产生在真实的工作状态下可能会遇到的各种情况。使数据的读、写能合理进行，遵循AMBA2.0协议。它有AHB_Master和AHB_Slave。当UART作为从机时，用AHB_Master。当UART作为主机时，用AHB_Slave。此模块采用SYNOPSYS的AHB VIP。

2. Agent_In：将Sequencer以及Driver还有Monitor_In封装在一起，主要负责向UART以及Reference Model发数据。数据由Sequence产生，由Sequencer送给Driver。Driver根据transaction里存储的信息，给UART和Reference Model产生相应的激励。此

时为UVM_ACTIVE。

3. Agent_Out：由于只需要对UART(DUT)的输出进行检测，并把结果通过接口传送给Scoreboard模块，因此，只需要有Monitor_Out模块即可。此时为UVM_PASSIVE。

4. Scoreboard：主要用于比较UART(DUT)和Reference Model的输出是否一致。

5. uart _if：将UART(DUT)的实际的输入输出和测试平台的相应部件进行连接，使它们之间能进行正常通讯。

6. APB_if：将UART (DUT)与APB进行连接，使它们之间能进行正常通讯。

7. UART_VIP：作为外设与UART(DUT)进行数据通信。

4.1.2 UVM 验证平台执行流程

通过Makefile，调用VCS仿真器。使用VCS仿真器进行仿真时，系统会第一个执行的就是对DUT进行实例化的uart_top.v文件。

在 top 中，实例化时，通过 config 机制，将 $root.test_top.ahb_if 的值给 uvm_test_top.env.ahb_system_env 中的 vif。在 top 中会有一个 run_test()函数，这个函数在整个 top 文件中占有重要地位。此函数是一个全局函数，当运行完 run_test()这个函数，就会开始自动启动验证平台，大致执行流程如下所示：

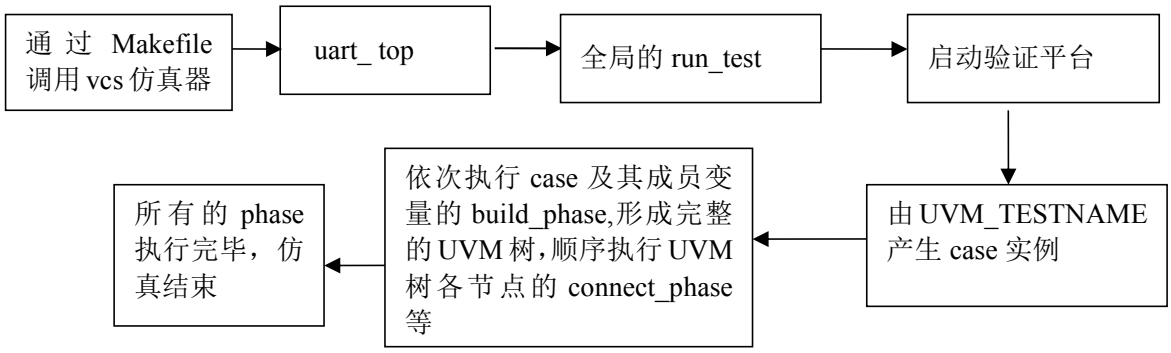

图 4.2　　UVM 验证平台执行流程

4.1.3 数据流向描述

UART(DUT)的寄存器通过读取来自AHB_VIP的信息，决定对外设采取何种操作。AHB_VIP对会根据需求，对我们的UART(DUT)的相应的寄存器进行的合理配置，通过相应的test_case,发送相应的数据。

当外设UART_VIP对UART(DUT)发送数据时，发送的数据由Agent_In中的Sequencer产生，在将数据给Driver。Driver通过收到的transaction产生对应的激励并发送标志位，通知UART(DUT)准备接收数据。此时，在Agent_In中，Monitor_In会检测到总线发生了变化，有数据需要传递，相应的标志位会起作用，将发送给UART(DUT)

的数据复制一份，然后将其送给UART_VIP, UART_VIP将数据给Scoreboard。Agent_Out中，Monitor_Out检测UART(DUT)的输出，并将结果送给Scoreboard进行比较。

当需要对外设发数据时，读取AHB_VIP中产生的要发送给UART_VIP数据和标志位信息。Monitor_Out通过检测UART(DUT)输出的变化，并将数据通过TLM接口传送给Monitor_In，最后发送给UART_VIP。在Agent_Out中，Monitor_Out检测UART(DUT)的输出后，还会将结果送给Scoreboard进行比较。

4.2 各模块功能介绍

4.2.1 总线行为功能模块

AHB_VIP，APB_VIP和 UART_VIP都采用SYNOPSYS公司的用于UVM验证的IP。所用的代码均被加密，因此，我们无法看到内容。在实际使用时，我们需要在验证环境中说明所在路径，并对相应的组件进行实例化，不需要我们编写代码。

1. AHB_VIP

本次验证采用 UVM 格式的代码，使用时，需要说明代码路径，此 VIP 既可以作为 master,也可以作为 slave,具体的作用可以在实例化的时候通过设计相关参数进行配置。在本次验证中，AHB_VIP 既需要做 master，也需要做 slave，具体的应用由仲裁器决定。

AHB_VIP 通过 register_model，可以根据测试的需求，实现对 DUT 的相应的寄存器进行相应的处理。

2. APB_VIP

由于本项目中的 UART 的数据位宽比较小，传输速度并不快，因此，不需要作为 master，挂在在 AHB 总线上，只需要挂在 APB 总线即可。此次的 APB_VIP 来源于 SYNOPSYS 公司。使用时，值需要根据需求对相关参数进行配置即可。

3. UART_VIP

在本次验证中，UART_VIP 的作用就在于与 DUT 进行数据通信，通过收发数据的结果来验证 DUT 的发送接、收功能是否正确。在对验证环境进行设置时，需要在验证环境中的相应文件中，说明读取 UART_VIP 源代码文件的路径，这样，就可以正常使用。

4.2.2 事物数据包

在 UVM 验证平台中，以 System Verilog 语言编写的 component 之间的通信，采用的是 TLM 方式。所以，transaction 在整个测试平台间流动。各组件之间的通信是

采用收发数据包的形式进行的，不像 DUT 的各个 module，以 bit 为通信单位。为了能使数据流通，需要将与 DUT 接口相关的物理数据抽象为事物数据，以包的形式进行传输。在 UART 发送据为例，它的 transaction 形式如下：

表 4.1　UART 发送数据时的 transaction

SOF	Flags	Mode_Type	数据长度	奇偶校验	EOF
1bit	8bit	2bits	4bits	3bits	1bit

Flags 位通知 UART 开始准备工作，奇偶校验位则进行数据检查，如果数据不正确，则进行丢弃。其部分代码如下：

```
typedef    enum bit {send,receive,idle} type;
class transaction_in extends uvm_sequence_item;
rand bit[3:0] data_length;                    //UART发送的数据长度
rand bit[7:0] data;
rand bit[3:0]mode_type;                       //UART工作模式说明
rand bit[3:0]    baurd;                       //UART波特率说明
rand bit extrabit；
rand bit program;                             //是否产生可编程位
rand bit communication;                       //是否多机通讯
rand bit [7:0] saddr;
rang bit [7:0] sadden;                        //产生多机通讯的地址
……
constraint    work_type _constrain{
work_type    inside {send,receive,idle}}
 ……
 `uvm_object_untils_begin(transaction_in)
 `uvm_field_int (mode_type,UVM_ON)
 ……
`uvm_object_untils_end
function transaction_in::new(string name=“transaction_in”);
super.new(name);
endfunction
endclass
```

在事务数据类中有两种数据类型，一种是起到控制作用的数据，用于控制往

interface发数据，例如Driver可以根据communication的值，决定是否发送saddr等信号。communication决定了是否为多机通讯状态，如果是，则将saddr和saden信号往外发。另一种是实际的激励，例如start_bit,program,这些都是实际的激励数据。但在模式0中，数据中没有起始位和可编程位，所以，这些激励必须要通过mode_type等信号进行判断。

为了产生随机数据，使实验结果更具有完备性，也为了能检测到更多的可能出现的情况，在transaction中，可以使用rand()函数。其部分代码如下：

```
function int genRndNum(int unsigned min,max ,mult=1);
this.rndNumMin=min;
this.rndNumMax=max;
this.rndNumMult=mult;
assert(this.randmize())//使用断言，当条件满足时，执行，否则执行else语句
else $fatal(0,”Gen Random Number :Randomize failed”);
genRndNum=this.rndRum;
endfunction
```

在验证中，当我们使用 field_automation 机制时，可以直接实现 pack，但不需要编写函数。这样，使得 transaction 的编写变得相对容易，并且，当数据包需要改动时，不会使整个模块重新编写。但需要注意的是，在 UVM 中一个 object 在使用 field_automation 时，必须要在 uvm_object_utils_begin 和 uvm_object_utils_end 之间加上 uvm_field_*，要是没有 uvm_object_utils_begin 和 uvm_object_utils_end，系统在编译时会报错。

4.2.3 Sequence 模块功能分析

在 DUT 刚启动的时候，我们需要对 DUT 的寄存器行进配置，当配置完成后，我们才可以向 DUT 发送数据。这样，DUT 才能工作在我们想要的方式下。为了实现这一目的，在 UVM 验证中，可以采用 virtual sequence 来实现。

Sequence 产生 transaction，通过 Sequencer 交给 Driver。要想产生不同的激励，需要借助于 Sequence modle 的功能。不同的激励也就是不同的 case。case 间的差异主要体现在 Sequence 上。通过控制 Sequence，我们可以产生不同的激励，使测试具备完备性。

每个 Sequence 虽然有差异，但是，它们还是有共同的地方，因此，可以通过用一个 base_sequence 衍生出不同的 Sequence。这个 base_sequence 派生于 uvm_sequence，其部分代码如下：

```
class base_sequence extends uvm_sequence #(transaction_in);
```

```
`uvm_object_utils(base_sequence)
.....
function new(string name="base_sequence");
super.new(name);
   .....
`uvm_info("TRACE",$sformatf("%m"),UVM_HIGH);
endfunction
task pre_body();
if(starting_phase!=null)
starting_phase.raise_objection(this.get_type_name());
endtask
 task pos_body();
 if(starting_phase!=null)
starting_phase.drop_objection(this.get_type_name());
endtask
endclass:base_sequence
```

在这段代码中，我们可以看出，验证平台的打开与关闭可以通过 Sequence 来控制。

通过 raise_objection 和 drop_objection，可以控制 run_phase 的执行。在代码的第一行，base_sequence 参数化了 transaction_in，这个参数在所有 Sequence 中都会有。不同的 Sequence 根据测试要求的不同，对参数进行实例化。以发送数据为例，其部分代码如下：

```
class seq_send_base extends base_sequence;
rand bit[7:0] data;
rand bit[1:0]mode_type;                          //UART 工作模式说明
rand bit[3:0]    baurdrate;                        //UART 波特率说明
....
constraint mode_type_dist{0:=10,1:=50,[2:3]:=60}；
                                    //对四种工作模式的出现的权重进行进行控制
task genRndPkt(input int length,output packet pkt);
//对发送数据的大小进行限制
int length=this.genRndNum(min,max,mult);
for(int i=0;i<length;i++)begin
pkt.push_back($random_range(0,255));
end
endtask
constraint baurd_size{
```

```
if(mode_type==mode0)    baurdrate=system_clk/12;
if(mode_type==mode2)    baurdrate inside{system_clk/32,system_clk/64}};
                                                    //波特率大小限定与工作模式相关
```

