分类号 TN47 学号 GS13062402

UDC 密级 公 开

工程硕士学位论文

基于 UVM 方法的 DMA 功能验证及分析

硕士生姓名 丁一博

工 程 领 域 软件工程

研 究 方 向 微处理器设计

指 导 教 师 孙永节 研究员

马胜 助理研究员

国防科学技术大学研究生院

二〇一六年三月

基于 UVM 方法的 DMA 功能验证及分析

国防科学技术大学研究生院

The Function Verification and Analysis of DMA Based on UVM Method

Candidate：Ding Yibo

Advisor：Prof. Sun Yongjie

A thesis

Submitted in partial fulfillment of the requirements

for the professional degree of Master of Engineering

in Software Engineering

Graduate School of National University of Defense Technology

Changsha，Hunan，P.R.China

March，2016

独 创 性 声 明

本人声明所呈交的学位论文是我本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的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志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谢意。

学位论文题目：基于UVM方法的DMA功能验证及分析

学位论文作者签名：丁博　　日期：2016 年 4 月 1 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于UVM方法的DMA功能验证及分析

学位论文作者签名：丁博　　日期：2016 年 4 月 1 日

作者指导教师签名：　　日期：2016 年 4 月 1 日

目 录

表 目 录

图 目 录

摘 要

伴随着 IC 技术的高速发展以及数字通信等领域对复杂数字系统的需求，芯片的集成规模以及功能复杂度呈现出指数型增长，与此同时验证工作也成为了整个芯片设计过程中的重难点环节。如何保证验证的完备性、提高验证的效率已经成为决定芯片成败的关键性问题。

本文验证的对象来自于我校自主研发设计的一款高性能 M-DSP 项目，针对 DSP 芯片中的 DMA 部件进行模块级功能验证。考虑到本文 DMA 部件传输方式的复杂性和多样性，提出基于 UVM 验证方法学的建模思想构建 DMA 验证环境。本文对 UVM 验证平台进行建模时，首先根据 DMA 的传输特性和设计规范规划出验证平台的整体结构图，再按照结构图的功能顺序对各个组件分别进行建模。UVM 验证平台构建完成之后，需要根据 DMA 的功能点对平台进行调试以保证验证环境的正确性。为了确保验证结果的完备性，本文以功能覆盖率和代码覆盖率作为判断依据，综合使用随机激励和定向激励完成整个验证流程。经过平台的不断测试，完成验证计划中所有功能点的验证，最终功能覆盖率达到 100%，代码覆盖率达到 98%，覆盖率收集满足了功能验证的需求指标。

根据 UVM 验证方法学的特性，构建出高效的 DMA 验证环境是研究的核心部分，利用此验证环境完成 DMA 部件的功能验证是整个研究的终极目标。验证环境中输入激励的实现、黄金模型的构造以及比对策略的安排是平台构建时需要仔细研究的重点，同时也是本文创新点的体现。

UVM 方法学的应用相对于传统验证节约了一半的验证时间，在提升了整个功能验证进度的同时完成了预期目标。此外，平台中各功能组件具有很好的可重用性，对其他部件验证环境开发也具有指导意义。

主题词：DMA； 功能验证； UVM； 验证环境； 覆盖率

ABSTRACT

Along with the high-speed development of IC technology and the requirement of the complex digital system in the field of digital communication, the scale of integration and the functional complexity of the chip show exponential growth. At the same time, the verification work has become an important and difficult part of the whole chip design process. How to ensure the completeness of the verification and improve the efficiency of verification has become key issues to determine the success or failure of the chip.

Verification the object in this paper comes from a high performance M-DSP project which designed by our school independently. We verify the module level function of DMA component. Considering the complexity and diversity of DMA transmission modes, this paper builds the DMA verification environment based on the modeling idea of UVM verification methodology to improve verification efficiency. When UVM verification platform was constructed in this paper, we planned the overall structure of the verification platform according to the transmission characteristics of DMA and the design specification. Then the various components are modeled respectively follow the order of the function with structure diagram. After the UVM verification platform is completed, we need to debug the platform according to the requirements of DMA to guarantee the correctness of the verification environment. We used the function coverage and code coverage as judgment to ensure the completeness of the verification result. Comprehensive use of random excitation and directional incentive to complete the verification process. After continuous testing, final function coverage reached 100%, code coverage reached 98%, coverage collection to meet the needs of the functional verification.

Building a highly efficient DMA verification environment with the UVM methodology is the core part of the study. Using this verification environment to complete the functional verification of DMA parts is the ultimate goal of the whole research. In the verification environment, the construction of the gold model and the arrangement of the comparison strategy are the key points of the platform construction, which is also the embodiment of the innovation of this paper.

Compared with the traditional verification methodology, UVM methods has saved half of the time for verification. In enhancing the function verification progress while achieving the expected objectives. In addition, the each functional components in the platform have good reusability, also have guiding significance for other parts of the verification environment development.

Key Words：DMA, Functional verification, UVM, Verification environment, Coverage rate

第一章　绪论

1.1　课题背景

集成电路设计规模的爆炸式增长以及深亚微米工艺技术的日趋成熟[1]，使得整个集成电路行业迅速发展崛起[2]，集成电路的应用也广泛延伸到各行各业并成为其中不可替代的关键部分。为了满足市场对芯片产品日益苛刻的需求，如高性能、低功耗、多功能等特点[3]，将各种功能不同的 IP 核单元以及系统关键部件集成到一个芯片上的片上系统（SOC）成为目前主流的设计方向[4]。伴随着 SOC 的功能复杂性和 IP 核单元集成度的持续提升，保证芯片系统功能的正确性和可靠性的验证工作复杂程度也随之成倍的增加，在项目开发过程中验证工作的地位和重要性也愈发的突出。尽管验证方法和技术也在不断的发展和革新，验证工作在整个项目工作中所占比例仍然高达 70%以上[5]。验证的目的在于发现设计中隐藏的错误并保证系统芯片功能符合设计规范，据统计因验证不完备充分而导致芯片一次性流片失败的概率高达 75%以上[6]。随着市场上芯片项目规模的增大、项目时间的紧缩，如何在保证验证完备充分的前提下缩短验证所需的时间已成为验证发展的主要方向。

