中国科学院大学

University of Chinese Academy of Sciences

硕士学位论文

基于 UVM 的通用 DMA 验证平台设计与研究

作者姓名：许海洋

指导教师：张　浩　副研究员　中国科学院微电子研究所

张书迁　高级工程师 北京昂瑞微电子技术有限公司

学位类别：工程硕士

学科专业：集成电路工程

培养单位：中国科学院大学微电子学院

2021 年 6 月

Design and research of universal DMA verification platform based on UVM

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Engineering

in Integrated Circuit Engineering

By

Haiyang Xu

Supervisor: Professor Hao Zhang

School of Microelectronics

University of Chinese Academy of Sciences

June 2021

中国科学院大学
研究生学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：许海洋

日　　期：2021年4月20日

中国科学院大学
学位论文授权使用声明

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：许海洋　　导师签名：

日　　期：2021年4月20日　　日　　期：2021年4月20日

摘 要

随着集成电路集成规模和复杂度不断提高，芯片制造工艺不断提升，验证工作也愈发具有挑战性，难度也越来越大。所以，提高验证效率也就成为当今亟待解决的问题。针对传统验证方法可重用性差，验证效率低等问题，已不再适用于搭建具有复杂结构和功能的DMA模块的验证平台，借鉴当今主流的验证方法学，基于UVM验证方法学搭建了一个可重用性强的验证平台。本论文的研究工作和创新点如下：

1. 针对传统验证平台可重用性差和定性激励等缺陷，设计了可重用的验证组件和随机激励。通过设计可重用的验证组件并将相关组件封装在一起形成一个验证IP，与原有平台相比，缩短了验证平台开发周期。通过生成随机激励，验证某一功能时可以产生此范围内的测试激励，减少编写定向激励所用时间，提高验证效率。
2. 针对验证平台与 DMA 内部寄存器交互复杂和定位错误速度慢等缺陷，设计了寄存器模型和断言分析机制。通过使用寄存器模型，使用前门和后门访问，可以对 DMA 的寄存器进行读写操作，方便快捷。通过使用断言分析，可以减少人为检查时间，快速定位设计错误点，使对验证结果的检查更加自动化。
3. 针对传统数据比对方法需要较高精度的参考模型或者花费仿真时间长的缺陷，设计了使用 config_db 直接取出数据进行对比的方法。通过使用config_db 机制实时的将从设备中 DMA 搬移的数据取出后传递给数据对比模块进行比对，这样不仅能保证数据正确性，避免编写复杂的参考模型，而且还能节约仿真时间，提高仿真效率。

本论文使用可重用的验证组件、随机激励、寄存器模型、断言分析机制和config_db 的数据比对方法完成了该 DMA 模块的验证。实验表明代码覆盖率达到99.8%，功能覆盖率达到100%，符合该DMA的验证需求。

关键词：DMA，UVM，可重用，AHB，功能覆盖率

Abstract

With the scale and complexity of integrated circuits continue to increase, and the chip manufacturing process continues to improve, the work of verification has become increasingly challenging and difficult. Therefore, improving the verification efficiency has become an urgent problem to be solved today. In view of the poor reusability and low efficiency of traditional verification methods, it is no longer suitable for building a verification platform for DMA modules with complex structures and functions. Drawing lessons from today's mainstream verification methodology, a reusable verification methodology is built based on UVM verification methodology. The research work and innovations of this thesis are as follows:

1. Aiming at the defects of poor reusability and qualitative incentives of traditional verification platforms, reusable verification components and random incentives are designed. By designing reusable verification components and encapsulating related components together to form a verification IP, compared with the original platform, the verification platform development cycle is shortened. By generating random stimulus, a test stimulus within this range can be generated when verifying a certain function, which reduces the time used to write a directional stimulus and improves the verification efficiency.
2. In view of the complex interaction between the verification platform and the internal registers of the DMA and the slow positioning error, a register model and an assertion analysis mechanism are designed. By using the register model and using the front door and back door to access, the DMA registers can be read and written, which is convenient and quick. By using assertion analysis, human inspection time can be reduced, design error points can be quickly located, and the inspection of verification results can be more automated.

3. Aiming at the defect that the traditional data comparison method requires a high-precision reference model or takes a long time for simulation, a method of using config_db to directly fetch the data for comparison is designed. By using the config_db mechanism, the data moved by DMA from the device is taken out in real time and transferred to the data comparison module for comparison. This not only ensures the correctness of the data, avoids writing complex reference models, but also saves simulation time and improves simulation efficiency.

This thesis uses reusable verification components, random excitations, register models, assertion analysis mechanism and config_db data comparison method to complete the verification of the DMA module. Experiments show that the code coverage rate reaches 99.8%, and the function coverage rate reaches 100%, which meets the verification requirements of the DMA.

Key Words: DMA, UVM, Reusability, AHB，Functional coverage

目 录

图目录

表目录

第 1 章 绪论

1.1 研究背景与意义

随着集成电路集成规模的快速发展[1]，当今大规模集成电路的验证需求也越来越大。随着芯片产品迭代次数不断加大，产品研发周期不断缩减，验证所扮演的角色也就更加重要了[2]。因此，在一款芯片进行流片之前，如何快速有效的发现并修复所遇到的设计问题，这是作为一个验证工程师所必备的技能。

SOC(System On Chip)设计[3]的主要方式是使用可复用性强的知识产权IP(Intellectual Property)通过总线连接组成一个系统。所以，当今的 SOC 系统相当于不同 IP 核的集成[4]，当数据和操作通过总线传输到各个 IP 核后，可以观测并检验系统中不同部分之间的信息交互情况，所以，这就需要构建验证平台来进行芯片验证。

UVM (Universal Verification Methodology) [5]是当今最受欢迎的验证方法学，来自于 OVM(Open Verification Methodology)，并且还加入了 VMM(Verification Methodology Manual)的寄存器模型 RAL(Register Layer) ，成为了 IEEE 的标准[6]。使用 UVM 构建的验证平台不仅能将测试激励与待测模块分隔开，使二者关联性降低，有利于验证平台可重用，而且还可以生成随机激励，使验证更加完备。如图 1.1 所示，是使用 UVM 验证方法学搭建的验证平台，使用虚接口 interface 连接验证平台与被测设计，收集被测模块的输入和输出数据，并做相应处理。

针对传统验证平台可重用性差、验证平台与 DMA (Direct Memory Access) 内部寄存器交互复杂、定位错误速度慢、传统数据比对方法需要较高精度的参考模型或者花费仿真时间长和定性激励等缺陷，使用可重用的验证组件、随机激励、寄存器模型、断言分析机制和 config_db 的数据比对的方法完成了 DMA 模块的验证，不仅可以减少验证周期，而且还可以实现较强的可重用。

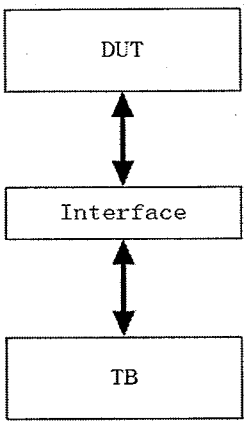

图 1.1 UVM 验证平台

Figure 1.1 UVM Verification platform

1.2 国内外研究现状

随着集成电路产业的快速发展[7]，制造工艺水平的极大进步，已经实现了从微米，亚微米到深亚微米的巨大成功，现今摩尔定律已经快不适应当今的IC(Integrated Circuit)设计制造产业。但是，在消费电子领域，除了手机和电脑等高新设备芯片需要较高的工艺去实现更低功耗和更高性能的芯片，其他一些设备芯片基于成本与需求等原因对工艺要求并不是很严格。国内大力发展芯片产业，不仅从教育上培养优秀的集成电路学生，而且扶植相关芯片公司，从资金和技术上给与支持，补充国内在芯片设计制造领域的相关短板，提高自给自足的能力。因为芯片设计与芯片验证息息相关，可以说二者相辅相成。一款性能良好的芯片在流片之前，要经过严格的验证才能保证芯片不出大的错误，一次流片就能成功的概率在当今大规模集成电路设计中很小，所以，芯片验证工作在芯片设计阶段占据越来越重要的地位。国内芯片设计企业起步比较晚，相关的验证技术和验证平台积累较少，近几年出现了新的验证方法学，用其搭建的验证平台的高效和实用性正在被广大芯片设计公司所采用。

验证平台的好坏决定着验证效率，也就是决定着研发产品的周期。所以，使用哪种语言来搭建验证平台就显得很关键。传统的验证平台是使用 Verilog[8]编写的，后来又相继推出了 e 语言，Vera，SystemC[9]以及 SystemVerilog[10]等验证语

言。当前最受欢迎的是SystemVerilog语言，在2005年被评为IEEE标准，已经成为各个芯片设计公司的首选语言，兼容Verilog语言，在其基础上，添加了很多属性和方法，使其更加适合验证平台的编写。

芯片验证技术的发展，除了有用于编写验证平台的验证语言外，还相应的研发出了用于验证的验证方法学。验证方法学是对一系列验证技术和验证方法的总结和探索，一开始验证方法学的发展是由EDA厂商们推出来的，因为最初它们设计自己的前后端开发工具，也提供一些专用的定制IP核,为了更加方便使用和验证这些IP核，它们会提供一套用于验证这些IP核的验证IP。但是这种VIP(Verification Intellectual Property)只能在自家的仿真工具上使用，并不被其他家的仿真工具所兼容，这就造成了各自资源不能共享，限制了验证方法学的发展。在SystemVerilog语言发布后，2005年，国外三大EDA厂商之一Synopsys推出了VMM[11](Verification Methodology Manual)，这是最早的使用SystemVerilog语言搭建的验证环境，相对来说架构比较简单，但是为设计通用验证平台打下了坚实的基础，被广大芯片设计公司所采用的，用于搭建IP级和系统级验证平台。接下来，在2008年，美企Cadence和Mentor推出了OVM (Open Verification Methodlogy)验证方法学，加强了搭建的验证平台可重用性，但是也只是自家的仿真工具支持。在2011年，UVM验证方法学被推出，由SystemVerilog语言编写的，现在已经变成当前主流的验证方法学。它还有很多机制，如sequence机制，objection机制，config_db机制和phase机制等，把一个大任务分解成一个个小任务，便于处理。所以,使用UVM可以节省验证人员极大地搭建验证平台的时间，使得验证更加完备，缩短项目交工日期，从而能让产品更快更好的投入生产，提高了产品的竞争力。随着知名度和使用率的提高，UVM验证方法学今后也会得到更大范围的推广，推动整个验证方法的发展。

目前在国内，一些大学机构和相关企业也针对UVM验证方法学搭建的验证平台和方法申请了相关专利，关于芯片验证领域的论文也越来越多，例如西安电子科技大学做过APB-I2C等验证平台[12],合肥工业大学做过I2C、UART和AXI[13]等验证平台。在国外，也有相当大一部分公司和机构对UVM验证方法学进行深耕和探索。例如三大EDA公司在给出一些设计的IP核有验证环境。Jeff Vance

等人在 uvm harness 的基础上，提出了一套较为新颖的解决方案[14]。而对于 DMA 验证平台的研究，国防科技大学做过一款专用 DMA 的验证平台设计[15]。

可以看出，针对 UVM 验证方法学，国内外相关科研机构和企业都进行了相关研究，并统一成标准，是今后芯片验证方面的一个不可被忽视的趋势。但是，当前芯片验证技术还存在着不少的挑战，也会花费更多的时间来进行验证。如何快速而有效的验证一个复杂的设计已经是当今验证工程师亟待解决的一个关键问题，所以制定一个有效的、完备的和可靠的验证计划和平台是解决这一问题的一个很好的方法。为了完成芯片验证任务工作，实现芯片验证的快速收敛，我们可以设计一个有效的、完备的、可重用的和可靠的方案来进行芯片验证。而如何衡量验证平台的完备性和可重用性问题，我们可以采用覆盖率驱动的验证方法，通过设置覆盖点和覆盖组，使用仿真软件自动收集覆盖率信息，这样可以检测被测设计的功能点是否被覆盖到，这需要一个完整且可靠的验证计划。UVM 使用封装的模块组件分层分级的搭建验证平台，优化了验证平台的整体架构，使整个验证平台具较强的可重用性，可以被今后相似的工程所使用。

1.3 论文研究内容

本论文通过分析当前主流的验证方法学，结合 DMA 结构和功能特点，提出了采用 UVM 来构建验证环境，最终实现新方法的创新和验证平台的可重用的目标。根据 DMA 功能点设计验证计划，设计了可重用的验证组件并将其封装成验证 IP，使用新的数据比对方法提高验证效率，使用随机激励、寄存器模型和断言分析来完成 DMA 的验证。本论文的主要研究内容如下：

（1）根据传统验证平台可重用性差和定性激励等缺陷，设计了可重用的验证组件和随机激励。通过设计可重用的验证组件并将相关组件封装在一起形成一个验证 IP，与原有平台相比，缩短了验证平台开发周期。通过生成随机激励，验证某一功能时可以产生此范围内的测试激励，减少编写定向激励所用时间，提高验证效率。

(2)根据验证平台与DMA内部寄存器交互复杂和定位错误速度慢等缺陷，设计了寄存器模型和断言分析机制。通过使用寄存器模型，使用前门和后门访问，

可以对 DMA 的寄存器进行读写操作，方便快捷。通过使用断言分析，可以减少人为检查时间，快速定位设计错误点，使对验证结果的检查更加自动化。

（3）根据传统数据比对方法需要较高精度的参考模型或者花费仿真时间长的缺陷，设计了使用 config_db 直接取出数据进行对比的方法。通过使用 config_db 机制实时的将从设备中 DMA 搬移的数据取出后传递给数据对比模块进行比对，这样不仅能保证数据正确性，避免编写复杂的参考模型，而且还能节约仿真时间，提高仿真效率。