在 seq_send 对发送数据所需要的所有变量进行约束，这些变量包括了产生各种场景，包括正确和错误场景的产生。由于模式 0 只是测试是否上电工作，实际应用中不会使用，对它的测试不是主要的，因此，可以降低它的权重，缩短测试时间，提高测试效率。通过 dist 操作符，我们可以实现这一目标，模式 0 将它的权重设为 1/18, 模式 1 为 5/18,其余的模式的权重均为 1/3。数据的结构与工作模式相关联，除了模式 0 外，其余模式下，数据都有起始位和结束位，要对发送的数据结构进行限制，此外，还需要对波特率进行限制，除了模式 1、3 可以由用户根据要求设定波特率的值，其余模式的波特率均与系统时钟相关。通过 constraint baurd_size，我们可以实现这些要求。场景注错是模式 2 和模式 3 下，监测不到停止位时发生的，此时，需要将发送数据的结束位置 0。

```
virtual task body();
  'uvm _create(req)
assert(req.randomize());
req.flag_value={1'b0,option_m,data_length....};
req.data=data;
.....
`uvm_ send(req)
endtask
```

以上代码中，通过 uvm _create()函数来发出请求，要求 Sequence 产生激励。uvm_ send 用于要求 Sequence 发送激励。data 为激励的实际内容。UVM 的一大优点是采用随机激励，这有利于找到我们有时忽略的 bug。req.randomize()实现对 req 的随机化。在 seq_send_base 中重新约束的参数将赋值给 req 参数，比如 req.flag_value={1'b0,option_m,data_length....}，同时将不需要的字段进行关闭，避免资源浪费以及产生不恰当的激励，比如在发送数据时，将多机通讯模式关闭，则通过将 multi_communication=0, 可以实现。通过 uvm send(req)，将进一步约束好的 transaction_in 发送给 Sequencer。

在 seq_write_*将对在 seq_send_base 中定义的参数进行实例化，产生实际的激励。比如，在模式 2 和模式 3 中，发送数据，其部分代码如下：

```
class seq_write_2or 3 extends base_sequence;
seq_send_base    seq_write_2or 3;
  virtual task body();
```

```
`uvm_create(seq_send_base)
seq_send_base.program=1'b0;
seq_send_base.sel=1;
seq_send_base.length=4'b1001;
assert(seq_send_base.randomize());
......
`uvm _send (seq_send_base)
endclass;
```

因为在默认情况下，UART 是处于模式 0 下，在随机化前打开选择模式，则就可以产生非模式 0 的模式，再将 length 设置为 9，则只能出于模式 2 或模式 3。将 program 赋值为 0，则只能出非多机通讯模式。在验证过程中，所有的数据通信都是以数据包的形式进行的，通过对相应参数进行例化，产生需要的激励、测试场景。根据验证需要，数据包可以单独传输，也可以组合后使用。例如，在模式 2 和模式 3 下，接收到 VIP 发送的数据后，再发送出去。其场景的部分代码如下：

```
class re_se_mode2_3 case extends base_sequence;
......
seq_read_2or 3    seq_read_23_1 ;
seq_write_2or 3    seq_write_23_1    ;
virtual task body();
`uvm _do(seq_read_2or 3    seq_read_23_1) ;
`uvm _do(seq_write_2or 3    seq_write_23_1) ;
endtask
endclasss
```

在 seq_read_23_1 将特定功能的数据包实例化，由于只是接收数据然后再发送出去，并不需要对接收到的数据进行处理，因此不需要先利用 uvm_create 进行空间分配，再使用 uvm_randomize 进行随机化，然后通过使用 uvm_send 进行发送，直接使用 uvm_do 可以将这三步一起实现。

通过使用 UVM 机制，我们可以方便的产生所需要的激励，尤其在场景复杂，指令较多时，它的优势尤为明显。虽然在发送、接收数据时的激励和参数不同，但是其数据的基本帧格式是相同的。因此，可以将相同的部分作为基础，即 base_sequence，通过对这个基本类进行扩展，得到不同功能的基本类模块，例如发送的基本类 seq_send_base、seq_rec_base。以发送数据为例，在不同模式下，发送的数据会有所差异，通过将发送的基本类模块 seq_send_base 中的参数进行实例化，可以达到发送相应模式下激励的目的。将这些具有实际意义的 Sequence，例如 seq_send_0，seq_rec_0

按照一定的次序进行组合，得到测试所需要的实际场景，即 case。

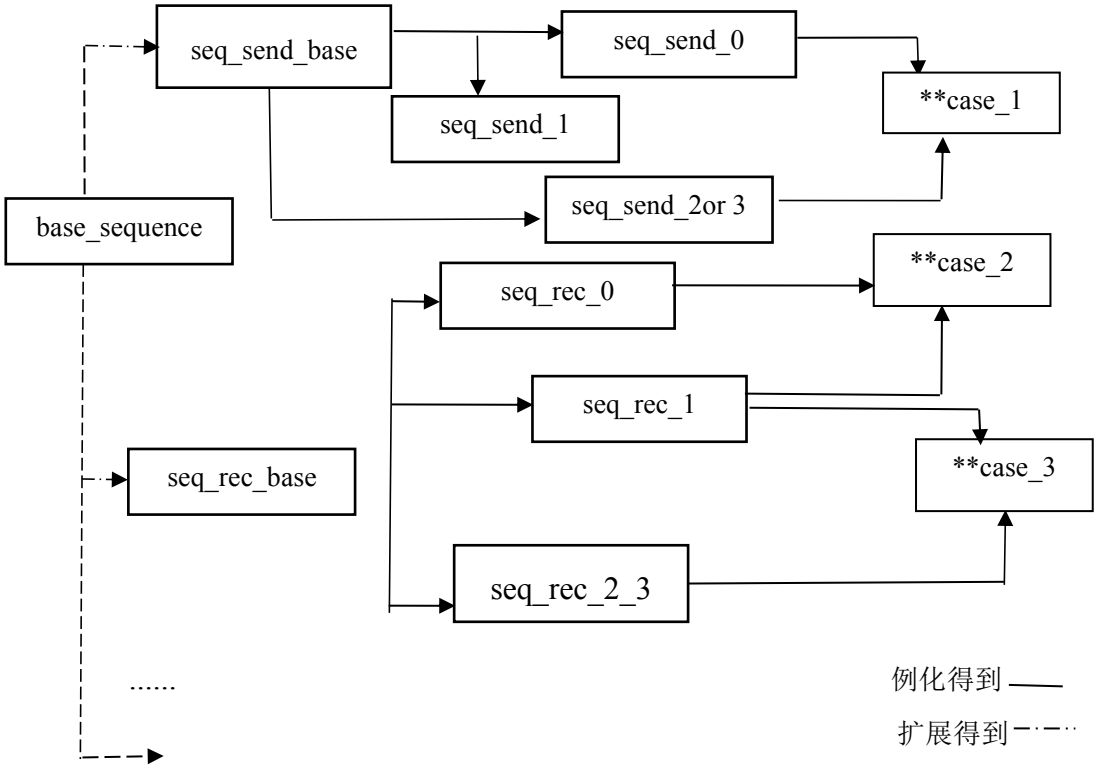

图 4.3　　sequence_lib 简图

4.2.4　Agent_In 模块功能分析

Agent_In 将 Sequencer、Driver、Monitor_In 封装在一起，这样，就只需要 Agent_In 模块与 DUT 打交道，所以，可以说是 Agent_In 负责向 UART(DUT)发送数据。在 Agent_In 中实现 Sequencer、Driver、Monito_In 的实例化，还有连接，部分代码如下：

```
class Agent_In extends uvm_agent;
uart_sequencer #(transaction_in)sequencer;
uart_driver driver;
uart_monitor monitor;
.....
`uvm_component_utils_begin(Agent_In)
`uvm_field_enum(uvm_active_passive_enum,is_active,UVM_ALL_ON)
`uvm_component_utils_end
function new(string name,uvm_component parent);
super.new(name,parent);
endfunction:new
.....
```

```
virtual function void build_phase(uvm_phase phase);
super.build_phase(phase);
monitor=uart_monitor::type_id::create("monitor",this);
if(is_active==UVM_ACTIVE)
begin
driver=uart_driver::type_id::create("driver",this);
sequencer=uart_sequencer::type_id::create("sequencer",this);
\\只有在 UVM_ACTIVE 下，才需要对 Diver 实例化
end
endfunction:build_phase
.....
virtual function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
if (is_active==UVM_ACTIVE)
begin
driver.seq_item_port.connect(sequencer.seq_item_export);
end
this.analysis_port.connect=mon.analysis_port
endfunction:connect_phase
```

在 Agent_In 中，通过 connect_phase，将 Driver 中的 seq_item_port 与 Sequencer 中的 seq_item_export 相连接，从而实现这两个模块的通讯。下面对两个模块的通讯进行详细介绍。

1. Sequencer 和 Driver

Sequencer 将取到的 transaction 有序的传送给 Driver。Sequencer 和 Driver 间是 TLM 级别的通讯，通过 UVM 中的 analysis port 进行通讯。port 和 export 体现的是控制流，而不是数据流。如图 4.4 所示。

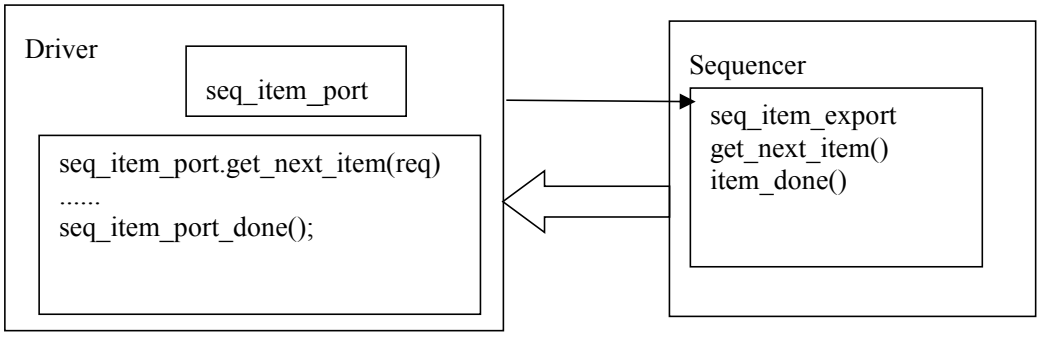

图 4.4　Driver 与 Sequencer 数据流

Driver 和 Sequencer 都是由 uvm_driver 和 uvm_sequencer 派生而来，所以，可以调用 uvm 中定义的各种 port 口来实现。这样避免了使用全局变量，或利用 config 机制等较为复杂的方法。由于是 Driver 通过 seq_item_port 中的 get_next_item()向 Sequencer 发出请求，Sequencer 接收请求后，向 Driver 发送从 Sequence 中获取的数据。当 Driver 接收到数据后通过 seq_item_port_done()传递参数给 item_done 函数，告知 Sequencer 数据已经收到。由此可见，get_next_item()和 item_done()可以协调 Driver 和 Sequencer 间的数据通讯。