传统的验证手段已经跟不上设计规模的发展，实际上，通过同一种验证手段解决规模不断增大的设计问题是不可能的。任何技术的发展都有其局限性，尽管基于 Verilog 语言的验证手段已经非常成熟，但是面对日益复杂的芯片系统，这种方法在很多方面都存在难以突破的桎梏[7]：

1. 基于 Verilog 语言构建的验证环境代码量大。特别是对于功能复杂的 SOC 芯片来说，验证环境的开发和维护的难度都非常大的，极大地影响了验证工作的效率。
2. 无法完全模拟出待测设计（DUT）复杂的工作行为。传统验证激励[8]是具有指向性的，更多的关注于特定的功能点，对于愈加复杂的 DUT 行为是难以通过这种激励来进行测试的。
3. 无法保证验证的完备性。传统设计的功能验证是通过具有指向性的激励来驱动的，验证是否充分很大程度取决于个人经验，显然这种取决于主观因素的情况容易影响验证的完备性。

为了解决当前验证手段无法满足设计发展需求的困境，新的验证方法不断被开发引进[9]，其中以 SystemVerilog 语言为基础形成了一整套验证方法学思想，并迅速发展成为主流的验证方法。主流验证方法学是基于抽象的高级验证语言对平台进行建模的，很好地解决了验证环境构建复杂、难以管理的问题。同时，上述

方法学还提供了随机测试激励和功能覆盖率收集等强大的工作机制[10][11]，极大地方便了验证平台的构建，同时也保证了验证的完备性。

1.2 国内外研究现状

伴随着 IC 设计业的发展变迁，验证工作在整个芯片设计项目中的地位愈发突出，甚至已经成为影响芯片成败的关键环节。验证地位的重要性使得其得到了整个集成电路行业的高度重视，为了使验证工作能够跟上设计规模不断增加的步伐，并保证设计符合功能规范，业界提出了很多行之有效的手段来提高验证的进度和排错率。

验证技术的发展推进基本上是由各大 EDA 工具厂商主导进行的，伴随着各大公司对验证思想和方法手段的探索研究，它们推出了一系列用于验证的语言和方法学。用于验证的语言主要有：Vera、SystemC 以及 SystemVerilog 等[12]，基于这些验证语言各大厂商分别研究推出了自己的方法学，其中典型的验证方法学包括：eRAM、RVM、AVM、UVM 等。尤其值得关注的是，以 SystemVerilog 语言为工具开发形成的三大验证方法学：VMM、OVM、UVM[13]，一经推出后就受到了各大芯片厂商和验证人员的欢迎。三大验证方法学通过 SystemVerilog 语言构建基本功能组件，以事务级建模完成组件之间的相互连接来进行 SOC 功能验证环境的构建。为保证验证环境构建的高效性和可重用性[14]，三大验证方法学根据各自的特点提供了不同功能的基本组件库和丰富的工作机制。SystemVerilog 语言的开发和拓展给验证方法带来了新的活力，同时也指明了以后验证的发展方向[15]。

验证方法学的形成和发展基本上都是由国外厂商和组织来主导研究并推出的，国内更重视的是对方法学的应用。目前国内的大部分 IC 设计公司对验证的工作越来越重视，因此市面上基于 SystemVerilog 语言的三种方法学已得到了广泛的应用和发展。

1.3 课题研究内容

课题的主要工作是对本校正在设计的一款高性能[16]DSP 中的 DMA 部件进行模块级功能验证。为了满足芯片高性能浮点计算的设计要求，提高数据搬移效率，针对控制 DSP 内核存储和外部存储之间数据传输的 DMA 部件设计了多种复杂的传输方式。通过分析目前主流的验证手段和思想，本课题决定采用基于 UVM 思想的通用验证环境架构来进行 DMA 验证平台的构造。为了保证 DMA 各种复杂传输方式功能的正确性，课题以收集的覆盖率为指标来调节激励生成的约束条件，保证验证的完备性。

UVM 方法学的研究是验证环境建模的思想基础，在对平台进行构建时需要充分理解 UVM 库中各个组件类的功能特点，掌握方法学中提供的各种工作机制的原理以及使用方法。其次，在对 DMA 进行功能验证之前，首先需要熟悉 DMA 部件的整体结构和功能，特别是对于 DMA 的特殊传输方式，需要了解清楚其数据搬移的整个流程和传输特点，并根据 DMA 的设计需求制定出详细的验证计划。然后，根据 UVM 基本架构和 DMA 的工作原理，按照 UVM 树形结构自下而上完成验证环境的构建，并根据提炼的功能点完成平台的调试保证环境的可靠性。最后使用调试完成的可靠环境根据验证计划对 DMA 进行大规模仿真验证，保证覆盖率的收集达到预期标准并分析 DMA 功能验证结果。验证环境的建模以及调试是课题研究的重点，完成 DMA 功能验证是课题的最终目标，这三部分也是最体现本文工程量的地方。

1.4 文章组织结构

本文总共分为六个章节来对研究的内容进行阐述，各个章节大致内容如下：

第一章：绪论。本章首先介绍了目前 IC 行业设计与验证发展不协调的大背景，指明了验证在整个芯片设计过程中的重要地位，然后就国内外验证方法学的发展和应用现状进行概述，并对课题要研究的内容进行了简要的说明。

第二章：UVM 验证方法学。本章首先对验证的发展以及方法学的形成进行了详细的概述，其次着重介绍了 UVM 平台的整体组织结构、各组件的功能和工作机制。

第三章：M-DSP 中 DMA 功能及验证分析。首先对 DMA 的整体结构和基本功能进行了阐述，并进一步分析了 DMA 四种工作模式的特点，根据 DMA 的功能特点和设计规范制定了 DMA 的验证策略和计划。

第四章：UVM 平台的构建。作为本文研究的重点内容，本章对 DMA 验证环境的建模过程进行了详细的介绍，验证环境按照结构主要分为三个部分：UVM 组件构造、黄金模型建模以及比对策略的实现，然后分别就三个部分说明了各自的功能以及代码实现过程。

第五章：验证结果及覆盖率。利用 UVM 验证环境完成对 DMA 的功能验证是研究的最终目标，本章主要对验证环境仿真生成的数据报告和波形文件进行了分析，说明了平台工作的基本流程，然后结合收集的覆盖率报告分析了 DMA 验证的完备性，