1.4 论文章节安排

本论文共有六章，每章内容如下：

第一章是绪论，介绍芯片验证工作的背景、意义与国内外研究现状。

第二章介绍了验证策略与 UVM 验证方法学。首先选用了灰盒验证策略，接着就介绍了 UVM 验证方法学，包括其组件、运行机制和整体架构等方面

第三章介绍了 AHB 总线协议与通用 DMA 模块的构成和功能，详细描述待测模块的功能，并提出了验证策略与计划。

第四章介绍了验证平台的整体架构，包括其内部组成的组件、连接与运行方式和数据处理方法，并设计新的数据比对方法，提高了验证效率。

第五章介绍了验证平台的测试流程和所使用的的测试 case，最后对测试结果进行分析，以功能覆盖率为驱动，从覆盖率覆盖程度说明验证的完备性。

第六章是总结与展望。首先对目前工作及其不足进行分析，最后对今后工作进行了规划。

第 2 章 验证策略与 UVM 方法学

随着数字芯片集成度越来越高，对单个 IP 核或者 SOC 系统进行验证也来越来越复杂，对于不同的 IP 核要搭建相应的验证平台却不能重用这种方法在当今大规模集成电路设计中已不可取了。针对这种情况，经过国内外大量的科研机构和企业的努力，总结和收集了大量科研工作者的需求和经验，相应的验证语言和验证方法学应运而出，已经逐步发展出一系列成熟的验证方法。本章就是分析当前各个公司和科研机构使用的验证方法学，提出验证策略并对本论文所采用的主流的 UVM 验证方法学进行介绍说明。

2.1 验证策略

通常，设计师们设计出来的产品都需要经过验证测试才能通过产品认证，最终推向市场。所以，不论是做软件还是做硬件，在产品上线之前，都需要经过严格的测试，才能保证其质量和安全。我们常用的验证方法分为三种，依次分别为黑盒验证、白盒验证和灰盒验证。对于硬件芯片验证来说，有形式验证、硬件仿真验证和 FPGA 验证这三种技术。对于 UVM 验证方法学而言，其支持随机激励生成、覆盖率收集和断言分析等方法，是当今最受欢迎的验证方法学。

对于一款芯片来说，最重要的就是实现所需要的功能，也就是完成项目需求书上的要求，验证被测设计 DUT 是否能像预期一样正常工作，所以，我们采用的是基于功能覆盖和代码覆盖相结合验证方法。这样不仅能验证所设计芯片功能，而且还能找出设计代码中的冗余部分，减少芯片面积和降低功耗等。在实际验证工程中，会遇到很多问题，即使你输入了正确的激励，但是可能因为内部时序与延迟等一部分原因，本来应该暴露的错误会被隐藏起来，从而造成设计的失误，导致流片失败。

黑盒验证不需要了解芯片的内部设计结构，只需要知道芯片接口信号，然后按输入接口信号的协议从外部施加激励，观察输出结果。可以用黑盒验证对模块级或系统级设计进行验证，还可以从不同的抽象层次对被测设计与参考模型的输出进行比对，然后给出比对结果。但是是否采用黑盒验证，还是要做一些考虑的。

比如说被测设计本身很复杂且自身的时序可能有多个周期，所以在做参考模型时很难做的非常精确，这就造成无法检测到所有的设计错误，也无法百分百模拟待测设计。因此，对于一个芯片设计，是否采用黑盒验证也需要考虑设计的细节，通过认真讨论并详细测算使用黑盒验证达到的性能和效率与使用其他方法所具有的差距，然后再确定使用哪种方法。因为这种方法是在不知道芯片内部结构的情况下使用，所以很难用于调试，而且，对于测试一些功能点的细节也不容易观测与验证。

白盒验证跟黑盒验证有很大的不同，它可以通过设置断言和一些信号监测模块来进行验证，省去了搭建参考模型的时间和精力。但是这种方法需要知道设计细节，清楚设计结构，一般是设计工程师自己验证，但是对于验证工程师来说，他并不了解具体模块的内部结构，也不知道设计的具体细节，所以无法写出信号监测模块和断言分析，所以白盒验证在这里就有了极大地局限性。而且，如果是一些定制 IP, 设计模块设计好之后需要卖给客户，客户如果需要自己去验证下该 IP 模块，因为只知道功能而不知道设计结构，就更不能采用白盒验证了。

灰盒验证是结合了白盒验证和黑盒验证的特点而形成的一个方法，既拥有参考模型，又在内部结构中加入了断言分析和一些监控信号模块，不需要太精确的参考模型，因此对于一些需要重点关注的信号，可以使用断言分析来进行监测。灰盒验证同时吸收了黑盒验证与白盒验证的精华和特点，完美结合二者的优点，成为当今验证工程师的首选，此外，在调错和改错方面，也比黑盒验证更加方便，提高了验证效率。

2.2 验证方法

通常，对于数字芯片设计流程来说，一般要经过芯片功能要求、芯片详细设计、RTL 编码、功能仿真、电路综合、形式验证、布局布线、功能和时序检查、流片、封装和测试等过程，如图 2.1 所示。在其中，功能仿真、形式验证和 FPGA 验证为芯片流片前必须要经过的的验证手段。

形式验证是通过逻辑抽象的方法对两个设计进行等价性比较，或者使用数学公式的方法来推算硬件的逻辑性，其特点就是无需仿真，只比较逻辑一致性，不

需要检查时序，不需要编写一些测试激励来对设计进行验证。和其他验证方法相比较来说，形式验证不消耗仿真时间，不需要测试向量作为输入，不浪费大量机器内存，节省了很多开销。这些优点可以使形式验证这个方法更全面的验证设计的特性，其中等价性检查这种方法要保证所验证的两个电路在同样的初始条件下如果是相同的输入要得到相同的输出，一般是两个设计的表达形式，用于比较两种实现方式是否一致，被广泛应用到各种设计中。属性检查主要是检查设计模块的某个属性的特定时序逻辑，也就是检查待测模块的属性。在一定的时钟周期内，设计模块的属性可以被量化成一定的逻辑电路，通过分析等价表达式的期望来判断设计是否正确，来评估表达式的正确性，最后判断是否符合所想要映射的硬件电路。可以使用 SystemVerilog 语言来定义断言，利用仿真工具来检查设计的属性。定理证明在形式化验证中占据很重要的地位，它是通过严谨的数学证明，使用数学推理和公式来比较 RTL 描述的设计结构和系统的形式化描述的输入二者是否一致，需要一些算法的支持。

FPGA 验证是数字芯片设计流程中必不可少的重要一个环节，在设计的芯片流片之前，一定要进行 FPGA 验证，这样可以检测被测设计在实际被生产出后，在实际电路上测试应用程序的模拟结果，与流片后的芯片接上外围电路进行测试所使用的环境接近。FPGA 验证就是把设计映射到可配置的硬件单元上，通过软件烧录进入一些测试程序来测试设计是否符合最终流片出来的产品，可以说是最接近实际产品的一次测试。其基本思想就是使用可以进行配置的验证平台运行测试程序来仿真被测模块，看其是否符合预期的功能和特性，可以通过一些检测设施来进行数据收集与比对步骤，最后观察比对的结果来判断待测设计是否合格。

功能仿真是在整个芯片设计流程中最关键的一个环节，一般由仿真工具完成，仿真工具则是由 EDA 厂商提供。具体步骤就是针对待测设计，给出测试激励，进行时序检查，一般在一个或多个时钟周期后，查看设计输出与现实的结果是否匹配，如果需要观测某个信号或者特定部分的细节结果，需要编写相应的监测模块查看具体结果，这个可以通过仿真工具调出波形显示的查看某一时刻的信号值，也可以通过打印信号值在仿真窗口和文本中体现，在如今发展先进的仿真工具可以很轻松的做到这一点。在仿真验证过程中，首先是要制定好验证计划，才能明

确知道自己要测试芯片的哪些功能点，当然，很多人都希望能设计一次就能成功，但是这是理想目标，所以要编写好验证计划非常关键，比如如何分配资源、提取测试点和保证关键功能不出问题等关键原因。在编写好验证计划后，在验证环境中通过产生数据激励并将激励驱动到待测设计的接口信号上，通过验证平台收集输入输出的数据信息，进行比对。在此期间，面对不同的初始值，根据不同的配置，可能相同的输入会产生不同的结果这些在搭建验证平台时都需要考虑到并设计清楚，所以，仿真验证会花费整个设计流程中绝大部分时间，所以，如何设计高效且可重用的验证平台对整个验证项目起着关键的一步。

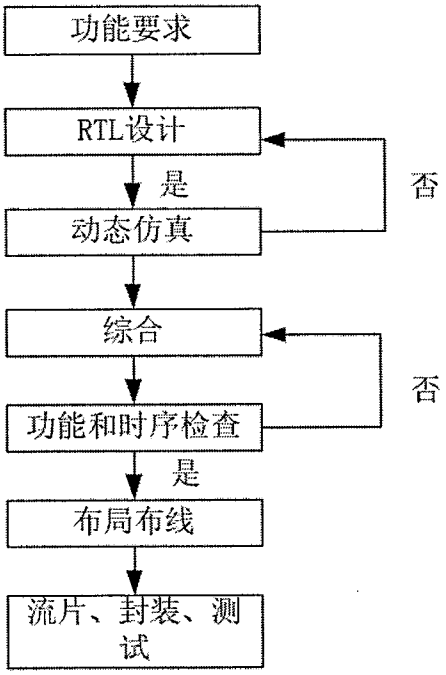

图 2.1 芯片设计流程

Figure 2.1 Chip design process

2.3 UVM 验证方法学

2.3.1 UVM 验证方法学概述

验证的目的是为了通过一些方法来查找出设计中出现的错误，并交给设计工程师来进行改正设计模块，以达到预期的功能。所以，如果要实现上述要求，就需要搭建一个验证平台来实现对待测设计的验证工作。UVM 验证方法学有很多

用于搭建验证平台的组件，如 uvm_driver，uvm_monitor，uvm_sequencer，uvm_test 和一些容器类的组件 uvm_agent，uvm_env 等。有了这些组件，可以设计自己的验证平台，像堆积木似的分层次设计。而且，UVM 还有很多强大的机制，例如 sequence 机制[16]，它使得激励的产生，发射和驱动分隔开，相当于把一个大任务分成一个个小的任务，方便管理。对于一个验证平台来说，其要有以下基本功能：首先要能模拟待测设计，在不同的初始条件下要表现出和待测设计一样的功能，也就是我们俗称的参考模型，在与待测模块有一样的输入激励时，具有和待测模块一样的输出；其次，验证平台要能判断待测设计与参考模型在相同初始条件下的输出是否一致，这就需要一个模块来比对二者输出的结果，这个模块在 UVM 验证方法学中有一个名字，叫做计分板，具体如何比对待测设计与参考模型的数据，这就需要有相应的判断标准，这个就需要根据具体情况来进行设计；最后，不论是输入还是输出数据，都是一组相关的数据，这样就需要有专门的模块去把这些数据收集起来，按照相应的数据传输协议格式，形成一个数据包通过端口传输给相应的其他模块，这就是 monitor 模块。

使用 UVM 验证方法学搭建的验证平台具有很多优点，因为它有已经开发好的库文件，而且这些库文件也可以进行扩充，具有很强的延展性。使用 UVM 搭建的验证平台在当今主流的 EDA 仿真工具上都可以使用，避免了同一个的验证平台开发者使用不同的 EDA 仿真工具会有不通用的情况，极大地减少了因为工具而不是设计带来的问题。使用 UVM 自带的库文件搭建的验证平台具有以下优势：第一，验证组件都是继承同一个类，而这个类则封装了很多用于操作数据流的函数，所以从其派生的类都具有这些操作，验证工程师可以直接调用而不用自己重新编写，极大地优化了验证平台的结构，减少了冗余代码的编写；第二，整个验证平台具有树形结构，可以从一个根节点出发，使用相关组件搭建具有层次分明的验证平台，便于日后维护；第三，验证平台把待测设计与验证平台用虚接口分隔开，降低了验证平台与待测模块的关联性，使其可用于相似的设计，简单的验证平台如图 2.2 所示。

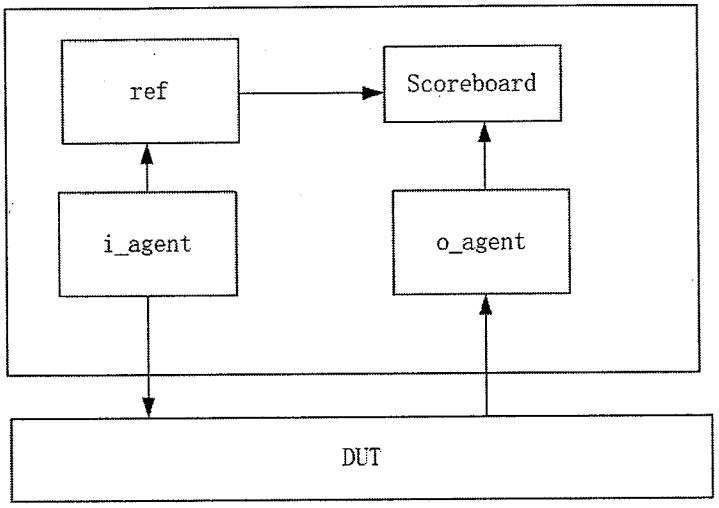

图 2.2 简单 UVM 验证平台

Figure 2.2 Simple UVM verification platform

2.3.2 基本组件与架构

UVM 验证方法学的基本组件主要有两类，分别为 uvm_object 和 uvm_component[22]这两种。但是仔细分析来说其实所有的类 uvm_component 和类 uvm_transaction 都来自于类 uvm_object，也就是说几乎所有的类都来源于类 uvm_object。类 uvm_object 是一个最基本的类，也就是说它具有很高的抽象等级，而且它内部封装了一些函数操作，比如 create、copy、compare、print 和 sprint 等等，方便了以后一些数据处理。从它派生出的类既保留了它原始的特性，又加入了各自的特点，成为了一个新的类，这种方法在面向对象语言中经常可以看到。整个验证平台的层次化结构是由这些组件构成的，最后组成一个树形结构展示给大家，它是 UVM 树的根节点，所有组成 UVM 树的组件都派生自类 uvm_component, 各个组件之间的关系很明白，层次清晰。而类 uvm_transaction 则与类 uvm_component 特性正好相反，它是一个组成事务的类，其性质在仿真过程中是动态的存在的，也就是说其作用只在一段时间内有效，当过了限定时间，它就会失去作用。由类 uvm_transaction 派生出来的类有类 uvm_sequence_item、类 uvm_sequence_base 和类 uvm_sequence#（REQ,RSP）等。

在 UVM 中，uvm_top 为整个树形结构的根节点，它只有唯一一个，是一个全局变量，由根节点会分化出子节点，就像一棵树一样生根发芽。组成树形结构

的组件在实例化时都会有两个参数，一个是 parent 变量，其实现方式是在验证平台内部生成一张查找表，记录着各个组件之间的关系，表示其由谁例化而来，这样便于后期维护和更新；另外一个是 name 变量，也就是要给这个 component 类型的组件实例化一个名字。