Driver 将收到的数据进行解析后，按照 UART 协议的要求发送相应的激励。其部分代码如下：

```
task run_phase(uvm_phase phase); //此部分为 Driver 的主要功能
`uvm_info(tID,"RUNNING:",UVM_MEDIUM)
get_and_driver();
endtask:run_phase
.......
task get_and_driver();   //通过调用 task 实现功能
forever
begin
seq_item_port.get_next_item(req) ;   //从 Sequencer 中获得数据
  ......
if(tran_end==0)     //判断是否需要数据传输
begin
if(right_trans==1)
send_to_dut(req);     //通过此函数发送数据
else
send_error(req);     //若是错误场景，则调用此函数
else
seq_item_port.item_done();
endtask
```

run_phase 实现 Driver 的主要功能处理，通过调用 task get_and_driver()，实现和 Sequencer 的通信以及发送数据。其流程图如下：

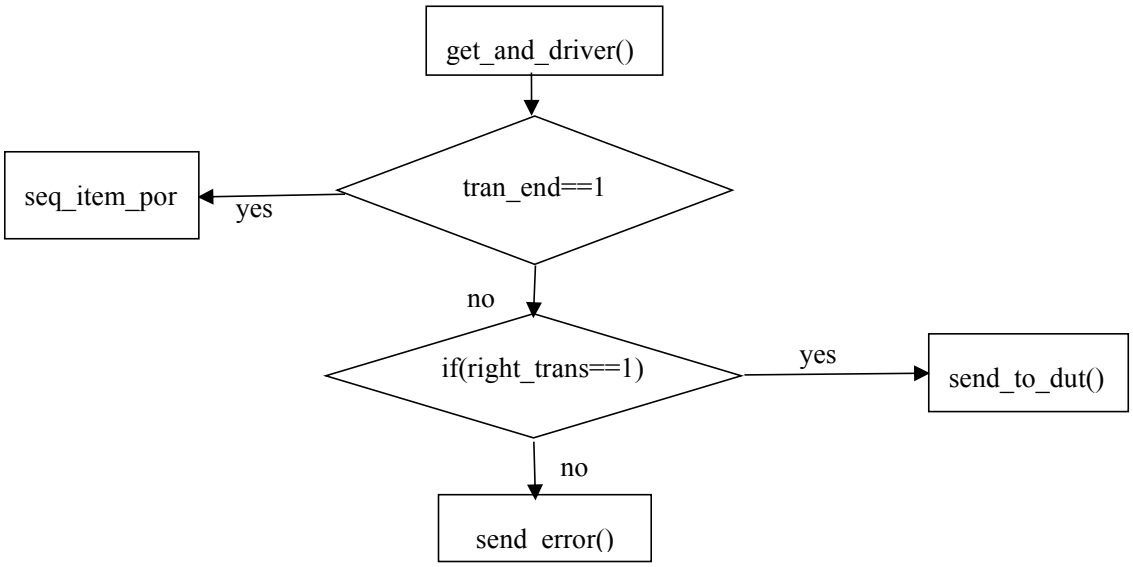

图 4.5　Driver 执行流程图

当上电后，两个 UART 开始通讯。以 UART_VIP 向需要验证的 UART(DUT)发送数据为例。首先发送数据的 UART_VIP 会给接收数据的 UART(DUT)发送一个信号，即 tran_end=0，表示要有数据送达。当 UART(DUT)收到 tran_end=0 这个信号后，会为数据接收操作做准备。当 UART_VIP 将数据全发送完后，tran_end=1，UART(DUT)将不再接收数据。当数据没有发送完毕时，tran_end 为 0，并且 right_trans 为 1，则通过 send_to_dut 这个任务给 UART(DUT)发送激励，否则为错误场景注入，则通过 send_eror()这个函数来处理。send_t o_dut()部分代码如下：

```
task send_to_dut(input transaction_in);
for (int i=0;i<=this.tr.datalength;i++)begin
if(this.cfg.mode!=0)begin
//非模式 0 且为发送状态，需要在数据上加上起始位和结束位。
if(this.cfg.direction==`SEND)begin
this.uart_out <=#1 `STARTBIT;//添加起始位
#((`one_s/this.cfg.baurdrate));
end
else if(this.cfg.direction==`RCV)
begin
@(negedge this.ifc.uart.in);
#((`one_s/this.cfg.baurdrate));
end
this.byte_read_write(this.dataInBuff[i-1],dataOut);
//调用 byte_read_write()函数实现 1byte 数据的发送
```

```
this.tr.dataOutBuff={this.tr.dataOutBuff,dataOut};
bytecnt=0;
if(((i%this.cfg.burstsize)==0)||(i==this.tr.datalength))begin
bytecnt=0;
end
if(this.cfg.mode!=0)
if(this.cfg.mode=2|this.cfg.mode=3) //在模式 2 和模式 3 中需要可编程位
begin
for(int i=0;i<=data_length;i++)
if(this.cfg.direction==`SEND)
begin
bit extrabit,party;
this.parity_calc(this.tr.dataInBuff[i-1],parity);
case(this.cfg.extrabit_type)   //判读模式 2 和模式 3 处于何种状态
  `ADDR_FRAME:exrtabit=1'b1; //多机通讯状态
  `DATA_FRAME:extrabit=1'b0; //数据通讯状态
  `PARITY_FRAME:extrabit=parity;
endcase
this.ifc.uart_out<=#1 extrabit;
#((`one_s/this.cfg.baurdrate));
end
else if(this.cfg.direction==`RCV)begin //接收状态
bit parity,parity_r;
parity=this.ifc.uart_in;
this.parity_calc=(dataOut,parity_r);  //进行数据检查
if(parity!=parity_r)
$display("[Error]：%m DUT sent %h with error parity bit %0t");
end                                          //如果数据有错，给出警告信息
#((`one_s/this.cfg.baurdrate));
end
.....
if(this.cfg.direction==`SEND) begin//开始添加结束位
this.ifc.uart_out=<=#1`STOPBIT;
#((`one_s/this.cfg.baurdrate));
```

```
end
if(this.cfg.direction==`RCV)begin
#1;
if(this.ifc.uart_in !=1'b1) begin
//在接收状态下，如果发现数据起始位为 1，则违反协议
display("[Error]：%m rcv stopbit error @%0t",$time);   //需要给出警告
end
#((`one_s/this.cfg.baurdrate)/2);
end
end
end task
```

send_to_dut 这个任务函数，需要根据 cfg.mode 的不同的状态，对数据进行下一步处理，通过 datalength 来控制发送几个 byte 的数据。当 cfg.direction 是 SEND 状态时，即为发送模式。在非 0 模式下，对需要发送的数据先添加上起始位，然后通过 task byte_read_write()任务函数来获取数据，完成后，需要进一步判断是否是模式 2 或模式 3，在这两种状态下，在数据后还需要添加可编程位，可编程位的值则由模式 2，模式 3 的具体状态决定，若果为多机通讯状态，则该位是 1，数据通讯状态则为 0。若不是这两种状态，则处于模式 1，不需要添加可编程位。最后，对非 0 模式下，需要发送的数据添加结束位，这样，一个完整的发送给 DUT 的数据包就形成了。但是要是 cfg.direction 是 RCV，则处于接收状态，不需要对数据进行加工处理。task byte_read_write()会对单个 byte 的数据进行处理，其部分代码如下：

```
task byte_read_write(bit8 dataIn,output bit8 dataOut);//接收与发送一个 byte 的数据
begin i=0 ;
if(this.cfg.mode==4'd0)      //判断是否为模式 0，如果是，则直接发送数据
begin
this.ifc.mode0_cb.uart_out<=dataIn[0];
repeat(7)begin
@(this.ifc.mode0_cb)
dataout={this.ifc.mode0_cb.uart_in0,dataout[7:1]};
//下降沿时，DUT 发送数据
@(this.ifc.mode0_cb)
this .ifc.mode0_cb.uart_out<=dataIn[i];//上升沿时，DUT 接收数据。
i++
end
```

```
.....
if(this.cfg.mode!=4’d0)//非 0 模式下，接收与发送数据
repeat(7)begin
#1;
this.ifc.uart_out<=dataIn[i];
#((`one_s/this.cfg.baurdrate)/2-1)    dataOut={this.ifc.cb.uart_in0,dataout[7:1]};
#((`one_s/this.cfg.baurdrate)/2);
if(i<7) i++;
else i=0;
endend
endtask
```

在 byte_read_write 函数任务中，通过 cfg.mode 的值来区分如何发送接收数据，在模式 0 下，根据设计文档，在 SCON_REN 是 0 时，发送数据，在 SCON_REN 为 1 时，接收数据，并且波特率是定值；当在非 0 模式下，数据的读与写由其波特率的值来确定，即根据(one_s/this.cfg.baurdrate)/2-1 来确定间隔多久读取或写入数据。

在需要错误场景注入时，可以通过调用 send_error()任务函数来实现，此场景只能用于模式 2 或模式 3，检测是否能检测到有效停止位，即停止位是 1。其部分代码如下：

```
task read_write_error();
bit8 dataOut;
int    bytecent;
Int tr_flag=0;
this.tr.datdOutBuff={};
bytecnt=0;
......
for(int i=1;i<=this.tr.datalength;i++) begin
if(this.cfg.mode !=0) begin
if(this.cfg.mode=2|this.cfg.mode=3)
if(this.cfg.direction==`SEND)begin
this.ifc.uart_out<=`STARTBIT;
#((`one_s/this.cfg.baurdrate));
end
.....
if(this.cfg.direction==`SEND)begin
```

```
this.ifc.uart_out<=~`STOPBIT;//错误注入。
#((`one_s/this.cfg.baurdrate));
....
if(this.ifc.uart_in !=1'b1) begin
$display("[error]: stopbit error!");
end
```

在此任务函数中，通过将 STOPBIT 设置为 0，则可以实现错误注入。添加数据起始位与正常发送数据的方式一致，在添加完成后，调用 byte_read_write 函数，获取数据，只是在添加结束位时，STOPBIT 的值与正常的值相反。

2. Monitor_In

Monitor_In 的作用是对 Interface 上的数据进行监测，并将 Driver 发送给 UART(DUT)的数据复制一份后，最后发送给 UART _VIP。主要的接口定义如以下代码所示：

```
virtural interface uart_if vif;
uart_pkg trans;
......
event e_trans_collected;
uvm_analysis_port #(uart_pkt) to_monitor_out;
//定义 TLM 接口,向 Monitor_Out 进行数据通讯
uvm_analysis_port #(uart_pkt)to_ vip;
//定义 TLM 接口，与 VIP 以及 Coverage 进行数据通讯
virtual function void build_phase(uvm_phase phase);
super.build_phase(phase);
if(!uvm_config_db#(virtual flash_if)::get(this,"","vif",vif))
`uvm_fatal("NOVIF",{"virtual interface must be set for:",get_full_name(),".vif"});
mode_table[0]=9'b000000001;//power down  模式
mode_table[1]=9'b000000010;//sleep  模式
mode_table[2]=9'b000000000;//work 模式
mode_table[3]=9'b111000000;//driver  发送  data
mode_table[4]=9'b110100100;
mode_table[5]=9'b100010100;//DUT  发送  data
endfunction:build_phase
```

整个设计有 power down、sleep、work 三种工作状态，只有在 work 模式下，Monitor_In 才需要进行发送数据处理，当检测到信号是 Driver 发送的，则发送给

UART_VIP；如果是由 DUT 发送出来的，则将信息给 Monitor_Out。通过 Monitor_Out 传送给 Scoreboard，进行比较。

```
task run_phase(uvm_phase phase);
`uvm_info(tID,"RUNNING:",UVM_MEDIUM)
collect_data();    //对数据进行处理
endtask:run_phase
```