第六章：结束语。对课题的研究内容进行了总结，并对后续研究制定了新的目标。

第二章　UVM 验证方法学

随着芯片设计规模和复杂度的提升，验证的语言和手段也在不断的发展和更新，已经逐渐形成了一系列的验证方法理论。本章就目前市面上已经成熟的验证方法进行了简要介绍，然后对本文研究的理论基础 UVM 方法学进行着重的分析说明。

2.1　验证方法学概述

2.1.1 验证的含义

验证是起源于设计、服务于设计的[17]，为了保证芯片的功能设计符合规范要求，在前端设计的各个阶段都需要验证来保驾护航。在现代芯片前端设计当中，项目的基本设计流程通常是从产品需求说明开始的[18]，其前端设计的大概流程如图 2.1 所示。

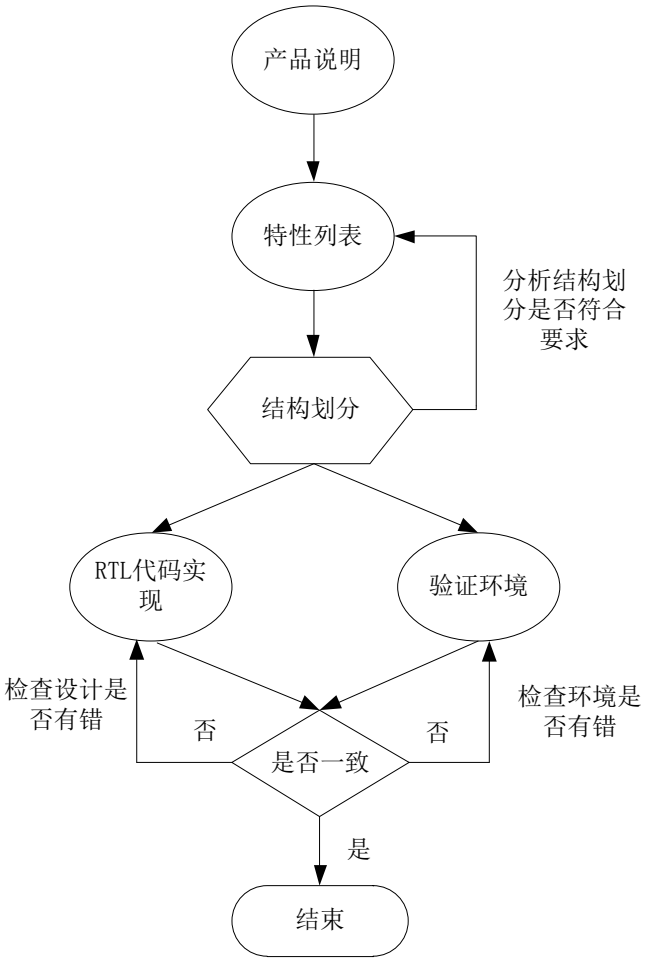

图 2.1 前端设计基本流程

工程师拿到产品的功能需求说明后会将其细化成特性列表，再根据描述的特性制定出设计方案并划分各个功能模块的结构图。整个设计的架构规划完成之后，设计人员会根据设计规范用 Verilog 语言实现其 RTL 代码，与此同时，验证人员开始用验证语言构建针对设计模块的验证环境。一般来说，这两个步骤是同时开始的，设计人员和验证人员分别根据自己对设计规范的理解开始任务。RTL 代码设计完成之后，验证人员根据环境对代码进行验证，若是验证的结果和设计规范一致则说明前端代码设计正确，若是不一致则需要验证人员逐步分析问题出错的地方。

根据整个芯片设计阶段中验证目标的不同，验证可以简单的划分为四个类型：功能验证、网表验证、时序验证和物理验证[19]。功能验证是保证 RTL 级代码符合产品功能设计规范需求的过程，网表验证是通过形式验证的方法检查 RTL 级代码和综合的网表是否保持一致，时序验证则是针对综合后带延时的网表信息查看时序是否达到性能指标，物理验证是保证版图参数符合物理设计规范，其中功能验证是整个验证流程的基础和核心。验证在芯片设计流程中的位置说明了其地位的重要性，特别是随着功能设计复杂度的提高，验证不仅与设计并驾齐驱并且验证过程的时间开销也明显超过了设计的时间周期。为了提高验证效率，保证验证的正确性，验证语言和技术方法的研究成为了业界所讨论的热点问题。

2.1.2 验证语言介绍

随着 IC 设计业的发展，Verilog 语言由于其简单实用的优点逐渐淘汰了 VHDL 语言，在两种通用的设计语言当中占据着主导地位。由于验证是服务于设计的，针对 Verilog 设计语言，经过不断的发展主要有以下三种主流验证语言[20]：

1) Verilog：Verilog 语言主要是针对设计而制定的，其起源于 20 世纪 80 年代中期，随后根据设计需求不断的发展和更新，目前最新的标准是 2005 年推出的。最初验证是起源于设计的，考虑到设计和验证合为一体的情况，在 Verilog 语言中专门设计了用于验证的一个子集，其中最典型的语言主要有 initial、function 和 task 三种。基于 Verilog 语言构建的验证环境测试效率不高，而且激励都是固定行为的测试用例。对于不断增加的芯片规模和功能复杂度来说，这种验证语言显然是不能满足验证需求的。
2) SystemC：SystemC 的开发在一定程度上满足了复杂设计对于验证的需求，特别在于算法类的设计系统当中，SystemC 拥有不可比拟的优势。在使用 Verilog 语言实现复杂算法的代码时，可以根据 C++或者 C 来同步对算法进行建模，然后将参考模型添加到验证环境当中就可以实现验证结果的比对。由于 SystemC 是基于 C++而开发的，因此其天然的继承了

C++中用户管理内存的这一致命的缺点。

3) SystemVerilog：SystemVerilog 是一门非常优秀的验证语言，作为 Verilog 语言的补充，SystemVerilog 和 Verilog 可以互相兼容，并且学习的时间周期较短。SystemVerilog 是以主流的设计语言 Verilog 为基础的，集成了面向对象编程、线程间通信和多态线程等特性于一体，并且具有面向对象编程语言的所有优点：继承、多态和分装[21]。不仅如此，作为一门高级验证语言，SystemVerilog 还为验证提供了功能覆盖率和随机约束激励等独有的特性。相比于 SystemC 语言的验证优势，SystemVerilog 不仅提供了内存自动管理机制，还提供了一种模型搭建语言，无论是非算法类还是算法类的系统设计，SystemVerilog 语言都很好地满足了系统设计的需求。