UVM 验证平台的各个组件主要有以下几种，如图 2.3 所示[17]，具体内容如下：

激励驱动器 uvm_driver：激励驱动器 driver 的功能是将相应的激励数据驱动到总线上，因为根据 UVM 验证方法学的知识，验证平台与待测模块是分格开的，所以，driver 是将激励信号驱动到虚接口上，在这个过程中，它完成了将 sequence 数据信息到待测模块能接受的数据格式的转换，其内一般有一组读写函数或任务。

监视器 uvm_monitor：监视器 monitor 的作用是收集总线上的数据，并把数据发送给需要的模块，比如说参考模型和计分板等。它与激励驱动器 driver 的功能正好相反，它不是发送数据，而是按照相应的数据格式将数据收集起来，所以，在验证平台中也是一个必不可少的一个子组件。

激励发射器 uvm_sequencer：激励发射器 uvm_sequencer 的功能主要是将 driver 要求的 sequence 发送给 driver，然后由激励驱动器 driver 将按照待测模块的接口协议驱动到总线上，它相当于是一个仲裁，不同的 sequences 有不同的优先级，然后按照不同的算法，将相应的 sequence 发送出去。

代理 uvm_agent：代理 uvm_agent 类可以让验证工程师定义一个代理，它体内主要封装了 driver，monitor 和 sequencer 这三个子组件。当它被设计好后，我们可以使用 is_active 这个选项来决定是否激活它，一般都把激励驱动器 driver 和激励发射器 sequencer 放到一块，这样的话如果是作为输入代理，则这三个组件同时使能，可是如果是输出代理，那么，激励驱动器 driver 和激励发射器 sequencer 就派不上用场了，可以选择不使能这两个组件，那么里面只剩监视器 monitor 这个子组件了，其作用就是收集待测模块的输出信息，并将数据发送给计分板。

计分板 scoreboard：计分板的功能就是比对数据，它的数据来源主要是来自两个地方，其一是来自待测模块的输出，经过监视器 monitor 收集到的数据传送给计分板 scoreboard，其二的数据来源是参考模型的输出数据，当计分板收集到

二者的数据后，然后在其内部进行数据比对，最后给出比对结果，输出到终端，最后打印到仿真文本中经过脚本进行处理。所以，计分板模块 scoreboard 相对来说设计并不复杂，但是要处理好何时进行数据比对处理。

参考模型 ref_model：参考模型也是 UVM 验证平台所不可或缺的一个重要部分，它的作用是模拟待测设计的行为，并与待测设计的行为保持一致，这在整个验证过程中非常重要。通常我们的验证方法选择灰盒验证，而不是采用不用验证模型的白盒验证和采用精确的参考模型的黑盒验证，这样不仅可以降低验证模型的复杂度，而且还可以使用监测模块对需要重点关注的信号信息进行收集与处理。在 UVM 验证方法学中，参考模型一般是用 SystemVerilog 编写的，也可以采用 c 语言编写的验证模型，通过调用从外部模块传输数据，这种方法在当今主流的验证语言中是被支持的。

寄存器模型 reg_model：无论在哪个设计模块中，其内部都不可避免的有一些寄存器，当这些寄存器被写入不同的值时，会相应的执行不同的功能，会得到不同的输出和处理。但是由于对于大型设计而言，其内部寄存器太多，为了方便管理和维护，寄存器模型 reg_model 就应运而出了，反映的是构成寄存器模型的函数和数据。寄存器模型在整个验证平台中具有很重要的地位，因为它可以模拟待测设计内部的寄存器行为特征，通过前后访问或者后门访问来待测模块的寄存器进行访问，按照验证计划的要求，寄存器模型都可以很好的完后才能任务。如果没有寄存器模型，就不会有后门访问操作了，极大地扩充了访问方式，不然只能依靠前门访问方式对寄存器进行操作。

激励产生器 uvm_sequence：激励产生器 sequence 是由激励序列 sequence_item 组成的，它的主要作用就是当激励发射器 sequencer 需要数据时，将其内的激励序列由激励发射器 sequencer 发送给激励驱动器 driver。而激励序列 sequence_item 可以组成一组事务，这组事务内封装了许多信息，比如说 ID 版本号等，根据具体要传输的信息来确定其内具有哪些数据。一组 transaction 是通过 TLM 端口进行传输的，它相当于 TLM 端口识别的数据格式，在其内也封装了一些数据成员变量和函数任务，当实例化后，可以很方便的使用这些函数和任务。

验证环境 env：验证环境 env 相当于是容器类，里面封装了有关验证平台的

的各个组件。

验证顶层 base_test:也是一个容器类，是比验证环境层次更高的一个验证平台顶层，里面可以包含多个验证环境 env.

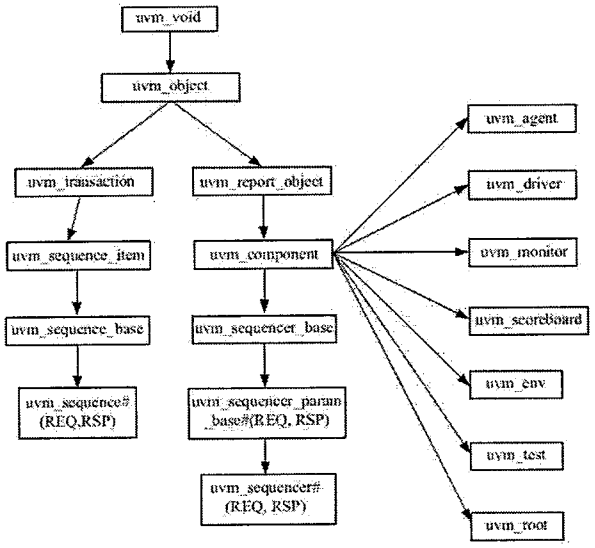

图 2.3 UVM 验证方法学组件

Figure 2.3 UVM verification methodology components

2.3.3 UVM 机制

UVM 验证方法学作为当今主流的验证方法学，深受广大验证工作者欢迎，除了其具有层次分明的验证结构和功能明确的组件外，还离不开 UVM 验证方法学自身所蕴含的运行机制。UVM 验证方法学主要有 phase 机制、objection 机制、config 机制和 sequence 机制等，接下来就分别介绍下这几种运行机制。

在 UVM 验证方法学中，phase 机制是控制整个验证平台按照特定的顺序有序执行的一种机制。如图 2.4[17]所示，分为两类。函数phase和任务phase对于 UVM 验证平台的运行起着至关重要的作用，它把一个大任务分割成一个个小任务，然后每一个小任务由每个 phase 执行，实现了更加精细化操作。比如实例化组件和 TLM 端口的一些连接，其他一些就是对仿真过后对数据进行报道和检测等。而在任务 phase[18]中，针对待测设计 DUT 进行仿真操作。在 phase 的执行中，会按照相关顺序从上到下自动执行，不需要人为的手动运行，这增加了验证平台的自动化程度，但是在不同的组件中，对于相同的 phase，采用的是深度优先法则。

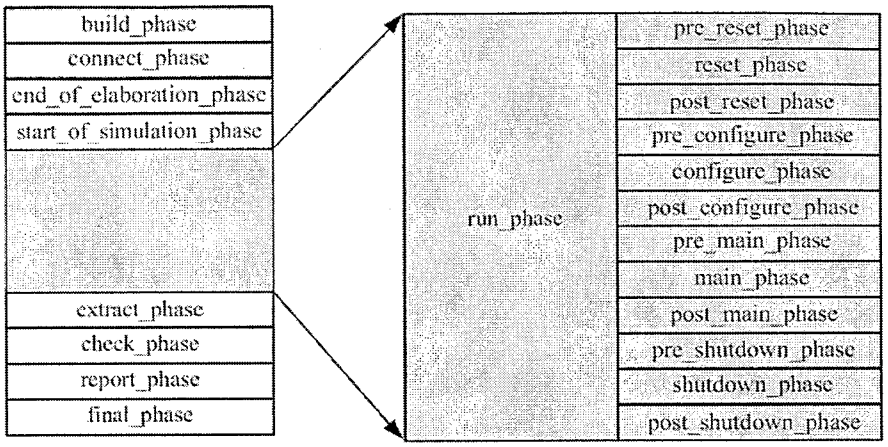

图 2.4 UVM 的 phase

Figure 2.4 UVM phase

Objection 机制是 UVM 验证平台中一个重要的监察机制，其作用是控制各个 phase 中任务的开始和结束，通过使用 raise_objection 和 drop_objection 这两个关键字来控制运行。当使用 raise_objection 提起进程时，其内的任务或函数就开始工作，按照顺序执行语句，直到遇到 drop_objection 才停止。在不同的 phase 中都可以提起 objection，当所有提起的 objection 全部执行完全后，整个验证平台的仿真过程就会结束。因为 objection 机制一般只用在消耗仿真时间的 phase，即用在任务 phase 中，所以 run_phase 的运行有两种方式，第一种就是在其他运行的 phase 中有 objection 提起，另外一种就是可以在 run_phase 中自己提起 raise_objection 和 drop_objection。否则，在这个组件中，run_phase 将不会执行。

Config_db机制也是 UVM 验证平台中的一个机制，传递的数据格式可以有不同种类，一般都有两个函数，set 和 get 函数，在原地址使用 set 函数传递数据，在目的地址用 get 函数接收数据，总是成对出现，也可以以一对多的形式。使用 config_db 传递数据时，除了设置传递数据的类型外，还需要四个参数，前两个参数组合起来表示地址，第三个参数表示 set 函数和 get 函数二者接收数据的名字，最后一个函数表示要传递的数据。因为待测模块与验证平台是使用虚接口连接的，所以，一般都是在 driver 中使用 config_db_set 函数发送激励到与待测模块连接的虚接口 interface 上。虽然 config_db 函数功能极其强大，可以在不同层次对同一个数据进行配置，但是如果目标地址出错而且有多个 config_db 函数，查找起来

不方便，所以这时可以使用它内置的一个函数 check_config_usage 来检查哪些 config_db 函数被配置了却没有被使用，数据会在仿真报告中给出。

Sequence机制也是UVM验证方法学在构建验证平台时使用的一个重要机制，它主要由激励产生器sequence与激励发射器sequencer组成。它把激励的产生，激励的传递和激励的发送分开来，使其更方便管理和运行，使整个验证效率提高。最开始的验证平台是使用driver来进行激励的产生、驱动与发射，这样会使driver功能复杂，不利于大型设计。所以为了提高效率，采用流水线操作，将这部分功能从driver中分割出来，独立设计为一个模块，这就体现了化繁为简的设计哲学。Sequence 机制的具体操作流程就是 sequence 向 sequencer 产生一个发送 sequence_item请求，然后sequencer将这个请求放入到内部的一个申请列表中，按照请求的优先级排序，然后查看driver是否申请了sequence_item，如果发现driver申请了sequence_item，则sequencer将此激励发送给driver，然后driver再返回一个接收成功的标志，这是一次握手协议，这就是一次完整的数据流操作，其中数据的流通是经过driver和sequencer之间内部的通道进行数据传输。当sequence发生后，就会执行它们内部的一个body任务，当然，还有 pre_body 和 post_body 任务，这两个在实际应用中并不常见。除了上面所提到的功能，sequence 应用最广泛的还是其仲裁机制和 virtual sequence 操作，对于 sequence 的仲裁方式，主要有 SEQ_ARB_FIFO 、 SEQ_ARB_WEIGHTED 、 SEQ_ARB_RANDOM 、 SEQ_ARB_STRICT_FIFO、SEQ_ARB_STRICT_RANDOM和SEQ_ARB_USER这几种，具体功能如表 2.1。Sequence 机制也有许多宏定义，当使用时可以直接调用，如 uvm_do 系列宏等，在其内还有锁定操作，比如 grab 和 lock 操作，二者优先级不同，其中，grab操作的优先级更高。当sequence较多时，若想实现sequence之间的同步，最好的方法就是使用 virtual sequence，在其内可以声明多个 sequence，然后按照你想要的执行顺序排列，这样当其被执行时，就会得到你想要的结果，它不产生 sequence_item，只是控制这些 sequence，在全局中起着调配的作用。

表 2.1 sequence 仲裁算法

Table 2.1 sequence arbitration algorithm

sequence 仲裁算法	仲裁算法说明
SEQ_ARB_FIFO	严格遵循先入先出
SEQ_ARB_WEIGHTED	加权仲裁
SEQ_ARB_RANDOM	随机仲裁
SEQ_ARB_STRICT_FIFO	先优先级，后先入先出
SEQ_ARB_STRICT_RANDOM	先优先级，后先入先出
SEQ_ARB_USER	用户自定义

2.3.4 TLM 通信

UVM 中要实现两个组件之间的通信，使用的是 TLM（Transaction Level Modeling）通信方式，可以构建一个强大的验证平台，具有多种类型的端口和操作可供选择。对于一个事务级建模而言，通信端口之间传输是使用 transaction，一个 transaction 就是封装了一组特定信息的数据，比如一次芯片的片上总线传输，需要地址、数据和控制信号等信息，把这些信息封装在一个类中就是一个 transaction。TLM[19]有许多方式对端口进行操作，操作类型主要是 put 操作、get 操作和 transport 操作这三种，一个组件可以有多个端口，也可以同时作为发起者和操作者。

UVM 中的端口主要有port端口、export端口和imp端口等，每种端口都有每种端口都有 put、get、peek、get_peek 和 transport 这五种类型[20]，其中每种类型都有阻塞、非阻塞和属于阻塞和非阻塞都可这三类。其中，put、transport 和 get 类型端口[20]对应着上文提到的 put 操作、get 操作和 transport 操作和 peek 类端口用于主动获取数据。而 get_peek 类端口则集合了 get 操作和 peek 操作，也就是可以作为 get 操作，也可以作为 peek 操作，这些端口可以进行互联。在这些端口中，analysis 端口很常用，因为它支持一对多操作且不区分阻塞和非阻塞，也就是一个 analysis 端口可以与多个 IMP 端口相连接，根据这个特性，可以把它说成是一个广播。当然，在 UVM 中也可以使用 FIFO 通信，在 FIFO 内部有很多端