在 Monitor_In 这个模块中，最主要的功能在 run_phase 这个任务函数中实现。这个任务会调用 collect_data 函数。当 UART 在工作，且当 UART 在接收数据时，对 Driver 发送的数据进行复制，直接通过 collect_data 这个函数，将数据接收到队列中，至到发送结束，valid 信号为 1，这些数据将通过 TLM 接口由 Monitor_In 发送一份给 UART_VIP。当 UART 处于发送状态时，则将其输出端的数据通过 virtual interface 接收后，发送给 UART_VIP，其部分代码如下：

```
task collect_data (ref my_transaction get_pkt);
logic valid=0;
int data_size;
byte unsigned data_q[$];
byte unsigned data_array[];
  ......
  forever
  begin
  if( vif uart_SE==1&&uart_rec==1);
  begin
  while(valid!=1)
  receive_one_byte(valid,data);
  while(valid)
  begin
  data_q.push_back(data);
  receive_one_byte(valid,data);
  end
  data_size=data_q.size();
  data_array=new[data_size];
  for(int i=0;i<data_size;i++)
  data_array=data_q;
  get_pkt.payload=new[data_size];
```

4.2.5 Agent_Out 模块功能分析

在 Agent_Out 模块中，由于只有 Monitor_Out 这一个模块，所以只需要对 Monitor_Out 进行例化，主要由两个作用。

第一个作用是：在 UART(DUT)向 UART_VIP 发送数据时，Monitor_Out 通过 uvm_analysis_imp 类型的 monitor_in2out 接口，将 UART(DUT)发送的数据传递给 Monitor_In，完成与 Monitor_In 的通讯。然后，通过 Monitor_In 将需要传输的数据进行转换后，发送给 UART_VIP。这样，减少了 virtual interface 上接口的数量，并且充分使用 TLM 接口，使得数据传输较为方便。

第二个作用是：在 UART_VIP 向 UART(DUT)发送数据时，通过 uvm_analysis_port 类型的 dut_out 接口，将 UART(DUT)输出的数据发送给 Scoreboard，用于检验 UART(DUT)功能是否正确。此时，不再需要借助 Monitor_In 向 Driver 传递数据。

4.2.6 Scoreboard 模块功能分析

Scoreboard 的主要作用是比较预期值(UART_VIP 的输出值)与实际值(被验证的 UART 的输出值)，并将结果显示在屏幕上。这主要通过 check 模块来实现。部分代码如下：

```
function checkPkt(packet resPKT,exPKT,int length=0)
int data_error=0;
if(length==0)begin
length=exPKT.size();
end
this.Checks++;
this.AllCheck++;
if ((exPKT.size()==0)||(resPKT.size()==0))begin//接收到空数据
`uvm_info($print("Failed.Empty packet detected"),UVM_MEDUIUM);
`uvm_info($print("The expect pkt is "exPKT,length),UVM_MEDUIUM);
`uvm_info($print("Result Packet",resPKT,length),UVM_MEDUIUM);
 CheckPkt=-1;
 this.ChecksFail ++;
 this.AllCheckFail++;
 end else begin
 for (int i =0;i<length;i++)begin
 if (resPKT[i]!=exPKT[i])begin//比较 DUT 和 VIP 的数值
 data_error++;
```

```
        end
        end
        if (data_error==0)begin//如果 error 的值为 0，则 Pass
      `uvm_info($print("PASS!!!");
       CheckPkt=0;
       end else begin//否则，报告错误信息
       `uvm_fatal($print("Failed!!! Current check has %d
errors",data_error),UVM_MEDUIUM);
      `uvm_info($print("The expect pkt is "exPKT,length),UVM_MEDUIUM)
`uvm_info($print("Result Packet",resPKT,length),UVM_MEDUIUM);
       CheckPkt=-1;
       this.ChecksFail++;
       this.AllCheckFail++;
       end
       end
       endfunction
```

由于 UART_VIP 的输出结果是由运算得到的，因此数据传输速度会较快。所以，在 Scoreboard 模块中，将会率先收到来自 UART_VIP 的数据，这就需要将这些先到的数据先存到数组中。然后，将这些存在数组中的数据与来自 UART(DUT)的输出数据进行比较。如果在比较过程中有数据不同，data_error 会一直为-1，那么，就会通过 uvm_info 等宏单元报告错误信息，并指出错误的数据的值。如果 data_error 为 0，则显示测试通过。

4.2.7 Register Model 模块功能分析

当有错误出现的时候，我们不仅需要知道错误数据，更需要根据这些错误信息推断出设计中可能出现错误的原因。在项目中，我们常常需要通过查看寄存器中的值来推断 DUT 出现错误的原因。这在直接验证可以通过查看寄存器中的值来进行 debug，在 UVM 中可以使用 Register Model 来解决。

如果不使用 Register model，则需要通过全局变量来实现对寄存器的访问，在需要查看的寄存器较多的情况下，这种方法十分不方便，且不利于平台维护，使用 register model 这种机制，可以方便地对寄存器进行配置。此外，在 check_phase 这种不消耗时间的阶段以 BACKDOOR(层次化引用)的方式进行数据读取。

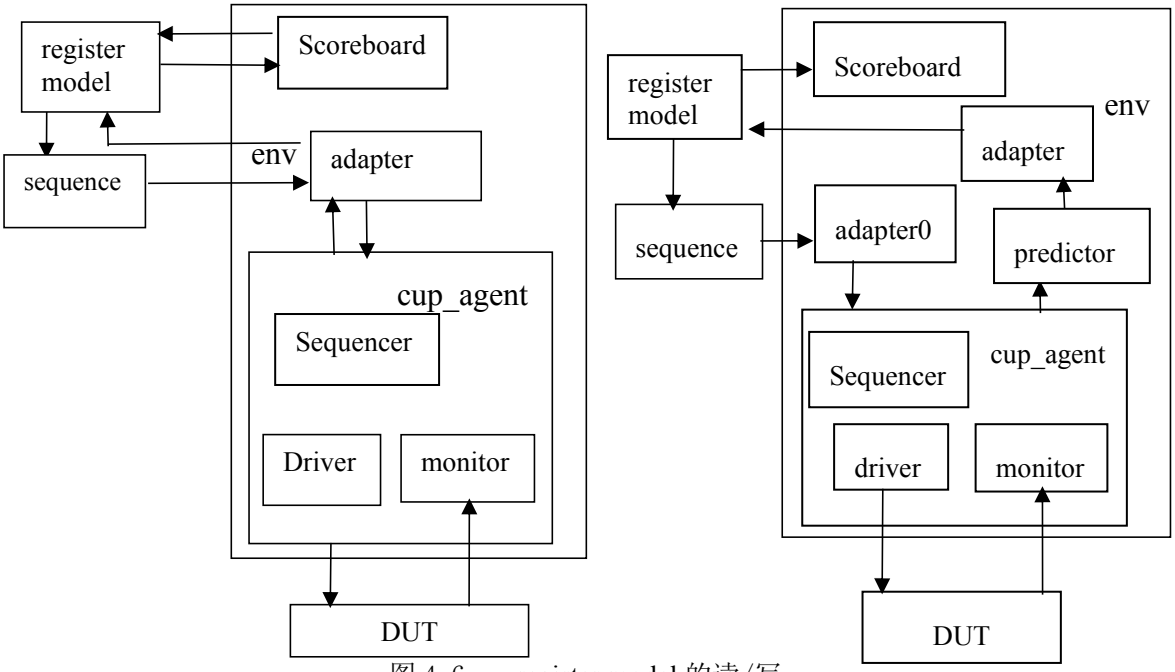

图 4.6　　register model 的读/写

如图所示，左图为 register model 的读操作，右图为写操作，这两种操作都是采用 FRONTDOOR 形式进行的，即通过总线进行读写操作。在进行写操作时，register model 会产生需要进行操作的地址，以及需要些进寄存器的数据。通过 adapter 交给我们的模拟 CPU，即 AHB_VIP 和 APB_VIP。在这个模拟 CPU 内部，由 AHB_VIP 的 sequencer 接收 adapter 的数据后，交给 Driver，最后由 APB_VIP 传递给 DUT，实现写操作。在进行读操作时，总线监测读操作时，将读操作的数据进行封装后发送出去，由 predictor 接收，转交给另一个 adapter，由此传递给 register model。

由于 DUT 中的寄存器的值会实时发生变化，为防止 register model 与 DUT 中的寄存器的值出现不一致的情况，我们可以在 register model 中会定义一个值——mirror value(镜像值)，这样，就可以最大限度的使 register model 与 DUT 中相应的寄存器的值的变化保持一致。通过改变 desired value(渴望值)，我们可以往寄存器中写入我们需要的值，在调用 update 函数，从而实现某种功能。在进行上电操作时，首先，我们需要对验证环境中 DUT 的进行复位操作，以防出现不定态而导致意想不到的错误产生。这也是通过 register model 和 config 机制实现的。以下表中 UART(DUT)的 SCON 控制寄存器为例，进行简单说明：

表 4.2　SCON 寄存器

位	7	6	5	4	3	2	1	0
位名称	FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI
读写属性	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

根据此表格，构建的 register model 如下：

```
scon=uvm_reg_field ::type_id::create("scon");
scon.config(this,8,0,"RW",1,0,1,1,1);
```

在上述代码中，config 的第二个参数是指这个寄存器的宽度，由表可知，此值为 8，需要注意的是，即使在寄存器中，某些位没有使用，这里还是使用寄存器的总位宽。第三个参数是指我们使用的最低位在寄存器中的位置，在 SCON 中，是从第 0 位开始的，因此为 0。第四个参数是指存储的方式。在 UVM 中，为验证提供了多种存储方式，使得在验证时对寄存器的模拟更加方便。因为 SCON 是可读可写的寄存器，因此为 RW。第五个参数表示是否是易失的，一般不常用。第六个参数对寄存器的默认值进行设定。由于 SCON 的初始值为 0，在此，第六个参数为 0。第七个参数一般都设为 1，表示寄存器中使用的这段数值可复位。第八个参数为 1，表示此域可随机化，如果为 0，则一直使用默认值。但是，只有当第四个参数有写的功能时，这个值才有效。最后一位表示此域可单独存储。

将这个定义好的寄存器放在 reg_block 中进行实例化，在一个 reg_block 中，所有的寄存器都有相同的基地址。其部分代码如下：

```
default_map=create_map("default_map",0x400000000,32,UVM_LITTLE_ENDLAN)
; .....
scon=SCON_reg::type_id::create("scon", ,get_full_name());
scon.config(this,null,"scon");
scon.build();
default_map.add_reg(scon,0,"RW");
```

通过 create_map 将 UVM 中的 uvm_reg_block 对应的 uvm_reg_map 实例化，其中的第一个参数是指名字，第二个参数对基地址进行说明，第三个参数是指系统总线的宽度。由于是 AMBA2.0 架构，需要指定大小端。在本项目中，用的是大端。

在对寄存器实例化时，通过 config 函数，指定进行 BACKDOOR 操作时的路径。在 scon.config(this,null,"scon")中，第一个参数 this 用于说明寄存器在 BLOCK 中的指针。第二个参数用于说明 reg file 的指针，由于只有一级 reg file，所以为 null。第三个参数是指出了如何在 DUT 中找到 SCON 寄存器。

通过使用 add，将寄存器加入到 default map 中，否则无法进行 FRONTDOOR 操作。在 default_map.add_reg(scon,0,"RW")中，参数 scon 是对指针进行说明，0 是指寄存器的地址，最后 RW 是指此 SCON 的操作方式为可读可写。