2.1.3 方法学研究

自从 SystemVerilog 语言开发问世以来，它所具有的强大数字设计和验证功能使得其在业界得到了广泛的应用，基于 SystemVerilog 语言的验证方法学给目前复杂数字系统所面临的巨大挑战带来了新的希望和出路。回顾验证方法学的发展历程，基于 SystemVerilog 语言的三种验证方法学逐渐成为主流的验证手段，其中又以 UVM 验证方法学[22]的前景最为广阔。

图 2.2 给出了整个验证方法学的发展历程示意图。验证方法探究的前期主要是以 E 语言和 Vera 语言为主的，在 2002 年 Cadence 公司采用 E 语言构建了第一个验证库—e 可重用方法学（eRM），紧接着 2003 年 Synopsys 采用自家公司的 Vera 语言构建了可重用验证方法学库（RVM）。随着 SystemVerilog 高级验证语言的开发，在 2006 年 Mentor 公司根据 SystemVerilog 语言构建了高级验证方法学 AVM，AVM 主要采用了抽象事务层的方法进行建模的。为了适应验证方法学发展的需求，Cadence 在 2007 年将 eRM 的 E 语言改进为 SystemVerilog 语言形成了通用可重用方法学（URM）。SystemVerilog 语言在验证方面的强大功能使得其在此后的时间得到了大力的发展，逐渐形成了以 SystemVerilog 语言为基础的三大验证方法学：VMM、OVM 和 UVM 验证方法学。

（1） VMM：VMM 是 Synopsys 公司在 2006 年推出的验证方法学手册[23]，这个方法学是在将 RVM 从原来的 Vera 语言向 SystemVerilog 高级语言转化之后形成的。VMM 充分利用了 SystemVerilog 验证语言的优势，将不同的功能机制封装到库文件中方便验证人员使用，显著提高了验证环境的构建速度。

（2） OVM：OVM 是两大 EDA 工具公司 Cadence 和 Mentor 联合推出的[24]，

这种方法学在设计时以 URM 和 AVM 为基础，集成了两种验证的功能特性。与 VMM 验证方法学相比较，VMM 和 OVM 的基本思想是一样的，只是在应用时 OVM 引入了很多功能强大的机制以方便验证和环境的可重用性，然而 OVM 本身也存在着不提供寄存器管理方案这一重大缺陷。

（3） UVM：2010 年 ACCELLERA 以 OVM 方法学作为研究基础，同时还引入了 VMM 中的一些功能特性，结合两种验证方法学优秀的功能机制并对不足加以改进，推出了通用验证方法学（UVM）的一个雏形。随后 UVM 验证方法学[25]的正式推出得到了所有主流工具厂商的支持，并且被广大验证工程师所接受使用。

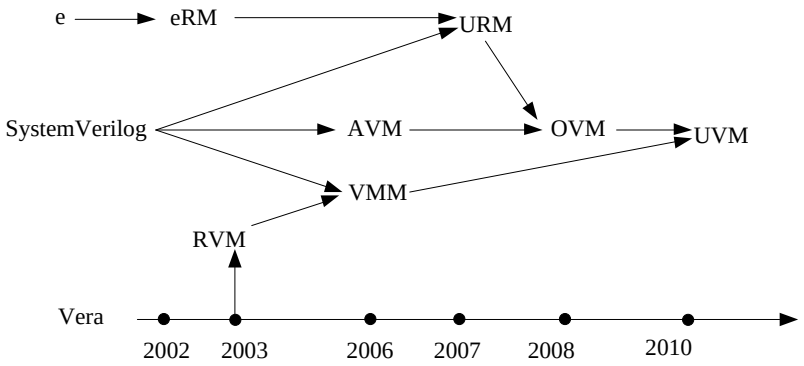

图 2.2 验证方法学发展历程

UVM 验证方法学得到各大工具厂商和 IC 设计公司的支持，说明了其良好的发展前景。UVM 方法学相比于 VMM 更多的是关注怎么样对平台进行配置、如何将可重用[26]的思想加入到验证当中，为此 UVM 集成了大量管理配置的工作机制，并使用工厂模式提升环境中各组件的重用性，这些思路和方法对平台的构建和以后验证的发展趋势都有着重大作用。

2.2 UVM 方法学介绍

2.2.1 UVM 平台架构

UVM 验证方法学作为一种指导验证的方法思想，其根本目的还是为了能够高效、准确、有序[27]的对验证环境进行建模。UVM 方法学可以看成是多个基本功能单元集合成的库文件[28]，其中每个单元都有各自特殊的功能，对不同的功能单元进行配置组合后就能构成一个基本的验证环境。图 2.3 给出了 UVM 的一个通用验证环境结构图，平台的各个组件都是从基本的库文件中拓展出来的，因此这些组件都天然继承了类中封装的基本功能。

验证环境中的基本组件都有着各自的特定功能，组件之间互相配合共同完成整个验证的工作流程。平台中环境类（env）内部是通过事务级建模（TLM）来实

现各组件之间的通信[29]，env 和 DUT 之间的通信通过接口（interface）作为桥梁实现。平台完成一次激励的测试需要经历以下几个步骤：首先，DUT 需要的测试激励是由序列生成器（sequence）负责随机产生的，生成的激励数据包交由序列发射器（sequencer）传递给驱动器（driver）进行驱动。其次，当 driver 将激励配置到 DUT 之后，DUT 根据配置的激励完成相应的行为并输出结果。一旦 DUT 的输入和输出端口有数据流通时，输入代理（i_agt）和输出代理（o_agt）中的监视器（monitor）会分别监视并保存输入和输出端口中流过的有效数据。最后，监视的输入端口数据会被送到参考模型（reference_model）当中得出预期行为的结果，由记分板（scoreboard）来完成 DUT 输出值和平台预期值之间的比较，判断 DUT 的设计行为是否正确。