口，可以在 connect_phase 中进行连接，使用方便。

2.3.5 寄存器模型

在 UVM 验证方法学中，寄存器模型[22]是很重要的一个部分，它最先来自于 VMM 的寄存器模型解决方案，经过 UVM 的继承与发展，成为 UVM 验证方法学的一大亮点。如果没有寄存器模型，那么对于待测设计 DUT 内部的寄存器访问将会变得复杂，而且不利于观测与检查当前待测设计的运行状态。对于一些测试用例，它需要时刻监测待测设计的某个寄存器的值，一旦它出现异常，就需要进行异常处理，这时就体现出寄存器模型的优势了，因为寄存器模型有一个后门访问机制，它可以不通过访问总线访问方式获得待测设计内部寄存器的值，不需要消耗仿真时间。

寄存器模型是模拟待测模块内部的寄存器[23]。Uvm_reg_filed代表寄存器的每一个比特，由若干个 uvm_reg_fields 可以组成一个 uvm_reg，它是比 uvm_reg_field 搞一个级别，可以代表一个具体的寄存器。而 uvm_reg_block 则是一个相对前两者来说一个大单位，可以由一个或多个 uvm_reg 组成，相当于一个寄存器组。在寄存器模型的基本信息定义完之后，还要添加它的 uvm_reg_map，用于寄存器模型进行前门访问时，进行访问地址时的一个指标，当待测设计的寄存器地址信息改动后，也需要在寄存器模型中进行相应的更新，以便于后期进行维护和处理。

在寄存器模型中，还需要介绍一个转换器，名字为 adapter，它的功能是在寄存器模型中转换数据格式，将寄存器模型能接受的数据格式和待测模块接受的数据格式相互转换，以实现寄存器模型发送命令或更新相应寄存器的值。转换器内部定义了两个函数，分别为bus2reg和reg2bus这两个函数。

在验证平台中，在寄存器模型中通过调用 read 和 write 这两个函数来实现前门访问。后门访问通过peek函数和poke函数来实现对待测模块进行读写操作。因为有后门访问操作，可以实现前门访问很多做不到的功能，比如可以直接对一个寄存器直接赋值等。但是后门访问操作不能在波形文件中发现痕迹，只能通过输出打印信息，这样会增加验证调试难度。

2.4 本章小结

本文主要介绍了验证策略与方法，然后对 UVM 验证方法学做了重点介绍。首先分析了三种验证方法的优缺点；其次介绍了芯片设流程和动态仿真验证；最后介绍 UVM 验证平台的整体架构，然后对其内部组件和运行机制进行分析，例如输入输出代理、参考模型和 TLM 通信等，还介绍了运行机制包括 phase 机制、objection 机制和 sequence 机制等。

第 3 章 通用 DMA 模块功能及验证计划

本章是对 DMA 模块结构及功能与验证策略与计划进行了介绍。该通用 DMA 是使用 AHB 总线进行数据传输，详细介绍了 DMA 的功能特点，然后采用灰盒验证的验证策略。

3.1 AHB 总线协议

3.1.1 AHB 总线协议简介

AHB[24](Advanced High-performance Bus)总线是一个使用很广泛的一个总线，其SOC系统如图 3. 1 所示。

一个AHB总线能够支持多个主设备和从设备，当master占中总线时，这个主设备会发送地址和控制信息给仲裁器，接着仲裁器会做出判断，若符合要求则将总线控制权交给此主设备。当从设备接收到信息后，会根据所得信息做出相应处理，并将处理结果反馈给主设备。在 AHB 总线协议中，从设备响应信号有四种类型，主要是OKAY、ERROR、RETRY和SPLIT。OKAY 状态主要是表示此次传输成功，ERROR 状态表示此次传输失败，RETRY状态表示此次传输未完成，SPLIT状态表示传输未完成，需要分离传输，一般是在从设备需要多个周期才能获得数据。其中只有 OKAY 状态是单周期反应，其余三种回复状态是两周期响应，使得主设备有足够的时间来处理下一次传输。

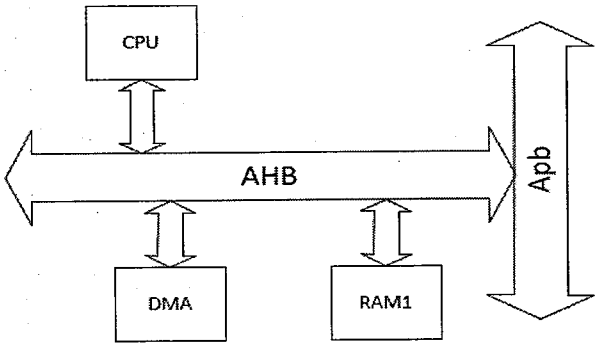

图 3. 1 AHB 总线连接的 SOC 系统

Figure 3.1 SOC system connected by AHB bus

3.1.2 AHB 总线数据传输实现

AHB 总线数据传输分为一次无需等待的数据传输、需要两个或多个等待周期的数据传输和 burst 传输等几种类型。

对于一次无需等待的数据传输，传输阶段分为两个部分，地址周期和数据周期，它们都为单个周期，其他的就是数据周期需要多个周期来完成，具体执行情况需要根据传输类型而决定。Burst 传输是使用流水线技术来进行数据传输，有八种类型，分别为SINGLE、INCR、WRAP4、INCR4、WRAP8、INCR8、WRAP16和INCR16等。其中最常用的就是 INCR 类型，但是不能穿越一个 1K 边界。AHB 总线有四种 transfer 类型，分别为IDLE、BUSY、NONSEQ和SEQ等。

3.2 通用 DMA 模块介绍

DMA(Direct Memory Access，直接存储器访问)在当今 SOC 系统中起着至关重要的作用，它在当今片上系统中的作用是可以取代 CPU 进行数据搬移，即从一个地方把数据取来然后送到另一个地方，而不需要 CPU 进行中断。DMA 有很多种类，大体上可以分为专用 DMA 和通用 DMA 两种，专用 DMA 一般是嵌入到某一专用模块内进行数据搬移，只能被此模块所使用，例如嵌入到 DSP 中的 DMA 模块等。通用 DMA 的功能和使用范围比专用 DMA 更强大，可以直接连接到总线上进行数据传输，通过 CPU 单元对通用 DMA 相应寄存器进行写入操作，它就可以通过得到的配置信息根据地址和控制信息进行数据搬移。解放了 CPU 的工作，增强了整体效率，已经成为现今 SOC 系统的必备模块。

3.2.1 DMA 模块架构

本文所验证的 DMA 是一个通用 DMA[25]，最多支持八个数据通道来传输数据，可以同时开八个通道[26]，也可以只开其中一个通道来进行数据传输，但是只有一个总线来进行数据传输，所以如果同时开多个通道来进行数据传输，但是每一时刻 DMA[27]控制器只能响应一个通道的数据传输请求。每个通道都是由一组寄存器来控制，通过这些寄存器写入的内容来确定其传输方式，可以设置它的不同的优先级。它支持 AHB 总线协议，在访问不同的外设时，总共有 16 组握手信号，而且，它还支持链式传输，具体原理图由图 3.2 可见，从那图中可以看到该

DMA 主要有四个部分，分别为 MASTER、SLAVE、ARBITER 和 DMA_CORE。其中 MASTER 和 SLAVE 主要功能是行使 DMA 的主模式和从模式的功能，当 CPU 通过总线对其配置时，它充当的是从设备，但是进行数据搬移时又充当主设备。

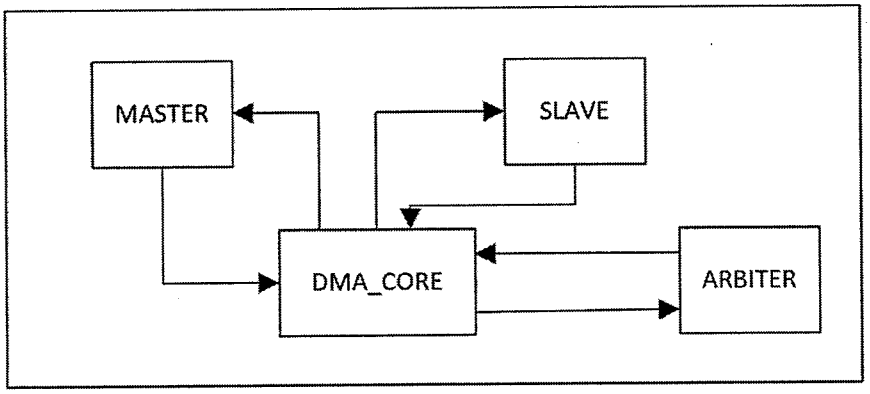

图 3.2 DMA 原理图

Figure 3.2 DMA schematic

3.2.2 DMA 内部寄存器

该通用 DMA 有八个通道，分别由控制寄存器 chnctrl，源地址寄存器 srcaddr，目的地址寄存器 dstaddr，传输数据数目寄存器 transize 和链式指针寄存器 llp 这几个寄存器控制。有一个中断寄存器，即 chabort，默认值为 0，即每个通道都使能，其中每一 bit 控制一个通道，该寄存器是只写寄存器，当向其内写 1 时，停止该通道传输数据，且其为自动清零，也就是触发之后就会自动恢复到默认值。具体功能如表 3.1 所示，介绍一组通道的寄存器，其余通道的寄存器与此类似，就是地址不同。

表 3.1 一组通道控制寄存器

Table 3.1 A set of channel control registers

寄存器名称	位数(bit)	类型	功能
chnctrl	32	可读可写	存储通道控制数据
srcaddr	32	可读可写	存储源地址开始地址
dstaddr	32	可读可写	存储目的地址开始地址

transize	32	可读可写	存储传输数据数目大小
llp	32	可读可写	存储下一组数据的首地址

3.2.3 DMA 功能描述

本文所验证的通用 DMA 具有数据搬移的作用，除了基本的功能外，还有通道仲裁、握手协议、链式传输和数据顺序传输等特色功能，接下来会一一介绍这些功能。

通道仲裁功能是通用 DMA 的一个特色功能，也就是每个数据传输通道都有两个优先级，一个是低优先级，另外一个是高优先级，DMA 控制器在进行通道仲裁时，若有多个通道同时需要传输数据，根据仲裁原理会优先选择高优先级的通道进行数据传输。

通用 DMA 最多支持 16 组握手协议，用于和低速设备之间的信息交换，每组握手协议有两个信号，分别为 req 和 ack，它们的握手协议如图 3.3 所示。在一次握手协议中，首先当准备传输数据时，req 信号会拉高，当 DMA 控制器看到 req 信号拉高后，就开始传输数据，当数据传输完毕后，ack 信号会拉高表示数据接收完成，当检测到 ack 信号拉高后，req 信号就会拉低表明数据传输完毕，紧接着 ack 信号也会拉低，这就是一次完整的握手数据传输。是否选择使用握手信号需要配置相应的寄存器来决定，可以使能其中一个或多个握手信号，这需要根据具体的设计而决定。握手协议可以解决设备之间的数据交换而保证不会出错，保证双方之间的信息正确，确保彼此身份，而且还可以有利于数据传输过程中不会丢失，造成信息泄露，一般用于异步模块之间的数据通信。

图 3.3 握手协议

Figure 3.3 Handshake agreement

链式传输也是通用 DMA 提供的一个功能，它可以进行多个数据块连续传输而不需要 CPU 的干预，也就是可以根据设置好的地址自动读取数据并写入到相应通道的寄存器中，自动执行且不需要人为干预。在链式传输开始之前，必须建立链表结构来描述数据块移动和关联的控制设置。列表的第一个元素（列表的头）是由通道控制寄存器描述的，列表的其余元素由链表描述符存储在内存中，链表描述符保存控件要加载到通道控制寄存器以继续数据传输的值。图 3.4 显示了一个链表结构的示例。当信道被启用时，DMA 控制器将首先根据信道传输数据通道控制寄存器。数据传输完成后，DMA 控制器将继续按照 chnll 指针进行数据传输。所指向的链表描述符的内容如果 ChnLLPointer 不为零，则 ChnLLPointer 将加载到通道控制寄存器。这个加载的描述符将成为列表的新头，此过程将重复，直到 ChnLLPointer 为零。当信道的终端计数中断（IntTCMask）被启用时，DMA 控制器将当列表头的数据传输完成时，生成中断并禁用通道。如果 ChnLLPointer 不为零，通道控制寄存器将预加载生成中断之前的下一个描述符。中断处理软件可以恢复通过重新启用通道实现链转移

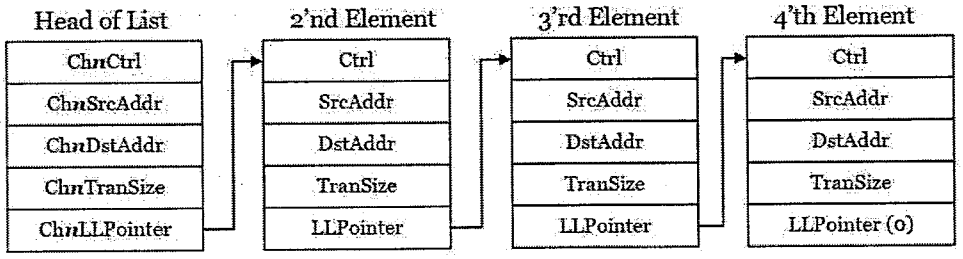

图 3.4 链表结构

Figure 3.4 Linked list structure

数据传输顺序也是通用 DMA 的一个功能，它提供了三种地址控制模式：递增模式、递减模式和固定模式。在递增模式下，DMA 控制器获得了源或目的地址的数据后地址开始一直增加。在递减模式下，地址在 DMA 控制器获得源或目的的地址数据后一直减少。在固定模式下，在 DMA 控制器获得源或目的的数据后地址保持不变。当源地址控制模式与目的地址控制模式相同时，DMA 控制器在源和目的地址之间保持相同的数据字节顺序。当源地址控制模式与目的地址控制模式相反时，数据写入目的的字节顺序与从源读取的字节顺序相反。数据固定模式的顺序与递增模式的顺序相同

3.3 验证策略与计划

3.3.1 验证策略

我们采用灰盒验证的方法，在搭建的 UVM 验证平台基础上，使用断言分析[28]和参考模型相结合的验证方法，而且还采用新的数据对比方法，不仅可以监测某一个或一组具体信号是否符合设计规范，而且还降低了参考模型的难度。

3.3.2 验证计划

针对 DMA 的结构和功能，对其功能点进行提取，根据覆盖率[29]统计设计出验证计划，表 3.2 是针对被测设计 DMA，设计的一些验证计划，进行模块级验证。

表 3.2 验证计划表

Table 3.2 Verification schedule

验证项目	具体描述
寄存器读写检测	检查 DMA 内部寄存器读写功能是否正确
复位检查	检查 DMA 的复位操作是否正确
master 检查	检查 DMA 作为主设备是否正确
slave 检查	检查 DMA 作为从设备是否正确
burst 传输检查	验证 DMA 通道的 burst 传输是否正确
通道仲裁检查	验证 DMA 通道仲裁情况
源地址变化类型检查	检查 DMA 源地址变化
目的地址变化类型检查	检查 DMA 目的地址变化
握手协议	检查 DMA 与外设的握手协议
链式传输	检查 DMA 的链式传输是否正确