4.2.8 test case 模块功能分析

在验证中，我们需要有足够数量的 case，才能将所有的功能测完。因此，我们需

要编写多个 case 才能将所有的测试情况包含在内。在这些 case 中，我们会根据实际需要对 env 进行实例化，得到我们所需要的场景，从而完成功能测试。

所有的 test 虽然不同，但仍然有相似的部分，我们将这些共同的部分编写到 base_test 中。通过对 base_test 模块中的参数进行修改，可以衍生出不同的 test。这个过程如下图所示。

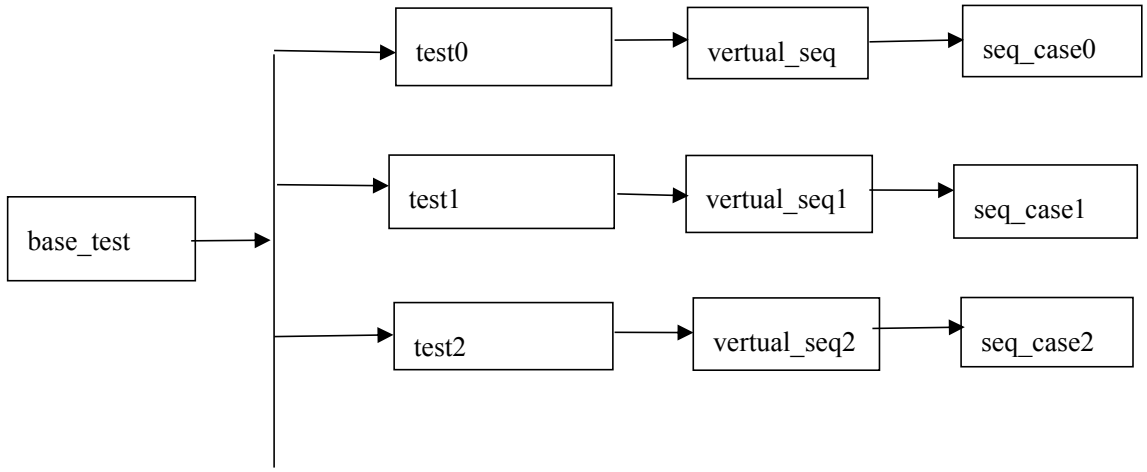

图 4. 7　　test 简图

在 4.2.3 节中，在对 sequence 进行研究时，就提到了 case，这些 case 是具有实际意义的场景，是这节所介绍的 case 的实例化。因此，可以看出，我们需要将这些 case 与 sequence 中的 case 一一对应，使得 AHB_VIP 模块中的 virtual sequence 可以对所有的 Sequence 进行控制，这样就能实现相应的功能测试。

4.2.9　env 模块功能分析

在 UVM 验证中，env 将常用的模块，进行定义，并在 build_phase 中对这些模块进行实例化，由于这些模块被封装到了一起，可以对他们进行统一规划管理。register model 的集成也是在 env 中完成的。在 env 中，通过 reg2ahb_adapter，将 register model 通过 sequence 发送的变量转换成总线可接受的形式。部分代码如下：

```
class uart_test_env extends uvm_env;
  uart_agent    agent_in;
  ahb_virtual_sequencer sequencer;
  reg2ahb_adapter; !@#//对模块进行实例化#$
  virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
    .....
```

```
    uvm_config_db#(svt_ahb_system_configuration)::set(this,"ahb_system_env",this);
    ahb_system_env=svtb_ahb_system_env::type::id::create("ahb_system_env",this);
    //为了使用 UVM 中的众多功能，使用 type::id::create 进行实例化
     ......
    reg2ahb.cfg=cfg; //使用 config 机制
    //不进行覆盖率统计
  uvm_reg::include_coverage("*",UVM_NO_COVERAGE);
  //进行实例化
  regmodel=`REG_MODEL::type_id::create("regmodel");
  //调用 build 函数，实例化寄存器
  regmodel.build();
  //使用 lock 函数，禁止加入新寄存器
   regmodel.lock_model();
   //调用 reset 函数，复位相应的寄存器
   regmodel.reset();
   regmodel.default_map.set_auto_predict(1);
   //将一个转换器和 ahb.sequencer 通过 set 函数告知 regmodel 的 default_map，
   并且使得 default_map 处于自动预测状态。
   uvm_config_db#(uvm_reg_block)::set(this,"*","regmodel",regmodel);
uvm_config_db#(uvm_reg_block)::set(this,"ahb_system_env.master[0].sequencer","regmo
del",regodel);
```

只有使用 connect_phase 将各个有数据通讯的 model 先连接起来，数据才能正常传递。对于 register model，从数据流中可以看出，还需要将 predictor model 和转换器以及总线检测的 model 连接起来。其部分代码如下：

```
  virtual function void connect_phase(uvm_phase)
   ......
   //将 ahb 中的 sequencer 与 agent 的 sequencer 向连接
   sequencer.vip_sequencer=agent_in.sequncer;
   //将 monitor_out 与 driver 向连接
  ......
  agent_out.monitor_out.dut_port.connect(agent_in.driver.monitor_out2driver_export);
  reg_predictor.adapter=cpu_mon_reg_adapter;
  cup_agent.cup_monitor.analysis_port.connect(reg_predictor.bus_in);
  ...
```

end function

本项目中，使用了 AMBA2.0 总线架构，AHB_VIP 起到控制作用。并且使用了两个 sequencer，一个是起到纵向控制作用，向待测 UART(DUT)发送我们预先设定的数据。另一个是用于产生 UART_VIP 向 UART(DUT)发送的数据。ahb_virtual_sequencer 为总的序列发生器控制单元，对整个验证环境中数据的合理流通。

4.2.10 top 模块功能分析

top模块在整个验证平台中处于最顶层，它派生于uvm_top，属于uvm component。在这个模块中，通过使用include和import，将我们所需要的.sv文件加进来，并且通过assign语句将在initial模块中产生我们需要的时钟信号clock信号给APB模块。

在top模块中将待验证的UART进行实例化，并且通过interface功能将Verilog语言编写的待测的UART接进我们的验证环境中。由于Driver和Monitor_Out等模块与DUT有直接的通讯，需要在top中将他们联系起来，这可以通过config机制来实现。在验证中，有时需要某个变量值在仅仅某几个模块中改变，这就可以借助config的半全局性的特点实现。例如，我们只希望在reset_mp模块对top中的env中的sequencer的值进行设定，如果使用全局变量的方法，就会增大出错几率，并且会增加排查难度。本项目中用的top部分代码如下：

```
initial begin
//使用config机制，将各个模块联系起来
uvm_config_db#(svt_apb_vif)::set(uvm_root::get(),
“uvm_test_top.env.apb_system_env”,“vif”,$root.test_top.apb_if);
//将sequencer接到apb总线上
uvm_config_db#(virtual apb_rest_if.ahb_reset_modport)::set(uvm_root::get(),
"uart_test_top.env.sequencer","reset_mp",$root.test_top.apb_reset_if.ahb_rest_if)
//通过interface将UART接入验证环境
uvm_config_db#(virtual uart_if)::set (uvm_root::get(),
"uart_test_top.env.uart_agent","idle_if",$root.test_top.idle_if)
  ……
run_test();//开启验证平台
end
……
endmodule
```

在 top model 中，当执行到 run_test()函数时，验证平台就会被开启，这个函数是一个全局函数，在 top 模块中占据重要作用。

在本项目中，整个验证环境按照 SYNOPSYS 公司的 triger 形式的文件管理，这有利于文件的分类管理以及后期由于测试需求，对环境进行调整，并且，使得在进行后续的相关项目时，具有一定的继承性。仿真工具的调用和 RTL 代码的管理，都是通过使用 MAKEFILE 等脚本文件来实现，这使得测试自动化程度大大提高。当所用测试案例都执行完毕，并且没有错误，覆盖率满足要求时，验证工作就可完成。

4.3 验证工具介绍

在本项目中采用 VCS 进行编译仿真。VCS 是由 SYNOPSYS 公司开发的一款高性能的仿真工具，它不仅支持 System Verilog 验证还支持基于断言的验证。它拥有目前业界范围最广的基于标准的验证 IP 产品组合，还支持 UVM 验证。

在验证时，虽然会生成 log 文件和报告供我们在出错时进行查看，但是在出错时，还是需要借助工具查看波形文件，以及追踪信号进行 debug。在此次项目中，采用了 Verdi 进行 debug。因为 Verdi 为 RTL 和信号之间提供了 trace 的功能，使得在通过追踪信号进行 debug 时变得更加方便。

此外，在使用 Verdi 是还可以通过 File 菜单栏下的 save signal 对常用的信号进行保存操作，过程如下图 5.1 所示。这样，当我们需要再次对这些保存过的信号进行波形分析时，只需要在相同的菜单栏中选择 Restore Signal，选择需要查看的信号即可。

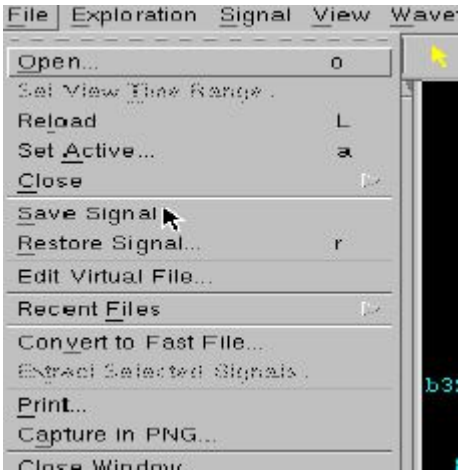

图 4.8　File 菜单栏

当波形信号太多时，为了能更加清晰的查看几个信号的变化情况，可以将他们的颜色着重的标出来。首先，选中需要查看的波形信号，然后打开菜单栏中的 Waveform 这个子菜单，选择其中的 Change Color/Pattern 选项，在弹出如图所示对话框中，只要选择相应的颜色即可。

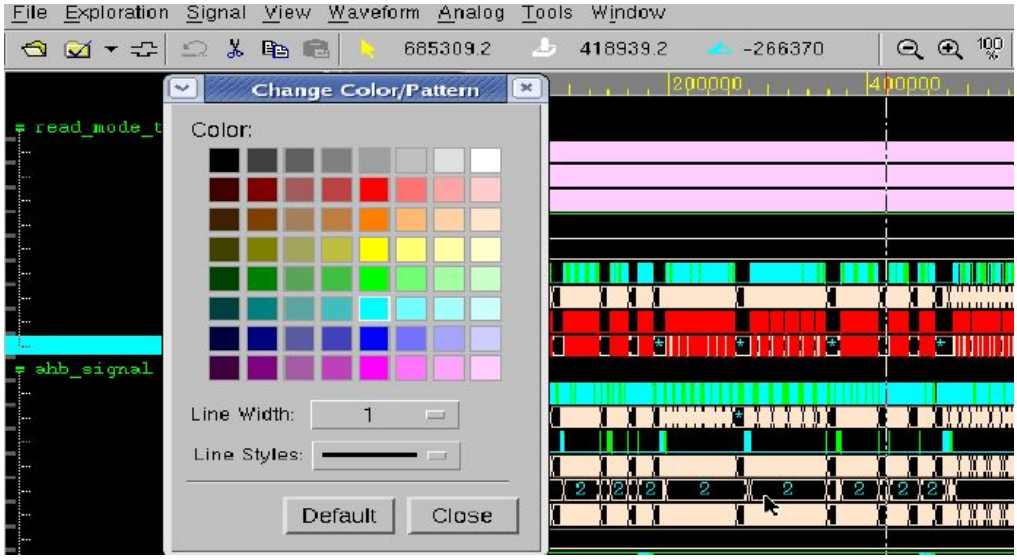

图 4. 9　　Verdi 波形界面

4.4 验证环境简介

在验证环境中，需要对许多文件进行管理，使得验证人员能快速找到相应文件。因此，可以使用由 SYNOPSY 公司的开发的 triger 方式，对所有需要用到的文件进行统一管理。验证人员值需要将相应文件放到相应位置即可，这样，通过脚本文件调用工具进行编译仿真时，就会在相应的目录下生成需要的文件。下图为验证环境的简图介绍。