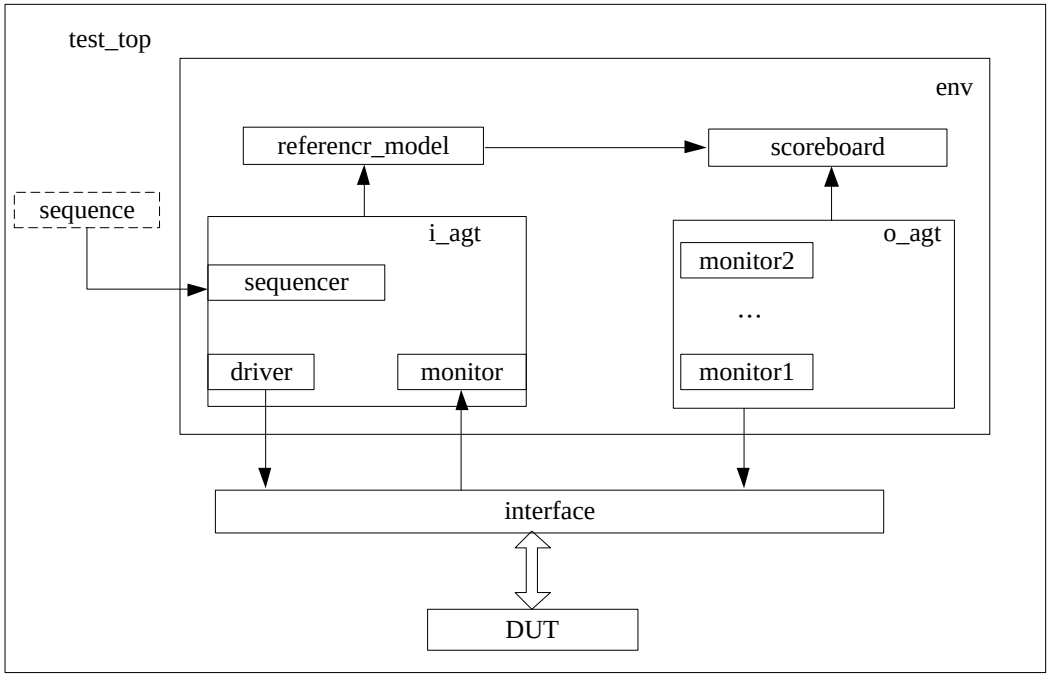

图 2.3 UVM 通用验证环境结构图

对于不同功能的待测部件来说，UVM 方法学具有非常好的通用性和可重用性，这两种特性对于提高验证平台的构建效率有很大的作用。其通用性主要体现在验证平台的基本结构和建模方法是相同的，都是根据图 2.3 所示的通用验证环境为设计基础，按照各个组件逐步完成平台的构建。由于各个组件在验证平台中的基本作用都是一致的，因此对于不同项目中类似的功能部件其验证平台和组件是可重用的。

2.2.2 UVM 组件介绍

UVM 方法学的实现采用了先分类后管理的思想[30]，平台中各个组件按照功能完成划分之后，采用树形的组织结构对各个部分统一进行管理。根据图 2.3 所

示的通用验证环境结构图，标准的 UVM 环境（env）可以划分为四个功能块：输入代理（i_agt）、参考模型（reference_model）、输出代理（o_agt）以及记分板（scoreboard）。各个功能块都是由 UVM 的基本组件构成，下面详细介绍 UVM 的基本组件及其功能：

驱动器：驱动器（driver）是环境的基本组件之一，负责给待测设计提供测试各种不同功能点的数据激励。UVM 平台中流通的是抽象的事务级数据流（transaction），驱动器需要按照待测设计的物理端口协议对其进行解析，再将激励驱动到输入接口当中。

序列发射器：序列发射器（sequencer）负责传输驱动器需要的事务级数据流，而数据流则独立在序列生成器（sequence）中生成，此外，序列发射器还对数据流起到仲裁作用。平台完成一次激励的生成到结果比对需要一定的时间，序列发射器需要对前后两次数据流进行仲裁，顺序完成数据流的发送。

监视器：监视器（monitor）主要负责对待测部件的输入和输出接口进行监视，收集接口中经过的有效数据流。监视器需要不停地按照协议对端口的数据进行收集，并抽象成事务级数据传递给其他组件。

参考模型：参考模型在平台中需要模拟出和待测部件相同的功能，并根据输入的激励输出 DUT 的预期结果。针对不同的设计，参考模型的复杂程度也不一样，在考虑参考模型的构建时，需要认真分析设计规范正确模拟出 DUT 的功能。

记分板：记分板的主要功能是比较待测部件的输出结果是否和参考模型的预期结果一致，通过二者的输出来判断待测部件的行为是否符合此次测试激励的预期行为。

代理组件：代理组件（agent）是相同事务处理机制的几种基本组件的一个集合，将功能不同但是协议相同的组件封装成高层次的代理组件符合 UVM 树形管理机制的思想。不同的代理在平台中表示的协议和功能也不同，上面结构图中的 i_agt 和 o_agt 分别代表着对输入和输出端口的处理机制。

环境组件：环境组件（env）是对 UVM 标准平台中所有组件的一个封装集合，为了方便所有功能块的统一管理以及各个组件的实例化，需要创建一个大的“容器”类来包裹所有的基本组件。

UVM 验证环境是根据上面的基本组件搭建而成的，对平台进行建模就是对各个组件进行设计实现的过程。针对不同的待测部件，各个组件的建模只需要根据其设计规范协议来完成其功能实现即可，大大的提高了平台的构建效率和可靠性。

2.2.3 UVM 工作机制

为了方便平台中各个组件功能的实现，保证验证环境正确地运行，UVM 方法

学提供了很多强大的功能机制，下面就平台构建过程中常用的功能机制做重点介绍：

一、 phase 工作机制

UVM 环境中一些组件的基本功能以及整个平台运行的管理都是由 phase 机制实现的[31]，按照 phase 机制实现的方法不同，可以将所有的 phase 分为两个大类：第一种是通过函数来实现功能的 function phase 机制，另一种是通过任务来实现功能的 task phase 机制。图 2.4 所示为 UVM 中的基本 phase 示意图，其中虚线部分表示的是 task phase 机制，实线部分为 function phase 机制。在验证环境中 phase 一般都是按照图中所示的顺序自上而下自动执行的，其中 function phase 机制在同一时间只会执行一个，task phase 机制可以在同一时间并列执行多项。