3.4 本章小结

本章是对 DMA 模块结构及功能与验证策略与计划进行了介绍。首先介绍了 AHB 总线协议的通信方式、数据传输类型和互联结构；然后介绍了通用 DMA 的结构与功能，对其整体架构、内部寄存器和功能应用做了说明，介绍了 DMA 与外设之间的握手协议和链式传输等功能；最后介绍了验证策略与验证计划。

第 4 章 基于 UVM 的验证平台构建与实现

本章主要是先整体描述整个验证平台的组成结构，然后对组成验证平台的各个组件进行详细介绍，最后针对传统数据比对方法，提出了一种新的数据比对方法，提高了仿真效率。

4.1 验证平台整体架构

传统的验证平台是使用 Verilog 语言编写的，与待测模块关联性太强，不仅费时费力验证效率不高，而且可迁移性弱，可重用性低。针对这种情况，本论文使用 SystemVerilog 语言和 UVM 验证方法学[30]，结合 DMA 模块特点，设计了一个可重用的验证平台。为了体现出验证平台可重用性，不仅将待测模块与验证平台使用虚接口 interface 连接[31]，使二者关联性减弱，而且考虑到待测模块的接口协议，设计了相应的可重用验证组件，数据包使用 TLM 端口[32]进行传输。验证平台各个组件及连接关系如图 4.1 所示。

在图 4.1 所示的验证平台整体架构中，可以看到主要有 master_agti、master_agto、reg_model、转换器、覆盖率收集模块、断言模块、env、sequences、test 等组件构成。它们之间使用 TLM 端口互联，其中一些数据使用 config_db 机制进行数据传输。其中输入代理通过内部的 driver 和 monitor 组件将 DMA 的接口信息驱动到总线上和收集收集来自总线上的数据并打包发送给参考模型，输出代理将收集 DMA 的输出数据并将数据发送给计分板。在数据比对方面，还专门设置了数据比对模块，将来自参考模型和待测模块的数据进行预处理后进行比对，并将比对结果输出，接下来章节会详细介绍组成验证平台的各个组件及其功能。

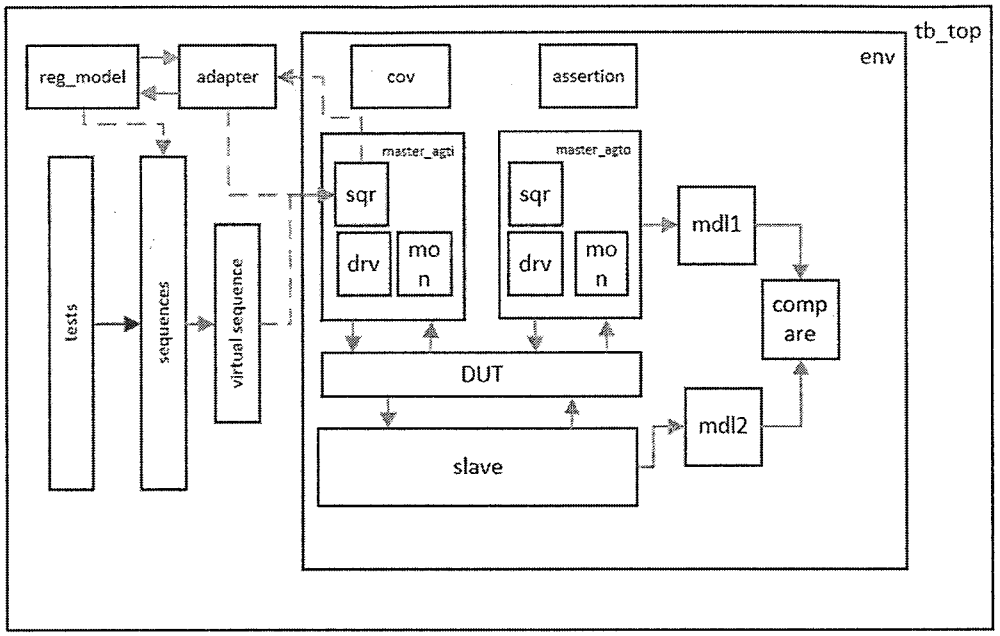

图 4.1 验证平台整体架构图

Figure 4.1 The overall architecture diagram of the verification platform

4.2 验证平台所用主要组件

本文在搭建的过程中使用UVM验证方法学所自带的库函数来进行各个组件的设计，最后根据通信机制将各个组件相联系起来，使各个组件之间互相通信，进行数据之间的交流。一般是使用事务级通信，将一组信息打包放在一个事务里，比如激励的产生就是一组事物transaction，它由激励产生器sequence产生，然后由激励发射器sequencer通过与激励驱动器driver之间默认的通道进行传输，每一笔数据传输都是一个事务。

4.2.1 输入输出代理组件

master_agt_i组件继承于容器类uvm_agent，如图 4.2 所示，作为一个主设备，模拟CPU对DMA内部相应寄存器进行配置。它相当于是一个容器类，里面封装了 driver，monitor 和 sequencer。在其build_phase和connect_phase中完成添加组件、实例化端口、设置内部端口连接方式和接收 config_db 传输来的数据等。如图 4.3 所示，在build_phase中实例化driver、sequencer和monitor等组件。

```
class my_agent extends uvm_agent;
  my_driver drv;
  my_monitor mon;
  my_sequencer sqr;

  uvm_analysis_port #(my_transaction) ap;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  extern virtual function void build_phase(uvm_phase phase);
  extern virtual function void connect_phase(uvm_phase phase);

  `uvm_component_utils(my_agent)
endclass

function void my_agent::build_phase(uvm_phase phase);

function void my_agent::connect_phase(uvm_phase phase);
endfunction
```

图 4.2 agent 组件定义

Figure 4.2 Agent component definition

```
function void my_agent::build_phase(uvm_phase phase);
  super.build_phase(phase);
  if(is_active == UVM_ACTIVE) begin
    drv = my_driver::type_id::create("drv", this);
    sqr = my_sequencer::type_id::create("sqr", this);
  end
  mon = my_monitor::type_id::create("mon", this);

endfunction
```

图 4.3 agent 中 build_phase 配置

Figure 4.3 Build_phase configuration in agent

在输入代理中的激励发射器 sequencer 组件是将从 sequence 传来的数据，通过与 driver 之间的数据传输通道，将一笔 transaction 通过端口传递给 driver。如图 4.4 所示，它不产生激励，只是把激励发射出去。

```
class my_sequencer extends uvm_sequencer #(my_transaction);

  //ral_block_dma_reg p_rm;
  ral_block_dma p_rm;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  `uvm_component_utils(my_sequencer);
endclass
```

图 4.4 激励驱动器 sequencer 定义

Figure 4.4 Excitation driver sequencer definition

激励驱动器 driver 的作用主要是将从 sequencer 传来的一笔 transaction，按照 AHB 总线协议，即将地址信号，控制信号和数据信号驱动到与 DMA 连接的虚接口 interface 上，完后对 DMA 的操作。Driver 相当于是输入代理的核心组件之一了，如图 4.5 所示，其内定义了一个读写任务，它主要任务是将得到的数据发送到与待测模块 DMA 连接的虚接口上，通过总线向 DMA 内部的寄存器写入或者读出相应数据，然后 DMA 就会启动占用总线，根据所得到的配置信息，启动对应的数据传输通道 0-7 的一个或多个通道进行数据传输。

在设计 DMA 的激励驱动器 driver 时，其获得每笔事务 transaction 都是通过与 sequencer 之间的秘密通道传输过来的，其通信方式遵循握手协议。如果 driver 获得一组事务，在当前这组 transaction 未传输完成之前，它是不能接受下一组事务的传输任务的，只有当前传输完毕后，driver 会发送一个传输完毕信号 item_done 给 sequencer，当 sequencer 收到此信号如果此时还有正在等待的 sequence，那么 sequencer 就会将此 sequence 发送给 driver，这就是握手传输。Driver 和 sequencer 之间的通道是 UVM 已经定义好的，使用的是 TLM 通信方式，端口名字为 seq_item_port，如图 4.6 所示，可以通过 get_next_item 和 item_done 函数调用来获得下一子事务和传递此次事务传输结束标志，。

```
class my_driver extends uvm_driver #(my_transaction);

  virtual my_if vif;
  `uvm_component_utils(my_driver)

  function new(string name = "my_driver", uvm_component parent = null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if(!uvm_config_db#(virtual my_if)::get(this, "", "vif", vif))
      `uvm_fatal("my_driver", "virtual interface must be set for vif!!!
  endfunction

  extern virtual task main_phase(uvm_phase phase);
  extern virtual task ahb_write(logic [31:0] addr,logic [31:0] data[$],
  logic [2:0] hburst,logic hwrite,logic hsel);

endclass
```

图 4.5 驱动器 driver 的部分定义

Figure 4.5 Partial definition of the driver

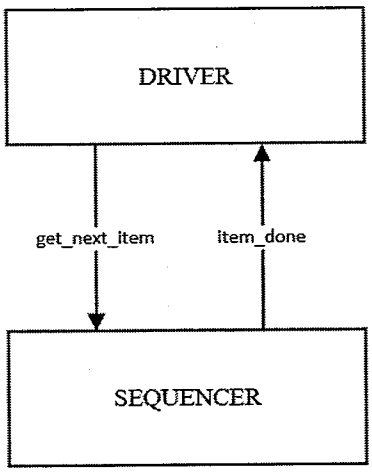

图 4.6 driver 与 sequencer 之间通信

Figure 4.6 Communication between driver and sequencer

激励监视器 monitor 组件主要是按照 AHB 协议规则，收集虚接口 interface 上的数据，然后打包成一个 transaction 通过 TLM 端口发送给其他组件，如参考模型，覆盖率收集模块和断言检查模块等。DMA 内部寄存器被写入相应信息后，DMA 就会被启动，在 DMA 输入输出接口上就会有相应数据，那么 monitor 就充当了收集这些信息的功能，不论是向 DMA 写入的数据还是 DMA 自身输出的数据信息，它都会将这一组信息收集好发给需要的组件，可以做到实时处理，如图 4.7 所示，是 monitor 定义的部分代码，其收集工作主要在 main 函数中进行。

```
class my_monitor extends uvm_monitor;

  virtual my_if vif;

  uvm_analysis_port #(my_transaction) ap;

  `uvm_component_utils(my_monitor)
  function new(string name = "my_monitor", uvm_component parent = null);

  virtual function void build_phase(uvm_phase phase);

  extern task main_phase(uvm_phase phase);
  extern task collect_one_phase(my_transaction tr);
  extern task my_print(my_transaction tr);

endclass
```

图 4.7 激励监测器 monitor 定义

Figure 4.7 Motivation monitor monitor definition

master_agt_o 组件和 master_agt_i 组件一样，为了体现出验证平台的可重用性，所以使用的都是同一组组件。但是不同的是，驱动器 driver 与发射器 sequencer 不使能，也就是通过配置后，只有一个监测器 monitor，用于收集总线上的数据。

4.2.2 寄存器模型组件

寄存器模型在 UVM 搭建的验证平台中起着很关键的作用，使用寄存器模型可以对 DMA 内部的寄存器进行建模，然后可以使用它的一些内建函数，如 read、write、peek、poke、update 和 set 等函数，对已建模的寄存器进行操作。通过调用相应函数，会产生相应的 sequence，经过转换器 adapter 转换后，传给设置好的 sequencer。当然，使用寄存器模型还有一个好处就是可以使用后门访问操作，这对想监测某一寄存器值来说是一个很好的选择。而不使用寄存器模型只能通过前门访问对待测模块的寄存器进行读取或写入操作，耗费仿真时间但是在波形文件可以观测到相关操作数据。

在设计的此次 DMA 的验证平台中，共对其内部 40 个寄存器进行建模分析，生成了 reg_model 组件对 DMA 内部的寄存器进行建模，对于如何对一个寄存器进行建模，如图 4.8 所示。首先要定义一个寄存器，一般是使用 uvm_reg 来进行建模，对于一个寄存器有多位的情况，需要定义多个 uvm_reg_field，定义它的名字和相关参数，如域的宽度、位置、存取方式和上电默认值等；其次，要把这个 uvm_reg 类型的寄存器放入到一个 uvm_reg_block 类中设置 map 地址并将其实例化；最后将这个寄存器模型加入到整个验证平台中，在 sequence 中使用寄存器模型的相关函数就可以调用设置好的寄存器模型了。在 UVM 验证方法学中寄存器模型还提供了一系列的 sequence 检测函数，比如说寄存器上电复位默认值检测 sequence 和检查寄存器读写 sequence 等，通过这些函数可以快速方便的验证该寄存器模型是否正确，如图 4.9 所示，在该 sequence 中声明此函数并实例化，最后触发，就可以自动检测设计的寄存器模型上电复位值是否正确，很适合用于刚设计的寄存器模型的自检测。

```
rand uvm_reg_field ch_0_en;
rand uvm_reg_field ch_0_int_tc_mask;
rand uvm_reg_field ch_0_int_err_mask;
rand uvm_reg_field ch_0_int_abt_mask;
rand uvm_reg_field ch_0_dst_req_sel;
rand uvm_reg_field ch_0_src_req_sel;
rand uvm_reg_field ch_0_dst_addr_ctl;
rand uvm_reg_field ch_0_src_addr_ctl;
rand uvm_reg_field ch_0_dst_mode;
rand uvm_reg_field ch_0_src_mode;
rand uvm_reg_field ch_0_dst_width;
rand uvm_reg_field ch_0_src_width;
rand uvm_reg_field ch_0_src_burst_size;
rand uvm_reg_field ch_0_priority;
local uvm_reg_data_t m_data;
local uvm_reg_data_t m_be;
local bit            m_is_read;

covergroup cg_bits ();
   option.per_instance = 1;
   option.name = get_name();
   ch_0_en: coverpoint {m_data[0:0], m_is_read} iff(m_be) {
      wildcard bins bit_0_wr_as_0 = {2'b00};
```