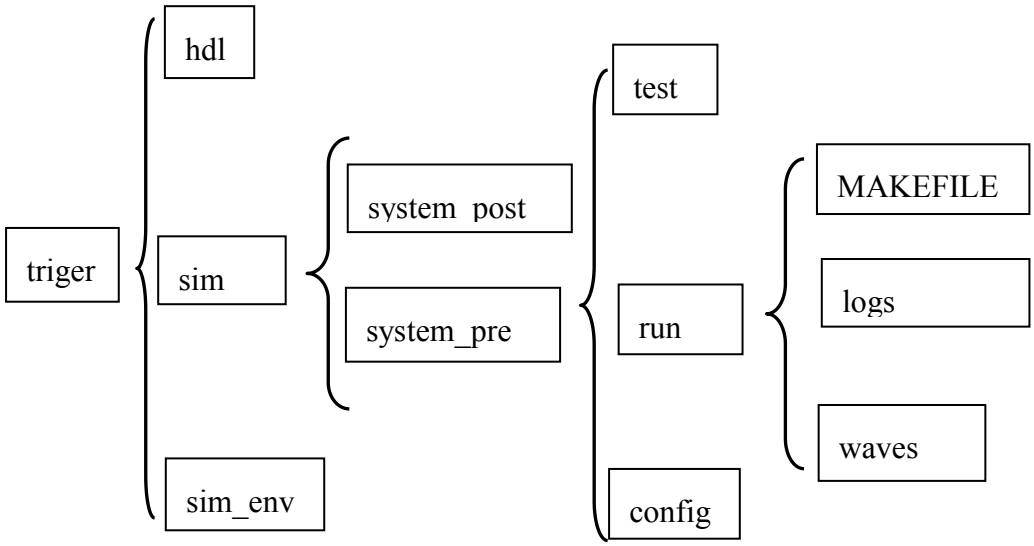

图 4.10　　验证环境简图

在 triger 下的 hdl 中，存放着我们所需要验证的 RTL 文件。

sim_env 中存放着项目环境需要的脚本文件，在进行仿真前，都需要 source 该脚

本，对初始环境进行配置。

在 sim 目录的 s_system_pre 中进行前仿真，在 s_system_post 中进行后仿真。在本论文中，只进行前仿真，因此对后仿真不展开描述。

Sim 目录下的 test 中，存储着搭建 UVM 验证平台所需要的.sv 文件。

在 run 目录下，我们通过编写的 MAKEFILE 来实现验证的自动化，在这个脚本文件中，实现对 RTL 代码的加载，以及工具的调用。当仿真结束后，我们可以在此查看 log 文件、波形文件等，与仿真相关的文件信息。

在 config 目录下，我们将所需要被验证的 RTL 的路径存放于此。这些路径都写在 file.list 中，这样，当有 RTL 文件修改，只要路径不变，我们就不需要对文件重新加载，此外，如果有些文件不需要时，我们可以直接将其路径再 file.list 中注释掉，这样就不会读取该文件。在以.cf 作为后缀的文件中，根据注释中给出的提示，对相应的参数进行配置，决定了是否需要生成波形、coverage report。

如果有波形文件生成，生成的波形文件会被自动存放到存放到 waves 目录下，相应的 log 信息在 log 目录下。波形文件可以通过 Verdi 打开。通过查看 log 目录下的文件，我们可以查看在编译以及仿真过程中出现的错误，方便 debug。

4.5 验证过程

为了调试的需要，可以先不加载设计文件，而只加载测试用例和接口文件，查看是否有错误，如变量名称命名前后不一致，缺失.sv 文件等。此时出现的问题一般为语法上的错误，通过查看 VCS 的 log 信息都可以解决。

但是，有些问题却不能通过 log 文件迅速解决。这时，可以通过在代码中使用 display 功能，这一函数的功能类似于 C 语言中的的 print 函数，可以将可能出错的地方的变量值打印出来，通过对比，发现错误所在。这种方法也可以用于排查某条语句是否执行。在需要测试的语句的前后都使用 display，分别显示不同的内容，通过这可以帮助我们进行 debug 工作。

在 VCS 编译时，可以显示错误行的行号，根据这，我们可以快速找到出错的代码行。当某条语句有错时，可以试着将出现错误的行注释掉，看编译是否正确。如果正确，则注释掉的那行代码有错误，如果还有其他错误，就先查看这些错误。

在确保仿真环境没有错误后，在加入所需要验证的设计文件和激励信号，对验证平台进行更近一步的调试。

在这个阶段首先要确保 DUT 无语法错误，并且 DUT 与验证环境的接口连接是正确的。这个可以通过将单独对环境进行调试时的测试样例再跑一遍来实现。然后，对基本功能进行测试。在我们设计的 UART 中，模式 0 是最简单的收发数据功能，我

们可以先测试模式 0 下功能是否正确。

在此阶段，只能解决以及测试出最简单的错误，更多的错误还需要通过后续大规模测试来排查。

4.6 验证结果

在 UART 的验证中，结果输出分为波形文件和 log 信息，分别存放在 run 目录下的 waves 和 log 文件夹中。当没有 error 出现时，一般可以不查看波形文件，只需要查看下 log 信息即可。当有 error 出现时，首先根据 log 中的提示，推测出现错误的原因，如果是 DUT 设计的问题，需要追踪相应信号进行排查。

通过模式 0 调试完整个验证平台后，开始正式地对 UART 的功能测试。在本次验证中对 UART 的功能测试主要测试点进行如下：

1. 在各个模式下，收发数据正常。

首先，我们需要确保 UART 的工作模式正确。我们可以通过查看两个 startrx_r 或者 starttx_r 间，用于数据打拍的 txtick 信号的个数来确定。因为模式 0 的数据位有 8 位，模式 1 工作模式下的数据位数为 10 位，在模式 2 和 3 工作模式下的数据位数为 11 位。由于模式 2 只能工作在特定波特率，可以通过查看波特率的方式加以区别模式 2 和模式 3。

对于仿真时 UART 的实际的波特率，我们根据波特率定义，可以通过计算 txtick 信号间的差值的倒数，我们可以得知接收数据是的波特率的值。在测试时，有时会出现 error，显示收到的数据与发送出去的数据不一致，此时，就可以通过查看波特率，判断是否是因为波特率不匹配导致的原因。这种情况在直接验证时出现过。由于需要测试的测试点很多，使用直接验证，这些激励需要验证人员一个个用 C 语言写进 case，在添加到效应目录下，施加给待测设计，并对 VIP 进行相应的模式设定，这就可能出现文件添加错误的错误情况，并且测试效率低。

下图是在模式 1 下，UART 接收数据时的波形图，txtick 信号间的差值是 108160ns，通过计算 txtick 信号间的差值的倒数 1/108160 得到值是 9245Bd/s，接近 9600Bd/s，结果与期望一致。在查看两个 startrx_r 信号间的 txtick 信号的个数为 10，位数符合模式 1 工作模式下的数据位数。

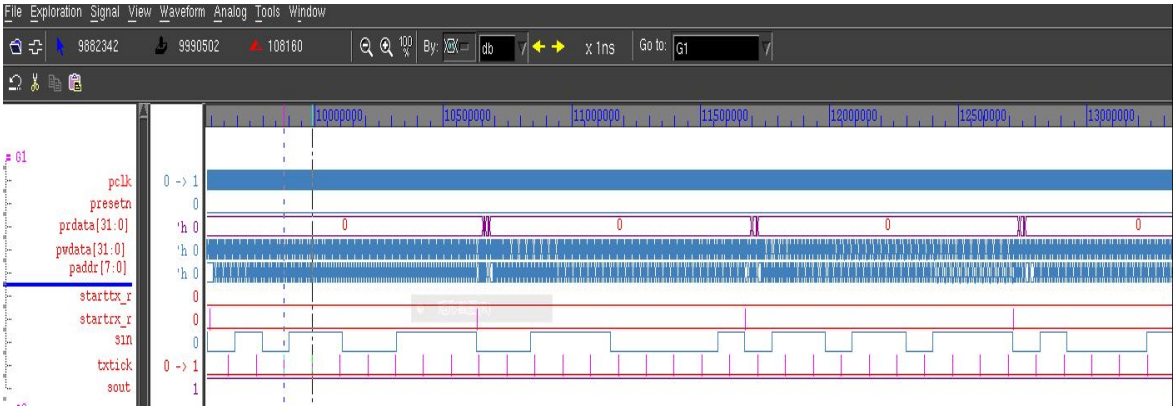

图 4.11　模式 1 接收数据

首先 UART_VIP 会向需要被验证的 UART 发送数据，仿真得到的波形图如下：

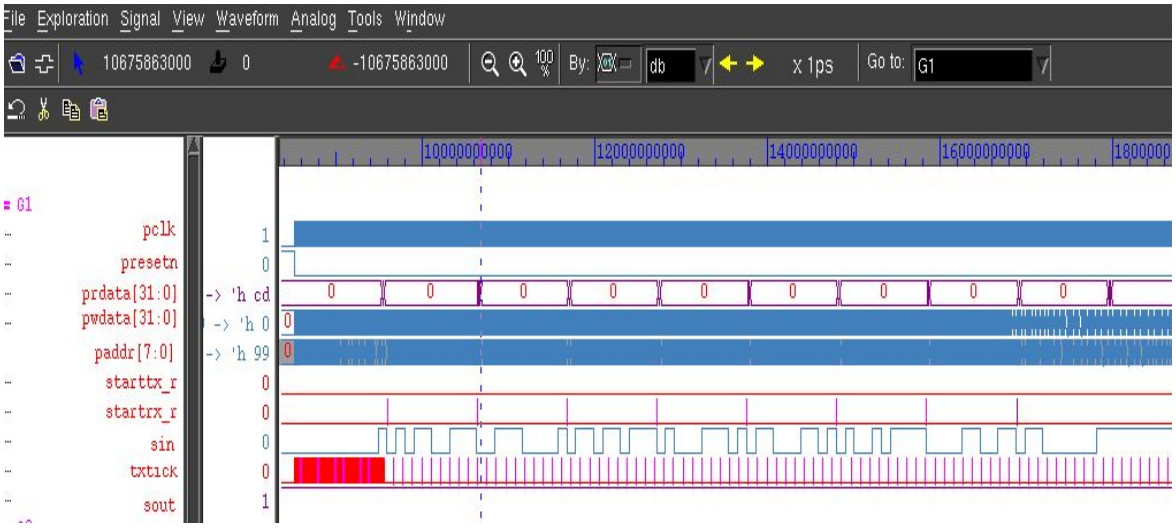

图 4.12　UART 接收数据

根据波形图，pclk 是 UART 实际工作时的时钟信号，presten 为复位信号。为了防止因为不定态而造成的仿真结果不正确，需要先对 UART 进行复位操作。根据 AMBA2.0 协议中，APB 总线的相关信号定义，prdata 信号为 APB 总线向总线上的设备传输的数据内容。通过 prdata 信号，我们可以看出 UART 第一个接收到的数据为 cd。starttx_r 信号是指开始发送数据，startrx_r 信号是指开始接收数据，这两个信号都是高电平有效。查看波形图，表示接收数据的 startrx_r 呈现出高电平，表示发送状态的 starttx_r 呈现出低电平，状态正确。此时根据第一个 startrx_t 与第二个 startrx_t 间的 txtick 信号对应的 sin 信号值，可得值为 0101100111。其中第一位 0 为数据起始位，最后一位 1 为数据的结束位，10110011 为收到的实际内容。由于采用的是 AMBA2.0 架构，并且采用大端的存储方式，即采用高位低字节，低位高字节的存储方式，此外，根据 UART 协议的规定，数据读取时低位先进，高位后进的原则，则