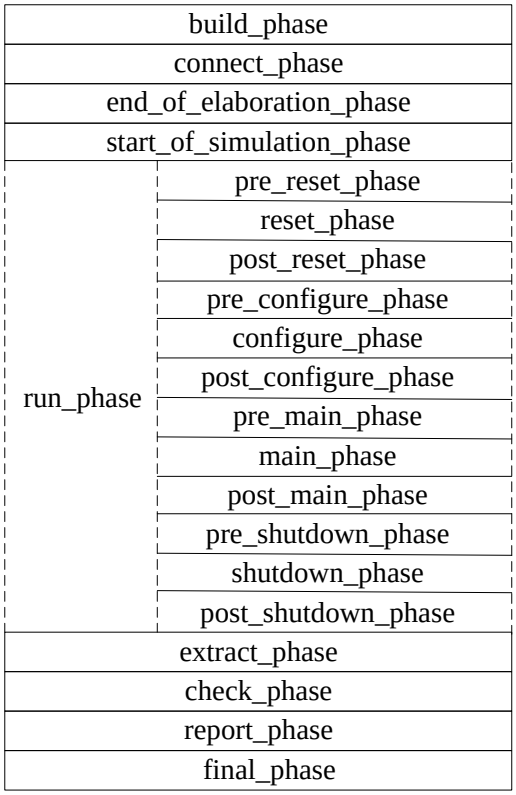

图 2.4 UVM 中 phase 机制示意图

验证平台的构建是一件极其繁琐的事情，phase 工作机制很大程度上方便了验证人员对各组件的工作步骤进行规划。UVM 提供了大量的 phase 工作机制，通过这些机制可以将整个仿真过程分为很多细致的工作步骤，方便其他的验证方法学向 UVM 进行过渡和平台的构建。例如，在各组件之间进行连接通信时，需要对组件进行实例化和连接两个操作，为了避免代码的先后顺序而产生竞争导致变量的不确定性，可以将实例化的过程放在 build_phase 中来完成，将组件之间的连接放在 connect_phase 中来执行。这两种 phase 机制的执行顺序是分开的，同一时间段内只会有一个工作机制被执行，因此 phase 机制的应用大大降低了代码顺序对

功能的影响。

UVM 中 phase 机制的设计不仅方便验证人员在对平台建模时根据各机制的作用对代码顺序进行划分，还具有自动执行其工作机制的功能，phase 机制的应用大大减少了平台开发的工作量，提高了建模效率。

二、 sequence 工作机制

UVM 中 sequence 机制[32]的主要作用是产生平台中需要的事务级激励（transaction），相比于传统验证方法中激励的提供方式，sequence 机制将平台激励的产生分为两个部分：一部分由 sequence 专门负责生成随机的事务级激励，另一部分由 sequencer 将生成的激励传递到驱动器中进行解析和驱动。这两个部分是实现 sequence 工作机制的基本组件，二者缺一不可，sequence 产生的事务只有通过 sequencer 才能导入平台进行驱动，没有 sequence 那么 sequencer 的存在也是没有意义的。

图 2.5 所示为 sequence 工作机制示意图，sequence 机制的工作协议可以分为两种情况来分析：①当 driver 向 sequencer 发送新的数据包请求时，driver 与 sequencer 之间的通道会被打开，若 sequence 与 sequencer 之间的通道中已经有等待的数据包则将其直接送给 driver，若是 sequence 与 sequencer 之间没有数据包则需要等待 sequence 产生新的数据包。②当 driver 没有向 sequencer 申请新的数据包时，二者之间的通道会被关闭，若此时 sequence 有新的数据包产生，则需要在通道里面等待 driver 的数据包申请。sequencer 在整个工作过程中起到了一个开关和通道的作用，负责控制 sequence 和 driver 之间的激励配置协议。

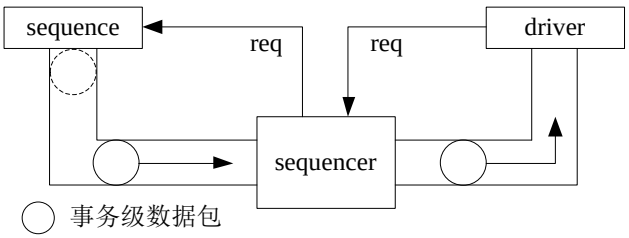

图 2.5 sequence 机制工作示意图

UVM 方法学中 sequence 机制的应用具有重大的意义，通过 sequence 机制从平台的驱动器组件中分离出激励生成这一过程，使得驱动器的功能更加的具体。例如，本课题根据 DMA 的验证计划添加激励生成的约束条件时，不需要对 driver 进行反复修改，只需在 sequence 中按照需求修改约束行为即可，有利于验证人员对随机用例的编写。同时，sequence 机制剥离激励生成也有利于平台组件的可重用性。

三、 field_automation 工作机制

UVM 环境中的各个组件之间是以事务级数据流来进行沟通的，为了方便对数据包进行一系列处理，UVM 提供了功能强大的 field_automation 机制来完成这

些操作。在 field_automation 机制中主要封装了复制、打包、比对、解包、打印等几个常用的功能函数，只需要通过`uvm_object_utils_begin 和`uvm_object_utils_end 两个宏定义对数据包中的参数进行注册之后，就可以在各组件直接对上面的功能进行调用。

除了上面三种常见的机制外，UVM 方法学还提供了其他丰富的工作机制来方便平台的构建，比如 object 机制、factory 机制、config_db 机制[33]等等。这些工作机制都具有各自强大的功能和作用，在对平台进行构建时极大地提高了建模效率，增加了平台的可重用性，充分体现了 UVM 方法学的精髓。

2.2.4 UVM 通信机制

验证环境的工作需要平台中各组件相互配合才能进行，根据上面 2.2.1 节所介绍的平台工作流程，环境中各个部分之间的通信可以划分成两个部分：第一部分是 UVM 的环境组件与 DUT 端口之间的相互通信，这部分通信是通过 SV 语言提供的虚接口（virtual interface）功能来实现的；第二部分是 UVM 环境组件之间的相互通信，这部分则是由 UVM 提供的事务级建模（TLM）来实现的。两种通信机制虽然工作的方式不同，但出发点都是为了方便验证人员维护验证环境。