图 4.8 寄存器模型部分代码

Figure 4.8 Part of the register model code

```
class dma_reg_reset_seq extends uvm_sequence;
    `uvm_object_utils(dma_reg_reset_seq)

    function new(string name = "dma_reg_reset_seq");
        super.new(name);
    endfunction

    virtual task body();
        uvm_status_e status;
        uvm_reg_data_t value;
        uvm_reg_hw_reset_seq ckseq;

        my_sequencer sqr;
        $cast(sqr,m_sequencer);

        ckseq = new("ckseq");
        ckseq.model = sqr.p_rm;
        ckseq.start(null);


    endtask

endclass
```

图 4.9 寄存器模型上电复位 sequence 定义

Figure 4.9 Register model power-on reset sequence definition

在涉及到寄存器模型中，那就需要设计一个转换器 adapter 了，它在寄存器模型中必不可少，可以说是至关重要的一个部分。在本文设计的验证平台中，如图 4.10 所示，这个组件的主要作用就是将 DMA 接受的数据格式与寄存器模型接受的数据格式进行相互转换，主要通过 reg2bus 和 bus2reg 这两个函数实现，如

图 4.11 和 4.12 所示，前者 reg2bus 函数将寄存器模型接受的数据格式转换成 DMA 总线能接受的数据格式，例如地址和控制信号等，后者 bus2reg 是是将 DMA 总线能接受的数据格式转换成寄存器模型能接受的数据格式，将相关信息转换后存入一个 rw 的事务中，然后传给寄存器模型去更新内部相关数据。

```
class my_adapter extends uvm_reg_adapter;
    string tID = get_type_name();

    `uvm_object_utils(my_adapter)

  function new(string name="my_adapter");

  function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);


  function void bus2reg(uvm_sequence_item bus_item, ref uvm_reg_bus_op rw);

endclass : my_adapter
```

图 4.10 转换器 adapter 定义

Figure 4.10 Converter adapter definition

```
function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);
  my_transaction tr;
  tr = new("tr");
  `uvm_info("my_adapter","reg2bus-----rw.addr",UVM_LOW);
  //$display("rw.addr=%0h",rw.addr);
  tr.haddr = rw.addr;
  //$display("tr.addrm=%0h",tr.addrm[0]);
  tr.hwrite = (rw.kind == UVM_READ) ? 1'b0: 1'b1;
  //$display("tr.hwrite=%0h",tr.hwrite);
```

图 4.11 reg2bus 函数部分代码

Figure 4.11 Reg2bus function part of the code

```
function void bus2reg(uvm_sequence_item bus_item, ref uvm_reg_bus_op rw);
  my_transaction tr;
  if(!$cast(tr, bus_item)) begin
    `uvm_fatal(tID,
     "Provided bus_item is not of the correct type. Expecting bus_transaction")
     return;
  end
  rw.kind = (tr.hwrite == 1'b0) ? UVM_READ :UVM_WRITE;
  //`uvm_info("my_adapter","bus2reg------rw.addr",UVM_LOW);
  //foreach(tr.addrm[i]) begin
    rw.addr = tr.haddr;
```

图 4.12 bus2reg 部分代码

Figure 4.12 Bus2reg part of the code

4.2.3 激励产生器 sequence 与 virtual sequence 组件

在本文设计的验证平台中设计了不同的 sequence，它的主要目的就是产生事务级激励，经过 driver 驱动到与 DMA 连接的总线上对其进行相应寄存器配置。

如图 4. 13 所示，在每一个sequence中都有一个body任务，当其被调用时就会自动执行这个body任务。

Virtual sequence 组件是将不同的 sequence 以不同的组合排列，实现 sequence 之间的同步。因为不同的 sequence 可能需要不同的 sequencer 来发射，而且之间还有先后顺序关系，所以使用 virtual sequence 可以很好的实现上述目标，如图 4. 14 所示，显示的是使用 virtual sequence 同时开启通道 0 和通道 1 进行数据传输。

```
function  new(string name= "ch0_seq");
   super.new(name);
endfunction

virtual task body();
   uvm_status_e   status;
   uvm_reg_data_t value;
   my_sequencer sqr;
   $cast(sqr,m_sequencer);
   if(starting_phase != null)
      starting_phase.raise_objection(this);

   repeat(2000) begin
   tr = new();
   tr.randomize();

   sqr.p_rm.src_addr1.write(status, 32'd100, UVM_FRONTDOOR);
   sqr.p_rm.dst_addr1.write(status, 32'd56, UVM_FRONTDOOR);
   sqr.p_rm.transize1.write(status, tr.size0, UVM_FRONTDOOR);
   sqr.p_rm.chn_1_ctrl.write(status, tr.ct0, UVM_FRONTDOOR);
   sqr.p_rm.src_addr0.write(status, 32'd100, UVM_FRONTDOOR);
```

图 4. 13 通道 0 的 sequence

Figure 4.13 Sequence of channel 0

```
class ch0_vseq extends dma_vseq_base;
    `uvm_object_utils(ch0_vseq)

    function new(string name = "ch0_vseq");
        super.new(name);
    endfunction

    virtual task body();
        ch0_seq ch0_test = ch0_seq::type_id::create("ch0_test");
        ch1_seq ch1_test = ch1_seq::type_id::create("ch1_test");
        repeat(1) ch0_test.start(sqr);
        //ch1_test.start(sqr);
    endtask

endclass
```

图 4. 14 virtual sequence 示例

Figure 4.14 Virtual sequence example

4.2.4 断言模块

本文搭建的验证平台是使用灰盒验证，使用了断言分析对部分内部信号进行监测，可以快速发现哪些数据是错误的，减少定位错误时间，如图 4.15 所示，可以监测一个信号是否有效。断言的设计分为三个部分，第一部分是写断言语句，也就是 sequence；第二部分就是设计属性，也就是设计 property；第三部分就是断言，也就是 assertion，通过这三个步骤，一个断言语句就完成了。

```
// Reuseable property to check that a signal is in a sa
property SIGNAL_VALID(signal);
  @(posedge hclk)
  !$isunknown(signal);
endproperty: SIGNAL_VALID
```

图 4.15 断言信号有效性

Figure 4.15 Assert signal validity

4.2.5 覆盖率收集模块

覆盖率收集模块是用来测试验证是否完备的一个指标，根据验证计划提取出测试点，构建覆盖率模型，仿真后给出覆盖率报告。

本文所设计的覆盖率模型主要根据所设计的测试用例，如图 4.16 所示，该覆盖率收集模块在 main 函数中进行采样，最为一个自定义组件最后嵌入到整个验证平台中。该覆盖率收集模型主要对源地址数据传输宽度 srcwidth、目的地址数据传输宽度 dstwidth、源地址突发传输 burst 类型、源地址控制寄存器srcctrl、目的地址控制寄存器dstctrl、传输数据大小 size 和优先级 prio 等测试点进行采样，根据设计说明书确定每个测试点的仓数大小，然后编写相应的覆盖组。

```
function my_cov::new(string name, uvm_component parent);
  super.new(name, parent);
  Cov1 = new;
 // extrans = new;
endfunction

function void my_cov::build_phase(uvm_phase phase);
  super.build_phase(phase);
  port = new("port", this);
endfunction

task my_cov::main_phase(uvm_phase phase);
  super.main_phase(phase);
  extr = new;
  while(1) begin
    port.get(tr);
    //Cov.sample();
    extr = new;
```

图 4.16 覆盖率收集模块

Figure 4.16 Coverage collection module

4.3 验证平台数据对比策略

为了保证数据传输正确，在完成传输后要进行数据比对。传统的数据比对方法有两种，一种是构建复杂的参考模型，这种方法需要参考模型很精确，需要花费大量时间和精力来构建参考模型，验证效率不高；另外一种方法是在数据传输完成后，根据配置信息通过总线取出原地址和目的地址的数据，然后进行比对。但是这样会造成一次配置信息进行了两次取数据，如果第一种方法完成一次数据比对只是取回数据——写入数据这两个操作，而第二种方法完后一次数据比对则是取回数据——写入数据——取回数据三个操作，这样会花费仿真时间，使验证时间增加大约 50%，不利于大规模验证，具体测试如表 4.1 所示。

表 4.1 不同数据对比方法消耗时间

Table 4.1 Different data comparison methods consume time

测试用例	取回-写入（min）	取回-写入-取回（min）
Case1	0.78	1.24
Case2	2.01	3.04
Case3	5.25	7.9

为此，本文使用 UVM 验证方法学提出一种通过使用 config_db 方法解决。在数据传输完毕后，通过在顶层模块 top_tb 中直接从从设备 slave 中取出数据，

通过 config_db_set 函数传输出来，然后进行处理后传给数据比对模块。这样不仅能保证数据正确性，避免编写复杂的参考模型，而且还能节约仿真时间，提高仿真效率，三种方法比对数据如表 4.2 所示。

表 4.2 三种方法比对数据

Table 4.2 Three methods to compare data

算法	参考模型复杂度(归一化处理)	仿真时间(归一化处理)
传统方法一[33]（复杂参考模型，取回-写入）	1	2/3
传统方法二[34]（简单参考模型，取回-写入-取回）	1/4	1
新方法（简单参考模型，config_db 方法）	1/4	2/3

4.4 本章小结

本章主要介绍了该 DMA 验证平台的具体实现。使用 UVM 验证方法学，先整体描述整个验证平台的组成结构，对验证平台的架构图进行分析；然后对组成验证平台的各个组件进行详细介绍，主要包括输入输出代理 agent、寄存器模型、断言分析模块和覆盖率收集模块；最后针对传统数据比对方法，提出了一个新的数据比对方法，提高了验证效率。

第 5 章 验证计划及覆盖率统计

本章主要是介绍了验证计划，然后检查代码覆盖率与功能覆盖率情况。最后经检查验证，代码覆盖率为 99. 8%，功能覆盖率为 100%，达到了验证要求。

5.1 验证平台测试流程

该验证平台设计思路是通过 constraint 函数生成随机激励，然后在各个组件间数据（transaction）由 TLM 端口传输，进行覆盖率收集，最后给出验证结果。通过编写 Makefile 脚本实现验证平台自动化处理。

验证环境配置后，通过编写脚本自动化运行，收集记录仿真信息和打印结果，具体测试流程如图 5. 1 所示。使用 UVM 验证方法学中的 UVM_TESTNAME 函数指定所要执行的测试用例，在 testbench 中通过 run_test 函数获得所要执行的测试用例名字来启动验证平台，初始化验证组件，产生测试向量，接着产生相应的 sequence，通过 sequencer 发送给 driver，然后由 driver 驱动到与 DUT 连接的虚接口 interface 上，通过向 DMA 写入或读出数据启动 DMA。当通过总线向 DMA 配置相应寄存器后，DMA 就会按照配置信息进行数据搬移。model1 通过取得配置的信息，模拟 DMA 在其内内部模拟数据搬移，并将数据打包发给 compare，model2 通过取得 slave 内部的数据搬移信息，将数据打包发给 compare。最后 compare 对接收到的数据进行比对，并给出比对结果，若数据比对成功，则打印 compare pass 的信息，收集覆盖率信息。若比对失败，则不仅打印出 compare fail 的信息，还会打印出配置信息和比对数据，方便查错。

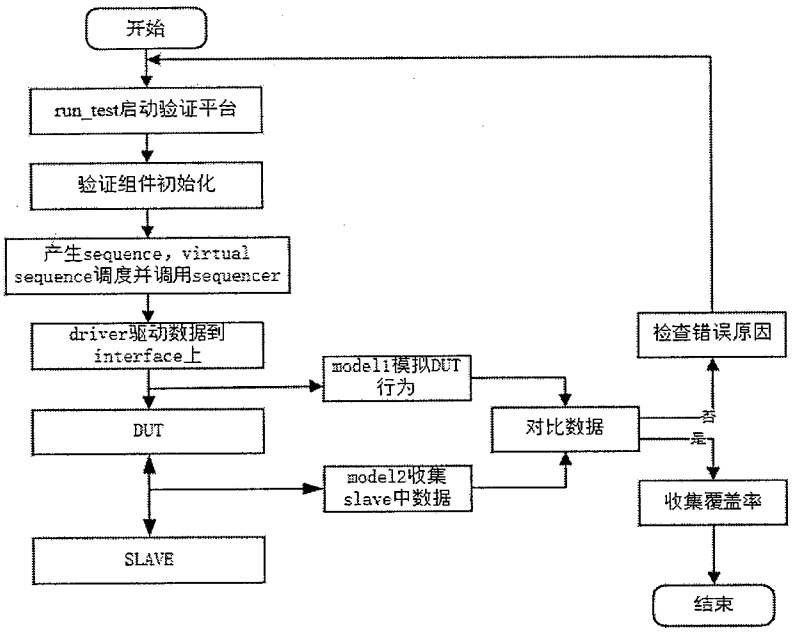

图 5.1 验证平台测试流程图

Figure 5.1 Verification platform test flow chart

5.2 测试用例

测试用例是整个测试流程中必不可少的一个关键环节，它所代表的是整个测试激励的生成，当基于 UVM 的验证平台搭建完成后，根据验证计划设计相应的测试用例，以功能覆盖率为驱动，测试用例所产生的测试激励能否完全覆盖整个验证计划并完成所有测试点的测试是检验验证是否完备的一个标准。

本验证平台的所采用的测试激励是使用带约束的随机验证，它与定向的测试激励不同，定向的测试激励需要手动编写每一个测试向量，会拖慢项目进度并会漏测一些功能点，而带约束的随机激励可以随机的产生某一范围的测试数据，这样可以在有限的时间覆盖到基本点，并且可以打破验证工程师的思维定式，覆盖到一些意想不到的功能点。根据验证计划所描述的内容，设计了检测该款通用 DMA 的功能点的测试用例，如测试内部寄存器访问情况、硬件握手功能、通道仲裁功能、链式传输功能和源地址与目的地址变化等情况，主要的设计的测试用例如表 5.1 所示。