实际的内容为 1100 1101 即 cd。从而证明 UART 读取的数据正确，与报告一致。

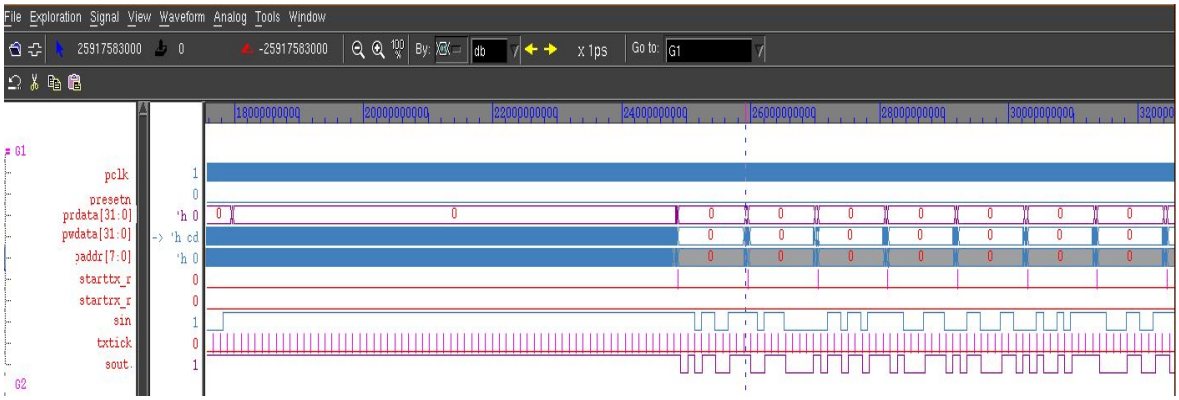

图 4.13　UART 发送数据

根据 AMBA2.0 协议可知，pwdata 信号为 APB 总线上的设备向总线发送的数据内容，由图 5.6 可得，此时 UART 向 APB 总线发送出去的第一个数据是 cd，与接收到的数据一致。此时，表示 UART 接收数据的 startrx_r 信号处于低电平，即无数据接收，而表示发送数据的 starttx_r 信号有高电平出现，信号的状态都与预计的一样，没有异常。此时根据第一个和第二个 starttx_r 信号间的 txtick 读取 sout 信号，值为 0101100111，值与 sin 信号相同，说明 UART 的发送功能正确，与报告内容一致。

2. 测试 UART 的错误检测功能。

此功能用于检测在数据传递结束时，最后一位的 stop 位是的状态是否正确即是否为 1。当结束位数值不正确时，将 framerr_r 置 1，同时不再接收数据。此错误数据会进行丢弃，不会发送出去，以免造成不必要损失，同时会联合其他设备给出警告。这项功能在非 0 模式下可使用。以模式 3 工作模式下的错误检测为例进行分析说明。

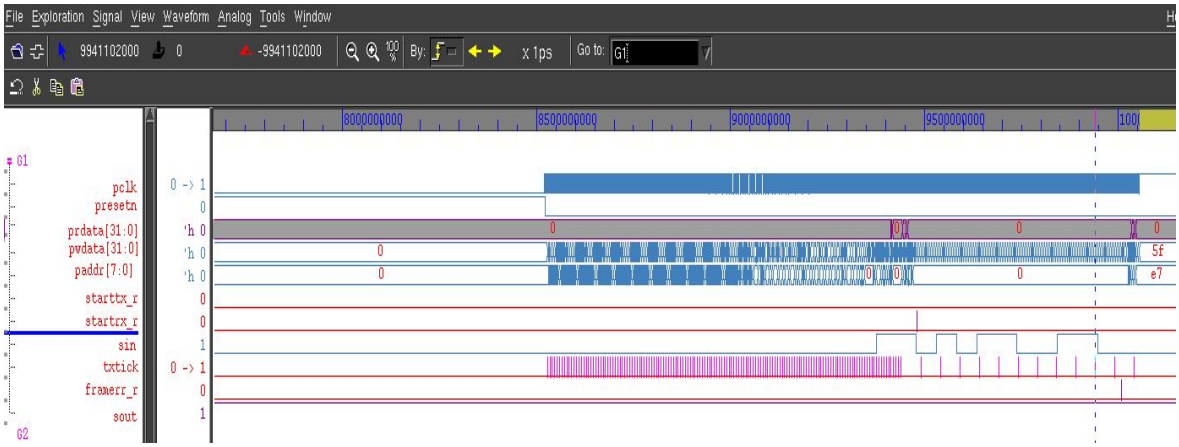

图 4.14　模式 3 下错误检测

根据 UART 协议，在模式 3 工作模式下，数据的帧格式为 11 位，其中，最后一

位表示的是数据的结束位。通过 startrx_r 信号，我们能判断出接收到的数据的起始位，再由用于数据打拍的 txtick 信号，读取 sin 信号对应的 11 位数据为 01011001100。此时数据的最后一位结束位为 0，最后第二位是可编程位，数值为 0，表示处于数据传输状态。由此，可以清楚的看到，在数据接收结束时，结束位为低电平，不是有效的数据结束位。此时 framerr_r 信号变为高电平，表示数据有错误。并且在此之后，没有 startrx_r 信号出现，表明不再接收数据。由此可得，这项功能正确，与报告内容一致。

3. 多地址通讯功能的测试。

在模式 2、模式 3 工作模式下，UART 才具有这项功能。多机通讯可以实现一个主机与多台从机间的通讯。以模式 2 工作模式下工作频率为系统时钟的 32 分频为例，进行分析。

对于此功能的验证，我们用 VIP 作为主机，待测的 UART 作为从机，进行多机通讯。在下图中，我们通过读取两个 txtick 信号，并做倒数运算，可以得出此时 UART 的工作频率，由此可得工作频率匹配。在多机通讯模式下，数据字节的可编程位是逻辑 1。通过 startrx_r 信号和 txtick 信号可以找到相应的数据位，进行工作模式判断。

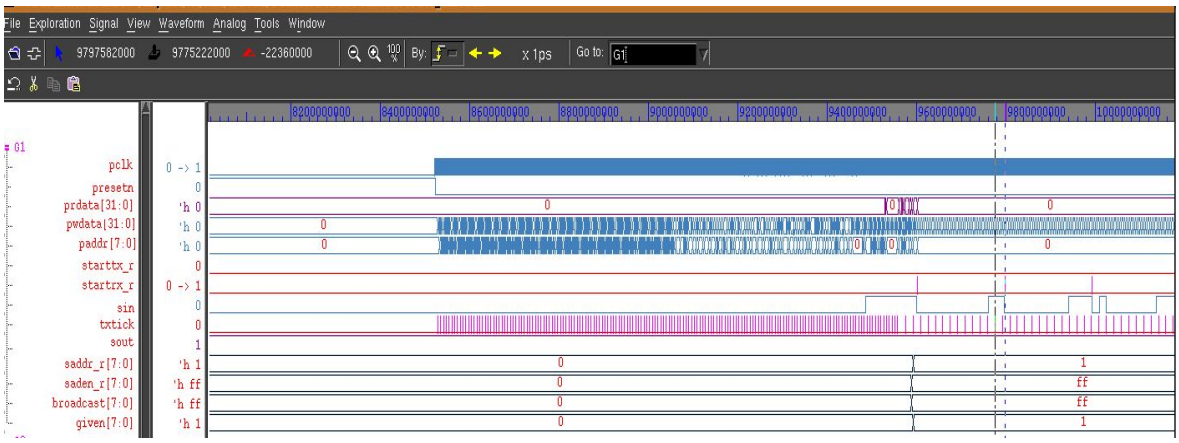

图 4.15　模式 2 下多地址通讯

由图可得，读取第一个和第二个 startx_r 信号间的 txtick 信号对应的 sin 信号，可得接收到的数据为 0 0000 0000 11。可编程位为 1，符合 UART 协议要求。从机的地址是由从机地址寄存器 saddr 和从机掩码寄存器 saden 共同决定的。其中，对于不同从机，saddr 寄存器中的值不同。由图可得，被验证的 UART 的指定地址即 saddr 的值，是 1。由于只对一个设备进行通讯，saden 的值必须是 0hff。将 saddr 和 saden 的值做逻辑或运算，得到广播地址，因此应该为 0hff。由图可得，此时 broadcast 中的值正确。此时 VIP 发送的用于的寻址信号 given 信号中的值为 1，此值表示向地址为 1 的从机进行寻址。因此功能正确，与报告内容一致。

4.7 覆盖率分析

在验证时，需要对源代码中的每条语句的执行状况，以及根据设计要求的功能被验证状况进行评估。在此次验证中，采用的是 VCS 仿真器，这款仿真器可以提供覆盖率收集的功能，并且将覆盖率报告以网页的形式保存。

进行验证评估时，一般需要保证设计中的每条代码都被执行过，才算基本完成仿真任务。但是，可能会出现无论怎样添加激励，代码中的某些语句始终无法被执行的状况。这时，就需要和设计人员进行协商，确保没有被覆盖的部分不是因为激励不够所造成的。如果是由于激励缺失所造成的，就必须根据要求添加新的激励，直至所有代码都被执行。在测试过程中，经常会出现覆盖率达到一定程度后就无法提高的情况，这时，也可以通过和设计人员讨论，对激励进行修改，提高覆盖率。

但是，即使代码覆盖率满足要求，还是不能保证所有的功能都已经被测到。因此，还需要保证功能覆盖率达到 95%以上，才能确保大多数功能已经在测试中被执行过。这样，也有助于发现潜在的 DUT 的 debug，确保 DUT 的功能正确性，提高流片成功率，使得验证具有完备性和准确性。

本次的覆盖率报告主要分为以下几个部分:

SCOR：这项是指声明覆盖率，即赋值语句执行了多少次。在一般情况下如果有哪项赋值语句没有被执行，则需要增加激励，进行仿真。

CCND:条件覆盖率。特指在基本的条件如(a>b)中以及较复杂的组合条件例如(a>b) or (b>c)中，有多少种条件组合被测试到。

TOGGLE:翻转覆盖率，用于检测设计中是否存在因信号的变化而执行某种电路。

FSM:状态机覆盖率，此选项是为检测设计中哪些状态未发生变化，以防止潜在的设计隐患。

BRANCH:分支覆盖率，特指 if 语句和 case 语句是否将所有分支都执行了。100%的语句覆盖率并不能保证 100%的分支覆盖率，因为可能一个 if 语句并没有 else 的分支。我们在 100%的语句覆盖率的基础上，进一步规定了 100%的分支语句覆盖率的指标。目的就是确保类似于上述的分支语句在缺省条件下也进行过测试。