1. 虚接口工作机制

为了方便对 DUT 输入端口的数据驱动，避免平台中绝对路径的使用，可以采用接口来完成两个部分之间的连接。图 2.6 所示为 UVM 环境和 DUT 之间的接口示意图，接口可以看成是两个部件之间的一个通信管道，在考虑二者之间的互连时，将所有的端口进行捆绑封装成 interface 模块来取代各个端口不同的路径。

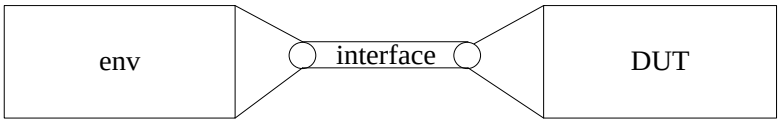

图 2.6 接口通信示意图

在使用接口来实现平台和 DUT 之间的通信时，只需要简单的在 interface 中将所有的输入端口定义成 logic 类型即可完成端口的封装，然后在验证环境的顶层通过 config_db 机制来实现平台和 DUT 之间的连接。

2. TLM 通信机制

根据 UVM 方法学便于管理、可重用强的思想，平台被划分成为了各个功能非常具体的组件，为了保证各个功能独立组件之间的协调性，UVM 提供了强大的事务级建模机制来完成组件之间的连接。图 2.7 所示为 TLM 工作机制示意图，发起者和接收者之间完成事务通信分别通过两个步骤来进行的，其中发出者通过 put 操作将 transaction 发送给接收者，接收者将 transaction 回复给发出者则是通过 get 操作来进行的。其中发出者的端口（方块表示）称为 PORT，接收者的端口（圆

圈表示）称为 EXPORT。

当发送方将收集的 transaction 向目标接收者传送时，这个操作是由 put 来完成的，此时数据的流向是从 PORT 端向 EXPORT 端传递的。当发出者向接收者申请一个数据包时，两个组件的端口依然保持不变，只是数据的流向是从右向左的，这个过程通过 get 来完成。两个组件之间的通信就是由 put 和 get 两个操作协同合作来完成，数据流在 PORT 和 EXPORT 之间相互传递，通过这种机制完成了整个 UVM 平台组件之间的数据流通。

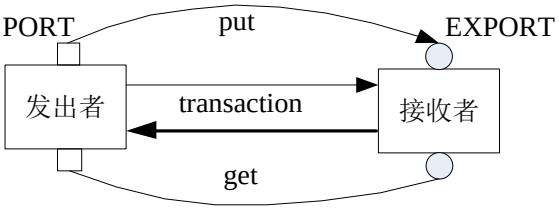

图 2.7 TLM 工作机制示意图

2.3 本章小结

本章从验证方法学的发展变迁出发，引出了 UVM 验证方法学的形成过程及背景，随后对 UVM 验证方法学进行了着重介绍。通过对 UVM 平台的整体架构和各种丰富的工作机制进行分析，充分说明了 UVM 方法学在平台的构建和可重用性上的重大优势，为本文后续构建 DMA 验证环境提供了强大的理论基础。

第三章　M-DSP 中 DMA 功能及验证分析

自从 TI 推出业界第一颗 DSP 芯片以来，经过三十多年的技术发展和市场开拓，DSP 已成为数字信息时代的核心引擎。由于 DSP 技术与大量数据运算[34]相关，DSP 的数据处理能力决定了其性能的好坏。数据处理能力与数据传输速度直接相关，因此 DSP 中负责内核存储设备和核外存储设备之间数据传输的 DMA 部件已成为高效 DSP 体系结构设计的关键。本章主要介绍了我校自主研发设计的 M-DSP 中的 DMA 部件，首先介绍 M-DSP 中 DMA 的结构和功能，然后介绍其作为主动设备发起数据搬移时四种不同的传输方式，最后介绍 DMA 的验证计划及如何进行验证。

3.1　DMA 功能介绍

3.1.1 DMA 的结构

M-DSP 芯片拥有两个 DSP 内核，每个 DSP 内核中都拥有一个独立的 DMA 控制器负责 DSP 数据的搬移[35]，DMA 在内核中的位置如图 3.1 所示。内核中可以通过 DMA 访问的存储设备有标量存储（SM）、向量存储（AM）以及调试仿真部件（ET），核外可访问的存储设备主要有本地 SubGC(Global Cache)和挂接在环网上的其他存储资源。DMA 可以在不打扰 CPU 运行的情况下，根据配置的参数字完成在核内存储设备和核外存储设备之间的数据搬移。

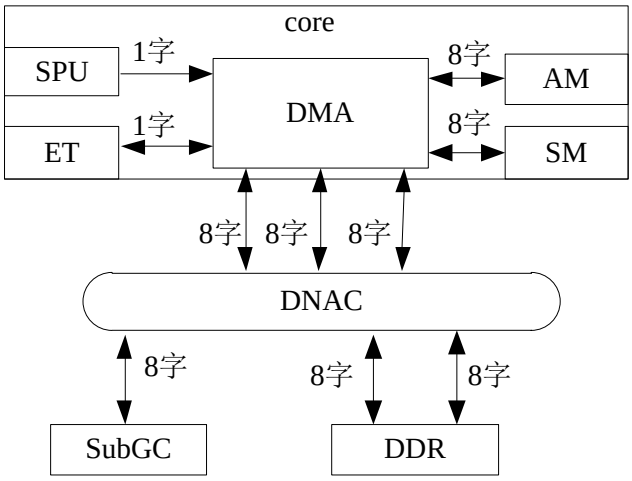

图 3.1 DMA 的核内位置

DMA 参数字是通过标量核（SPU）来进行配置的，参数字包含了 DMA 完成一次传输所有需要的信息。DMA 启动后根据配置的参数字去特定的源存储设备中读写数据然后将数据写入到特定的目的存储设备中，整个数据传输的过程包含

了读操作和写操作都是由DMA自己主导来完成的。例如，要将数据由SM搬移到本地SubGC中，首先由DMA向SM发送读请求信息，SM根据读请求信息将数据返回给DMA之后由DMA将数据写出到本地SubGC中。