表 5.1 测试用例及其描述

Table 5.1 Test case and its description

测试用例	用例名称	具体描述
ch0	通道 0 测试用例	启动通道 0 来进行数据传输
ch1	通道 1 测试用例	启动通道 1 来进行数据传输
ch2	通道 2 测试用例	启动通道 2 来进行数据传输
ch3	通道 3 测试用例	启动通道 3 来进行数据传输
ch4	通道 4 测试用例	启动通道 4 来进行数据传输
ch5	通道 5 测试用例	启动通道 5 来进行数据传输
ch6	通道 6 测试用例	启动通道 6 来进行数据传输
ch7	通道 7 测试用例	启动通道 7 来进行数据传输
2chs	同时开 2 个通道进行数据传输测试用例	启动 2 个通道来进行数据传输，查看数据搬移结果
3chs	同时开 3 个通道进行数据传输测试用例	启动 3 个通道来进行数据传输，查看数据搬移结果
4chs	同时开 4 个通道进行数据传输测试用例	启动 4 个通道来进行数据传输，查看数据搬移结果
5chs	同时开 5 个通道进行数据传输测试用例	启动 5 个通道来进行数据传输，查看数据搬移结果
6chs	同时开 6 个通道进行数据传输测试用例	启动 6 个通道来进行数据传输，查看数据搬移结果
7chs	同时开 7 个通道进行数据传输测试用例	启动 7 个通道来进行数据传输，查看数据搬移结果
8chs	同时开 8 个通道进行数据传输测试用例	启动 8 个通道来进行数据传输，查看数据搬移结果
reg_write	寄存器读写测试用例	验证 DMA 内部寄存器读写功能

handshake	硬件握手测试用例	验证 DMA 数据传输的硬件握手功能
chain_tranfer	链式传输测试用例	验证 DMA 数据传输的链式传输功能
data_order	源和目的地址变化方式测试用例	验证 DMA 数据传输时源地址与目的地址变换格式
channel_arbiter	通道仲裁测试用例	验证 DMA 多个通道传输情况

在上表 5.1 的测试用例中，源地址和目的地址都有三种变化，分别是自增、自减和固定，在通道进行数据搬移时会按照这九种方式中的一种进行，但是该通用 DMA 在同一时刻只响应一个通道的数据传输，这就涉及到数据传输通道仲裁的情况了。对于一个刚搭建好的验证平台而言，在进行大规模复杂环境验证之前，要其有效性进行测试，可以先采用简单地测试用例进行验证，待跟预期结果相一致后，在内部 TLM 通信等各种机制都没有问题后，再运用复杂的测试用例来进行验证，这样可以进行检错，避免出现错误时不知道是平台错误还是待测模块本身就有设计缺陷。

5.3 验证结果分析

本验证平台是使用 Mentor Graphics 公司的 QuestaSim 仿真软件来进行验证仿真。QuestaSim 仿真软件可以在 Linux 或者 Windows 环境下运行，这样对于只有 Windows 系统的用户有了很多便利，不用去额外安装虚拟机或者重装系统了，对于验证环境的配置提供了多种选择，对于一些验证工程师也是一个很好的选择，它支持 SystemVerilog 语言，而且还带有 UVM 的函数库，对于断言分析和覆盖率收集都支持，功能强大且实用性强。

5.3.1 验证平台编译结果

QuestaSim 仿真软件是 Modelsim 仿真软件的升级版，它能编译 SystemVerilog 和 Verilog 语言，支持混合编译功能，当编译完待测模块的设计代码后，通过载入 UVM 的库函数，然后编译验证平台的组件，编译过程如图 5.2 所示，可以看

到编译信息和结果，比如编译时间、编译对象和编译结果等。

```
Top level modules:
        --none--
End time: 14:18:41 on Apr 03,2021, Elapsed time: 0:00:01
Errors: 0, Warnings: 0
vlog +incdir+../uvm_tb/env ../uvm_tb/env/env_pkg.sv
QuestaSim vlog 10.4c Compiler 2015.07 Jul 20 2015
Start time: 14:18:41 on Apr 03,2021
vlog "+incdir+../uvm_tb/env" ../uvm_tb/env/env_pkg.sv
-- Compiling package env_pkg
-- Importing package mtiUvm.uvm_pkg (uvm-1.1d Built-in)
** Note: (vlog-2286) Using implicit +incdir+D:/questasim/uvm-1.1d/../verilog_src
-- Importing package agent_pkg
** Warning: ../uvm_tb/env/my_model.sv(136): (vlog-2697) MSB of part-select into

Top level modules:
        --none--
End time: 14:18:43 on Apr 03,2021, Elapsed time: 0:00:02
Errors: 0, Warnings: 1
vlog +incdir+../uvm_tb/sequences ../uvm_tb/sequences/sequence_pkg.sv
QuestaSim vlog 10.4c Compiler 2015.07 Jul 20 2015
Start time: 14:18:43 on Apr 03,2021
vlog "+incdir+../uvm_tb/sequences" ../uvm_tb/sequences/sequence_pkg.sv
-- Compiling package sequence_pkg
-- Importing package mtiUvm.uvm_pkg (uvm-1.1d Built-in)
```

图 5.2 验证平台编译结果

Figure 5.2 Verify platform compilation results

在编译完验证平台的主要组成组件后，接下来就是编译测试用例 sequences，一般情况下为了管理方便，我们会把这些测试用例放到一个包中，然后每次添加或修改测试用例就需要修改包中相应的信息，编译结果如图 5.3 所示。当所有模块编译完成后，仿真软件会给出最顶层模块的编译结果，如图 5.4 所示，若没有编译错误发生，那就说明整个编译过程没有出错，但是编译结果只能保证设计代码和验证平台设计没有语法错误，其功能是否正确还需要进行动态仿真验证。

```
Top level modules:
        --none--
End time: 14:18:43 on Apr 03,2021, Elapsed time: 0:00:02
Errors: 0, Warnings: 1
vlog +incdir+../uvm_tb/sequences ../uvm_tb/sequences/sequence_pkg.s
QuestaSim vlog 10.4c Compiler 2015.07 Jul 20 2015
Start time: 14:18:43 on Apr 03,2021
vlog "+incdir+../uvm_tb/sequences" ../uvm_tb/sequences/sequence_pkg
-- Compiling package sequence_pkg
-- Importing package mtiUvm.uvm_pkg (uvm-1.1d Built-in)
-- Importing package agent_pkg
** Note: (vlog-2286) Using implicit +incdir+D:/questasim/uvm-1.1d/.
** Warning: ../uvm_tb/sequences/case0_cfg.sv(112): (vlog-2240) Trea
** Warning: ../uvm_tb/sequences/ch0_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch1_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch2_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch3_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch4_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch5_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch6_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/ch7_seq.sv(67): (vlog-2240) Treatin
** Warning: ../uvm_tb/sequences/handshake.sv(112): (vlog-2240) Trea

Top level modules:
        --none--
End time: 14:18:44 on Apr 03,2021, Elapsed time: 0:00:01
Errors: 0, Warnings: 10
vlog +incdir+../uvm_tb/virtual_sequencer ../uvm_tb/virtual_sequence
QuestaSim vlog 10.4c Compiler 2015.07 Jul 20 2015
Start time: 14:18:45 on Apr 03,2021
vlog "+incdir+../uvm_tb/virtual_sequencer" ../uvm_tb/virtual_sequen
-- Compiling package dma_vseq_pkg
-- Importing package mtiUvm.uvm_pkg (uvm-1.1d Built-in)
** Note: (vlog-2286) Using implicit +incdir+D:/questasim/uvm-1.1d/.
-- Importing package agent_pkg
-- Importing package sequence_pkg
```

图 5.3 测试用例编译结果

Figure 5.3 Test case compilation results

```
Top level modules:
        req_ack_ctrl
End time: 14:18:49 on Apr 03,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
vlog ../uvm_tb/tb/tb_top.sv
QuestaSim vlog 10.4c Compiler 2015.07 Jul 20 2015
Start time: 14:18:49 on Apr 03,2021
vlog ../uvm_tb/tb/tb_top.sv
** Note: (vlog-2286) Using implicit +incdir+D:/questasim/uv
-- Compiling package tb_top_sv_unit
-- Importing package mtiUvm.uvm_pkg (uvm-1.1d Built-in)
-- Importing package test_pkg
-- Importing package env_pkg
-- Importing package agent_pkg
-- Importing package sequence_pkg
-- Importing package dma_vseq_pkg
-- Compiling module tb_dma

Top level modules:
        tb_dma
End time: 14:18:51 on Apr 03,2021, Elapsed time: 0:00:02
Errors: 0, Warnings: 0
```

图 5.4 顶层模块编译结果

Figure 5.4 Compilation result of top-level module

5.3.2 测试用例测试结果

根据编写的验证计划在表 5.1 中已经列举了大部分测试用例，通过使用+UVM_TESTNAME=case_name 的方式指定运行哪个测试用例，如图 5.5 所示，显示的是测试用例 ch0 的仿真。经检验，设计的测试用例都顺利通过，通过率达到 100%。如图 5.6 所示，显示在仿真过程中一些打印信息，便于查错。与在命令行中手动输入相比，通过使用脚本化管理，如图 5.7 所示，一个 make 命令就可以运行所有测试用例，提高了工程自动化程度。在验证平台搭建完后，接下来的工作就是编写测试激励了。测试用例的数据生成使用了带约束的随机激励，与传统的编写定向测试激励相比，不仅能节省编写激励的时间，而且还能随机出一些错误激励。

```
# vsim -novopt tb_dma "+UVM_TESTNAME=ch0_case" -wlf ch0.wlf -do "log -r /*; coverag
# Start time: 17:11:37 on Apr 03,2021
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is i
#
# //  Questa Sim
# //  Version 10.4c win32 Jul 20 2015
# //
# //  Copyright 1991-2015 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# //  WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# //  LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //  THIS DOCUMENT CONTAINS TRADE SECRETS AND COMMERCIAL OR FINANCIAL
# //  INFORMATION THAT ARE PRIVILEGED, CONFIDENTIAL, AND EXEMPT FROM
# //  DISCLOSURE UNDER THE FREEDOM OF INFORMATION ACT, 5 U.S.C. SECTION 552.
# //  FURTHERMORE, THIS INFORMATION IS PROHIBITED FROM DISCLOSURE UNDER
# //  THE TRADE SECRETS ACT, 18 U.S.C. SECTION 1905.
# //
```

图 5.5 测试用例仿真图

Figure 5.5 Test case simulation diagram

```
# UVM_INFO ../uvm_tb/env/my_adapter.sv(8) @ 0: reporter [my
# my_scoreboard's rootpath is uvm_test_top.env.scb .
# UVM_INFO ../uvm_tb/env/my_scoreboard.sv(33) @ 0: uvm_test
# UVM_INFO ../uvm_tb/env/ref_model1.sv(43) @ 0: uvm_test_tc
# UVM_INFO ../uvm_tb/env/my_adapter.sv(14) @ 0: reporter [m
# tr.hwdata[0]=64
# UVM_INFO ../uvm_tb/agent/my_driver.sv(64) @ 0: uvm_test_t
# haddr = 5c
# hwdata = 64
# hburst = 0
# UVM_INFO ../uvm_tb/agent/my_driver.sv(81) @ 16000: uvm_te
# UVM_INFO ../uvm_tb/env/my_cov.sv(111) @ 16000: uvm_test_t
# addr=5c
# data=64
# UVM_INFO ../uvm_tb/env/my_adapter.sv(14) @ 16000: reporte
# tr.hwdata[0]=38
# UVM_INFO ../uvm_tb/agent/my_driver.sv(64) @ 16000: uvm_te
# haddr = 60
# hwdata = 38
# hburst = 0
# UVM_INFO ../uvm_tb/agent/my_driver.sv(81) @ 36000: uvm_te
# UVM_INFO ../uvm_tb/env/my_cov.sv(111) @ 36000: uvm_test_t
```

图 5.6 测试仿真数据

Figure 5.6 Test simulation data

```
#all: work rtl agent env1 tb
all: work rtl agent env1 test tb
work:
    vlib work

rtl:
    vlog -f filelist_rtl +cover=sbf
agent:
    vlog +incdir+../uvm_tb/agent ../uvm_tb/agent/agen

env1:
    vlog +incdir+../uvm_tb/env ../uvm_tb/env/env_pkg.
    vlog +incdir+../uvm_tb/sequences ../uvm_tb/sequen
    vlog +incdir+../uvm_tb/virtual_sequencer ../uvm_t

test:
    vlog +incdir+../uvm_tb/tests ../uvm_tb/tests/test
tb:
    vlog ../uvm_tb/tb/ahb.sv
    vlog ../uvm_tb/tb/my_if.sv
    vlog ../uvm_tb/tb/tb_if.sv
    vlog ../uvm_tb/tb/req_ack_ctrl.sv
    vlog ../uvm_tb/tb/tb_top.sv
wave1:
```

图 5.7 makefile 脚本

Figure 5.7 Makefile script

5.3.3 DMA 功能测试结果

在对该通用 DMA 进行仿真验证时，首先要做的就是对待测模块内部的寄存器进行配置来启动该 DMA，然后 DMA 会根据所写入寄存器的数据信息来进行相应的操作，按照预期的目的进行数据搬移，根据与外部设备的交互进行对应的处理。仿真验证期间除了查看一些输出的打印信息来判断验证过程是否正确，其仿真过程中的仿真波形也是验证工程师必须要用到的一个判断依据，从仿真波形中可以查看到各种信号信息，是模拟待测模块在真实环境下的反应。图 5.8 所示就是通过总线对 DMA 内部的寄存器进行配置，从图中可以看到在时钟信号下，待写入数据根据 AHB 总线协议按照控制信号和写入地址将相应的数据写入到目的寄存器，当写入相应寄存器数据正确后，DMA 就开始工作，进行数据搬移。

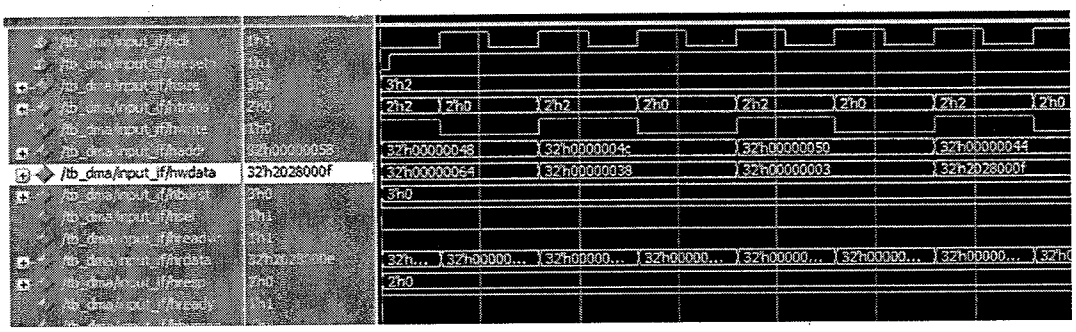

图 5.8 DMA 寄存器写入配置

Figure 5.8 DMA register write configuration

如图 5.9 所示，DMA 内部相应寄存器配置完成后，该 DMA 就被启动进行数据搬移。从图 5.9 可以看出，DMA 将数据为 32’h64 的数据读出后，暂存到 DMA 内部的 fifo 中，在下一个时钟周期就将该数据搬移到目的地址，具体什么时间写入还需要根据控制信号来判断，一般是一个至多个时钟周期。