PATH:实际所执行到的代码路径所占有可能路径的百分比。在系统级测试中，由于此项参数测试需要耗费很多时间，一般不用。

ASSERT:断言覆盖率。

由于验证时考虑了后期系统级验证，在待测试模块中有些代码与 UART 的功能无关，只与后期其它模块的地址分配有关以及系统工作模式切换相关，这导致最终的代码覆盖率中，为检测设计中哪些状态未发生变化以防止潜在隐患的状态机覆盖率(FSM)值为 65.53%，指出所有分支是否执行的路径覆盖率(PATH)值为 77.38%，以及

用于检测设计中是否存在因信号的变化而执行某种电路的触发覆盖率(TOCGLE)79.36%，都比较低。但是功能覆盖率已经达到 100%。经设计人员确认，这些未执行的代码都与待测的 UART 无关，再将这些代码去除后，代码覆盖率可达 100%。由此可见结果是合理，不需要再调整激励，以及验证平台设计的正确性。总体的代码覆盖率结果如下图 5.8 所示：

图 4.16　代码覆盖率

4.8 本章小结

本章在研究完待测设计的协议以及整个设计的构架要求后，根据验证计划书，搭建符合后期需要的验证环境。同时，对数据流向以及各个组成模块进行详细的分析研究，确保验证的合理性和正确性。

之后，对 UVM 验证的过程和结果进行描述，对各个测试点的内容进行解析，在所有测试都完成后，对仿真结果进行覆盖率分析。由于 UART 与其它模块有交互，所以覆盖率没有达到 100%，但是功能覆盖率达到 100%，这是合理的，这反应了 UVM 验证的正确性。

第五章 总结与展望

随着电子产品的不断更新换代，各个公司对验证不断提出新要求，都希望能缩短验证周期的同时提高产品良率。这对验证工作提出巨大挑战，能否提高代码复用率，使得验证周期缩短，同时提高测试准确性，成为验证中的重要问题。

传统的直接验证虽然验证起来十分便捷，但是代码复用率很低，不利于验证平台的维护和重复使用，而且对于超大规模的设计不适用。因此，追求更高抽象级别的验证将成为验证的趋势。

本文以华虹公司的一款低功耗芯片中的 UART 部件为依托，首先介绍数字集成电路验证的背景，然后从 UVM 验证平台的基本组成，UVM 验证平台的接口，UVM 验证平台提供的几种标准单元库，UVM 验证提供的几种常用机制入手，对 UVM 验证方法学进行概述，并最终与 system Verilog 验证进行比较，突显 UVM 验证的优越性。接着，对此次验证所用的验证平台架构，数据流向，各个组成模块功能进行详细描述。

本项目中在搭建平台时虽然出现各种 bug，但最终都得到了解决，总体而言还是比较顺利的，从验证过程中发现的 bug 可以看出 UVM 验证的优越性。本人在此次项目中所做的工作如下：

（1）学习 UART 的基本知识和 AMBA2.0 总线协议，深入理解 UART 的功能。

（2）学习 System Verilog 和 UVM，查阅相应资料，学习公司其它 IP 的验证。从中了解 UVM 验证。

（3）对验证环境进行规划。根据 UART 的功能，进行详细的验证计划设计，并与设计人员进行协商，确保验证的完整性。

（4）编写测试所需要的模块，并分析结果是否正确，确保符合要求。

由于本人在数字集成电路验证领域的经验不足，在 UVM 验证平台搭建过程中还存在一些不足，有待于优化，可以优化的方面为：

（1）由于时间有限，本次只进行了 IP 的验证，并没有实施系统级验证，还需要在进行系统级验证时尽可能地利用已有的代码进行研究。

（2）编写脚本的能力有待提高。使用脚本不仅可以提高验证的自动化程度，还可以避免由于人为因此造成的失误。如何使用脚本，还需要研究。

（3）在进行实例化引用时，代码编写的不够精简，需要继续学习。

参考文献

[1] G.Moore. VLSI:Some Fundamental Changes[J]. IEEE SPectrum,1979,(6.4):79-87.

[2] H.Lekatsas. Arithmetic coding for low Power embedded system design[J]. Data Compression Conference ,2000:430-439.

[3] Zhiyuan Ren. Hierarchical adaptive dynamic Power management[J]. Design, Automation and Test in Europe Conference and Exhibition, 2004, (1.1):136-141.

[4] R.S.Mireea. Low-Power encoding for global communication[J]. CMOS VLSI. IEEE Transactions on Very Large Seale Integration (VLSI)Systems,1997, (5.4):444-455.

[5] M.Alidina. Pre-computation-based Sequential logic optimization for low power[J]. IEEE Trans. On VLSI Systems,1994, (2.4):426-436.

[6] 阮圣彰.Synopsys Flow for Low power Design[M] .Department of Electronic Engineering National Taiwan University of Science and Technology, 2004.

[7] 钟涛. CMOS 集成电路的功耗优化和低功耗设计技术[M]. 微电子学，2000:106-112.

[8] M. Pedram. Power minimization in IC design: Principles and applications[J]. ACM Trans. on Design Automation of Electronic Systems,1996:3-56.

[9] Chae-seok. Design Power & Power Compiler[R]. CAP Lab Technical Report.Souel National Univ, 1998.

[10] M.Berkelaar and J.Jess. Gate sizing in MOS digital circuits with linear programming[J]. Proceedings of European Design Automation Conferenee,1990:217-221.

[11] C.H.Tan and J.Allen. Minimization of Power in VLSI Cireuits Using Transistor Siring[P], Input reordering and Statistical Power Estimation. Proe.of Int, Workshop on Low Power Design, CA, 1994: 75-80.

[12] H.R.Lin and T.T.Hwang. Power reduction by gate sizing with Path oriented slack calculation[J]. IEEE Trans Computer Aided Design of Integrated Circuits and Systems, 1999. (18.2):231-238.

[13] W.Yeh. Optimum halo structure for sub-0.1um CMOSFETS[J]. IEEE Trans. Electron Oct, 2001. (48.10):2357-2362.

[14] S.Mutoh, I-V power supply high-speed digital circuit technology with multi-threshold voltage[J]. CMOS. IEEE J.Solid-State Circuits, 1995.(30.8):847-854.

[15] M.C.Johnson. Leakage control with efficient use of Transistor stacks in single threshold CMOS[J]. IEEE Transactions on Very Large Scale Integration(VLSI)Systems, 2002,(10.1): l-5.

[16] S.Lee and T.Sakurai. Run-time voltage hopping for low-power real-time systems[J]. Design Automation Conference, 2000.Proceedings 2000.37th. June 5-9,2000:806-809.

[17] N.H.E. West, K. Eshraghian. Principles of CMOS VLSI Design: A System Perspective[M], 2th Edition, Addison-Wesley Publishing Co, New Work, 1993.

[18] 夏代川. SOI 技术及其应用[M].半导体光电,1993:319-326.

[19] LIN Cheng-lu. New progress in SOI teehnology[M]. Semiconductor Technology,2003.

[20] H.B.Bakoglu. Circuits, Inter connections and Packaging for VLSI[M]. Addison-Wesley Publishing ComPany, Ine.1990.

[21] 王传声.多芯片组件(MCM)的封装技术[J].微电子技术,2000,28(4):40-45.

[22] Frank Emnett. Power Reduction Through RTL Clock Gating[M]. Biegel Automotive Integrated Electronies Corporation, SNUG San Jose,2000:5-6.

[23] 温淑鸿,崔慧娟,唐昆.有效利用片上分块存储器[M].清华大学学报:自然科学版, Joumal of Tsinghua University, 2006,(46 .1).

[24] R.FSproull, I.E.Sutherland, C.E.Molnar. The counter flow pipeline processor arehitecture[J]. IEEE Design & Test Computers. Summer,1994,(11):48-59.

[25] Mircea R.Stan, Member. Bus-Invert Coding for Low-Power I/O[J]. IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION(VLSI)SYSTEMS, Mar, 1995.(3.1).

[26] Tom'as Lang, Enric Musoll, Jordi Cortadella. Extension of the Working-Zone-Encoding Method to Reduce the Energy on the Micro Proccssor DataBus[C]. Computer Design:VLSI in Computers and Processors, 1998, ICCD'98. Proceedings. International Conference Oct 5-7, 1998:414-419.

[27] 孙海厢,邵志标.基于自适应重排的低功耗地址总线编码[J].西安交通大学学报, 2006,(4).

[28] Chi-Ming Tsai, Guang-Wan Liao, Rung-Bin Lin. A low power-delay Product page-based Address bus coding method[J]. VLSI Design, 2003. Proeeedings.16th International Conference. , 2003, (4.8) : 521-526.

[29] T.Lang, E. Musoll, J.Cortadella. Extension of the working-zone-encoding method to reduce the energy on the microprocessor data bus[C]. Computer Design: VLSI in Computers and Processors, 1998. ICCD'98. Proceedings. International Conference,,1998.(5.7):414-419.

[30] L.Benini and P.Siegel. System-level optimization[J]. Proceeding of International Symposium on and Design Monterey. ACM Press .Power estimation and Low Power Electronics. 1998:173-178.

[31] Himanshu Bhamagar. Advanced ASIC Chip Synthesis using Synopsys Design Compiler Physical Compiler and Prime Time[M]. Kluwer Academic Publishers, 2002.

[32] Prime Power Manual[R]. Release U • 2003.06-QA. June2003.

致谢

在此论文完成之际，我要深深地感谢西安电子科技大学微电子学院对我的培养，为我提供了良好的学习环境，感谢在校期间教育我的各位老师，没有他们的辛勤劳动就没有我扎实的专业知识，为我踏上社会实习前打下良好基础，感谢微电子学院能给我出去实习的机会，让我能够学以致用，使理论与实践向结合，为正式踏入社会工作做准备。

由衷地感谢我的导师吴振宇副教授对我的辛勤培养和指导以及严格要求。吴老师严谨地学术作风，深深地感染了我，让我在今后的工作和学习中受益终身。在论文修改期间，吴老师不辞辛劳地给予我种种宝贵意见，使得论文最终能顺利完成。在此，向吴振宇老师致以深深地谢意。

感谢华虹集成电路有限责任公司为我提供的实践平台，感谢我的企业导师朱思良高级工程师，让我学会如何在项目出现困难时应该如何面对，这对我今后的工作受益匪浅。感谢在一起工作过的同事，在我实习期间给予我学习以及生活中的帮助。在大家的帮助下，我的学习水平和工作能力有了很大地提高，让我的任务顺利完成。

感谢寝室的王亚妮，樊周华，何静博同学给予我学习和生活上的帮助。感谢一起实习的李东起，马丹，邵旭东，高志峰，张俊磊同学，在实习期间给予我的帮助和鼓励。感谢王春珊学姐和黄旭学长，杨昕煜学长在工作和论文撰写上给予我的帮助。

感谢我的父母二十多年来对我的无微不至的关爱和照顾，让我顺利完成学业。在此，对他们表达衷心的祝福和感谢。

作者简介

1. 基本情况

夏晓芸，女，安徽，1991 年 1 月出生，西安电子科技大学微电子工程学院软件工程（集成电路设计方向）领域 2013 级硕士研究生。

2. 教育背景

2013.08～ 西安电子科技大学，硕士研究生，专业：软件工程（集成电路设计方向）

3. 攻读硕士学位期间的研究成果

参与科研项目及获奖

[1] 项目名称 TPCM，起止时间 2014.08~2015.03，项目进行中，作者贡献负责安全类芯片 SHHIC4805 的数字电路前端验证工作。