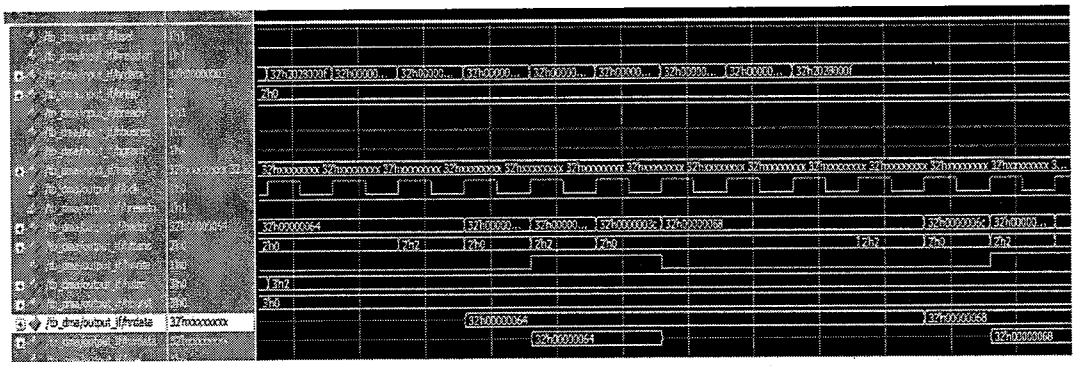

图 5.9 DMA 进行数据搬移

Figure 5.9 DMA for data transfer

硬件握手是通用 DMA 的一个功能，总共有 16 组 req/ack 信号。若使用握手协议，使用规则如下，首先从设备 slave 给出 req 请求，即拉高相应信号，然后，DMA 控制器检查内部 fifo 是否还有空间，若有，则进行传输后拉高ack信号，当从设备检测到 ack 信号拉高，则说明数据传输成功，然后拉低 req 信号，相应的 ack 信号也拉低，一次握手传输完成，如图 5.10 所示，在图中 DMA 的 req 和 ack 正如上述规则通信。

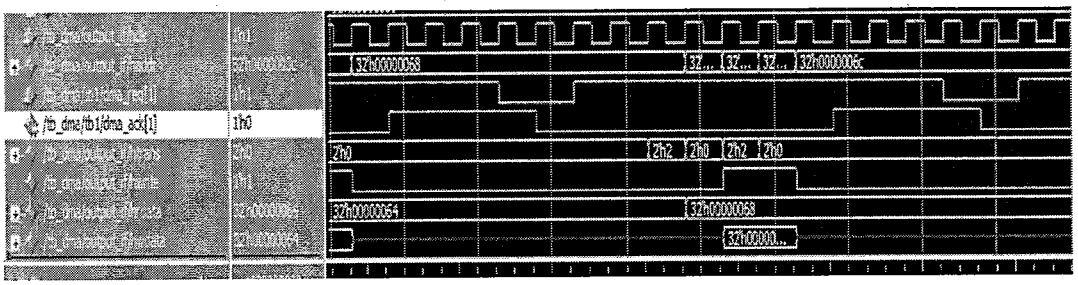

图 5.10 硬件握手检测

Figure 5.10 Hardware handshake detection

本文的通用 DMA 的链式传输是根据当前通道的链式寄存器内存储的地址，自动从外部 memory 中读取下一组通道配置，然后执行此配置信息，如图 5.11 所示，在黄色标志线之后 hrdata 信号显示自动读取了五个数据。

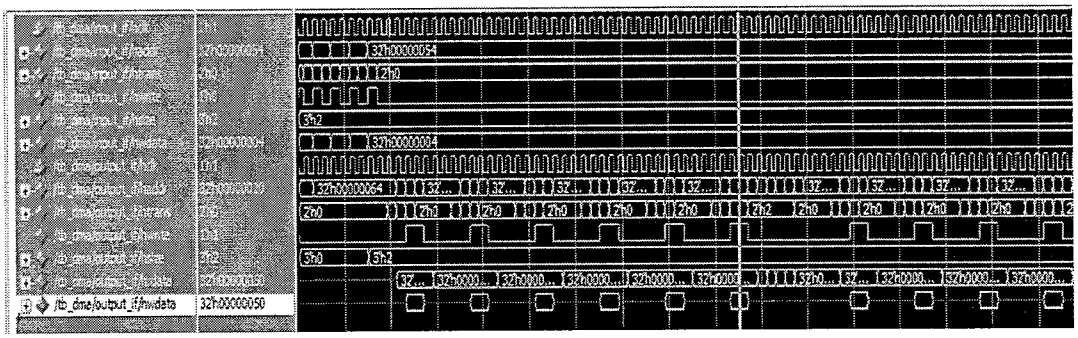

图 5.11 链式传输检测

Figure 5.11 Chain transmission detection

5.4 覆盖率统计

在验证工作中，覆盖率主要分为两个部分，分别为代码覆盖率和功能覆盖率。代码覆盖率可以查看哪些代码被覆盖到，这样可以检查被测设计是否有冗余代码；而功能覆盖率则是由验证工作者自行设计，然后编写验证组件来进行功能覆盖率统计。

5.4.1 代码覆盖率

代码覆盖率主要是查看 RTL 代码仿真情况，本文针对通用 DMA 模块的设计代码，使用 QuestaSim 仿真软件来进行仿真，在仿真过程中收集了待测模块的代码覆盖率，用来查看代码是否有冗余或者设计的验证计划不完备，这样可以当做是否增加测试用例的一个指标。查看总体代码覆盖率为 99.8%，如图 5.12 所

示，主要是由语句覆盖率和分支覆盖率这两部分组成。

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch
TOTAL	99.80%	99.91%	99.68%
tb1	99.80%	99.91%	99.68%

图 5.12 总体代码覆盖率

Figure 5.12 Overall code coverage

如图 5.13 所示，展示了语句和分支覆盖率的更多细节，在整个仿真验证过程中收集覆盖率信息，其中，语句覆盖率为 99.91%，分支覆盖率为 99.68%，虽然没有达到 100%的覆盖，但是在功能覆盖率达到 100%的情况下，代码覆盖率已经可以说达到了当初设计的目标。

Recursive Hierarchical Coverage Details:

Total Coverage:					99.83%	99.80%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	1237	1236	1	1	99.91%	99.91%
Branches	628	626	2	1	99.68%	99.68%

图 5.13 语句与分支覆盖率

Figure 5.13 Statement and branch coverage

因为本通用 DMA 由六个子模块组合而成，每个模块都占六分之一比例，在计算总的代码覆盖率时，每个子模块都会占据六分之一的份额，然后汇总后计算而出总的语句覆盖率和分支覆盖率。如图 5.14 所示，显示的六个子模块各自的代码覆盖率信息，其中 AHBMST 和 CHMUX 两个模块覆盖率未达到 100%，原因是一些语句未覆盖到。就如 CHMUX 模块而言，如图 5.15 所示，是软件复位信号未达到而使得代码覆盖率没达到 100%，在模拟真实的软件复位时，可以在 FPGA 验证时设置软件复位，达到应有的结果。

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch
TOTAL	99.80%	99.91%	99.68%
atcdmac100_engine	100.00%	100.00%	100.00%
atcdmac100_ahbmst	97.22%	100.00%	94.44%
atcdmac100_ahbslv	100.00%	100.00%	100.00%
atcdmac100_fifo	100.00%	100.00%	100.00%
atcdmac100_chmux	99.70%	99.77%	99.63%
atcdmac100_arbiter	100.00%	100.00%	100.00%

图 5.14 DMA 各个模块的覆盖率

Figure 5.14 Coverage rate of each module of DMA

if (~hresetn) begin			75.00%
Branch	**Source**	**Hits**	**Status**
IF	if (~hresetn) begin	1	Covered
TRUE	else if (dma_soft_reset) begin	0	ZERO
TRUE	else if (arb_en & ~arb_end) begin	1414	Covered
ALL FALSE	if (~hresetn) begin	8586	Covered

图 5.15 chmux 子模块软件复位语句

Figure 5.15 Chmux submodule software reset statement

5.4.2 功能覆盖率

功能覆盖率根据验证计划编写相关的覆盖率组，然后在整个验证平台搭建完后进行仿真验证时由仿真工具收集而来。

对于本通用 DMA 来说，根据 DMA 的功能，设计了以下的的覆盖点，分别为传输大小 SIZES、源地址类型 SRCBURST、源地址位宽 SRCWIDTH、目的地址 DSTWIDTH、源地址控制寄存器 SRCCTRL、目的址控制寄存器 DATCTRL、通道使能和优先级这几个覆盖点。如图 5.16 所示，从图中可以看出覆盖组覆盖率达到 100%，其中每个覆盖点都可以达到 100%，满足了功能验证的要求。

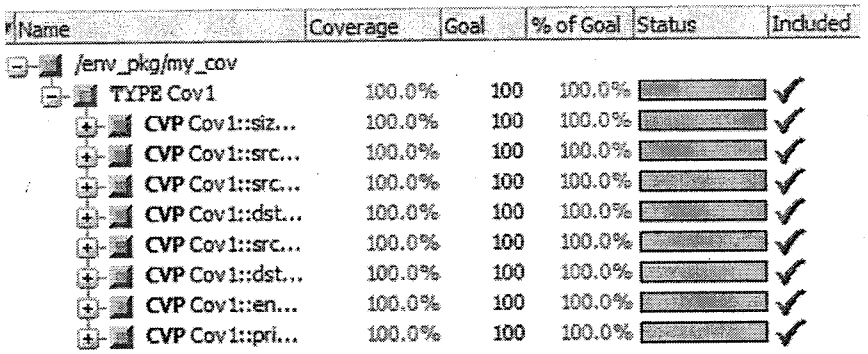

图 5.16 功能覆盖率结果

Figure 5.16 Functional coverage results

如图 5.17 所示，对于传输大小 SIZES 覆盖点来说，对于创建的不同的仓都覆盖到了，可以以此来检验传输不同大小的数据时功能都能正确。

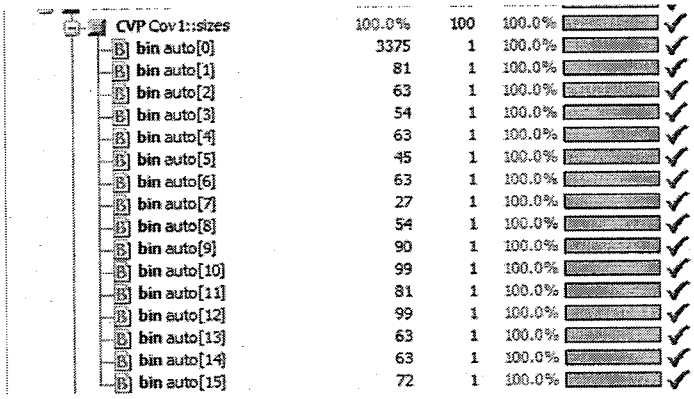

图 5.17 传输大小 sizes 覆盖点结果

Figure 5.17 Transfer size sizes coverage point result

5.5 本章小结

本章主要是从验证计划和覆盖率统计这两个方面来介绍，进而对整个 DMA 模块验证结果进行分析总结。在介绍覆盖率之前，先介绍了验证计划流程和所需要的测试用例；然后着重介绍了代码覆盖率和功能覆盖率，通过分析仿真文件、波形图和覆盖率统计结果，以覆盖率为驱动以达到验证仿真效果，得到预期目标。

第 6 章 总结与展望

本章主要是对论文所做工作和创新点进行总结和概括，通过查看代码覆盖率与功能覆盖率，分别为 100%和 99.8%,达到验证目的，最后提出一些不足与改进，为今后工作提供了方向。

6.1 工作总结

随着集成电路工艺和规模的发展，对于传统验证平台不可重用和验证效率低等问题，本文使用 UVM 验证方法学，针对 DMA 模块搭建验证平台进行验证，详细介绍了各个组件及其连接关系，主要工作总结如下所示：

(1) 研究了 UVM 验证平台，针对在仿真过程中出现错误不能及时定位的缺陷，设计了使用断言分析机制来解决。加入断言模块可以减少人为检查时间，快速定位设计错误点，使对验证结果的检查更加自动化。

(2) 研究了 DMA 的功能和结构特点，其作用是取代 CPU 进行数据搬移，针对在数据比对方面消耗仿真时间长和需要精确参考模型等问题，设计了使用 config_db 直接取出数据进行对比的方法。通过使用 config_db 机制实时的将从设备中 DMA 搬移的数据取出后传递给数据对比模块进行比对，这样不仅能保证数据正确性，避免编写复杂的参考模型，而且还能减少仿真时间。

(3) 根据验证平台与 DMA 内部寄存器交互复杂的缺陷，设计了寄存器模型方法。通过使用寄存器模型，使用前后门访问，可以对 DMA 的寄存器进行读写操作，方便快捷。

(4) 研究了 DMA 功能验证的特点，提出了测试计划，设计验证组件，搭建了整个验证平台，编写测试向量，通过查看代码覆盖率与功能覆盖率，分别为 100%和 99.8%,达到验证目的。

基于以上工作，本论文设计的 DMA 验证平台不仅可以减少验证周期，而且还可以实现较强的可重用，一方面是可用于具有 AHB 接口的模块验证，修改一些参数也可用于其他接口协议的模块验证；另一方面提出的数据比对思想可用于验证其他 DMA 模块的验证，也可对其他需要数据比对的模块提供借鉴思想。

6.2 不足与展望

本论文虽然已实现了对该通用 DMA 模块的验证，但是还存在一些不足和改进的地方。将来工作可以从以下几个方面改进：

（1） 虽然验证平台是以功能验证为驱动，但是未曾对代码覆盖率与功能覆盖率实现实时监测，不能及时发现覆盖率已满足从而终止仿真，这样会造成仿真时间的浪费。

（2） 验证平台还是手动编写，验证平台的搭建没有使用脚本生成，手动编写会延长项目时间，自动化程度不高。

（3） 研究 UVM 源码，使用一些高级功能，例如 callback 机制和 factory 机制重载等。

从这些地方出发，可以进一步优化该验证平台，并且将这些技术和方法运用到今后的学习与工作中。