DMA 数据传输是按照二维矩阵块的组织结构来进行搬移的，构成矩阵块的基本单位是由地址连续的多个字节组成的数据单元字，单元字最小数据宽度是 32 比特。如图 3.2 所示为一个二维数据组织结构图，由多个地址不断增加的数据单元字组成了一个数据阵列行，多个数据阵列行组成了一个二维数据结构，每一个数据阵列行中的数据单元总数是相等的。

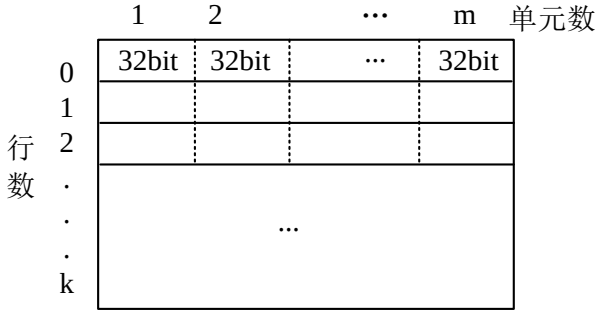

图 3.2 二维数据结构

3.1.2 DMA 的传输参数

DMA 作为主动设备完成一次数据传输任务时，其所有必须的配置信息包括源目的地址，传递的数据量、传输方式等都包含在 8 个控制参数字里面[36]。整个参数字的排列结构以及具体的说明分别如图 3.3 和表 3-1 所示，每个参数字的位宽是 32 位，不同的参数字及参数字中不同位都代表着不同的意义，它们共同组成 DMA 传输所需要的配置信息。

图 3.3 整体参数结构

表 3-1 DMA 参数字

参数名	位数	具体含义
配置参数项 0	32 位无符号数	DMA 传输模式控制字 0，单独参数字结构如图 3.4 所示
配置参数项 1	32 位无符号数	DMA 传输模式控制字 1，单独参数字结构如图 3.5 所示
源初始地址	32 位无符号数	源存储设备中数据块初始地址的低 32 位
源单元计数	16 位无符号数	源设备中要搬移的二维矩阵中每个阵列行的单元字数
源行计数	16 位无符号数	源设备数据矩阵块中包含的阵列行数目，其值为 0 时表示 1 个阵列行，值为 1 时表示 2 个阵列行，依次类推
目的初始地址	32 位无符号数	目的存储设备中数据块初始地址的低 32 位
目的单元计数	16 位无符号数	目的存储设备中要搬移的二维矩阵中每个阵列行的单元字数
目的行计数	16 位无符号数	目的数据矩阵块中包含的阵列行数目，其值为 0 时表示 1 个阵列行，值为 1 时表示 2 个阵列行，依次类推
源行偏量	16 位有符号数	源数据块中两个阵列行之间的地址偏移量
目的行偏量	16 位有符号数	目的数据块中两个阵列行之间的地址偏移量
参数链接地址	16 位无符号数	配置连接参数 RAM 地址的低 16 位
块偏量	16 位无符号数	在矩阵分块传输中，单个核接收相邻两块数据时，数据块之间的地址偏移量

1. 配置参数项 0

配置参数项 0 是所有控制参数字中结构最复杂也是最重要的参数，其中不同的位都具有其不同的意义，并且一些位的设置会成为后面参数设置的约束条件，因此在确定利用 DMA 完成何种传输任务之后首先需要设置的参数就是参数字 0。图 3.4 所示为参数 0 各个位作用域解析，下面将介绍各个位所代表的具体含义。

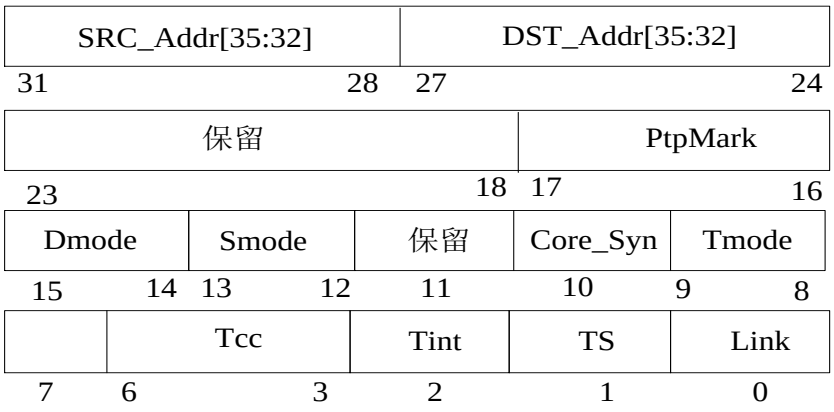

图 3.4 参数字 0 位域解析图

- SRC_Add 和 DST_Addr 分别表示源地址和目的地址的高 4 位，由于参数字的位宽限制为 32 位而对应的源目的地址位宽为 36，因此分别将其地址的高 4 位截取置于参数字 0；
- PtpMark 表示要参与 DMA 核间同步传递方式的核 ID 号；

- Dmode 表示目的地址更改方式，从图 3.2 的二维数据结构图中可以看出，一个阵列行中数据单元字的地址是连续递增的，但是阵列行与阵列行之间地址是可以有跨度的。当 Dmode 设置为“00”时，每一个阵列行的初始地址是不变的，Dmode 设置为“01”时，相邻阵列之间的地址按照一定的跨度变化；
- Smode 表示源地址更改方式，当 Smode 设置为“00”时每一个阵列行的初始地址是不变的，Smode 设置为“01”时，相邻阵列之间的地址按照一定的跨度变化；
- Core_Syn 指示核间同步传递的使能信号；
- Tmode 表示 DMA 要进行的传输方式：设置为“00”表示矩阵传输，“01”表示为转置变换传输，“10”表示为矩阵分块传输，“11”表示为矩阵多核传输；
- Tcc 指示传输结束之后的中断标记信号，从“0000”—“1111”总共 16 位分别对应着 DMA 的 16 个参数通道；
- Tint 表示 DMA 中断许可信号，置上此位时 DMA 在结束传输数据之后发出中断信号；
- TS 表示矩阵块同步许可信号，当此位为 1 时表示传输的是整个二维矩阵，为 0 时表示只是传输二维矩阵的一个阵列行；
- Link 表示参数连接使能信号，当存在参数连接时需要配置此位为 1；

2. 配置参数项 1

参数项 1 各个位的配置也是相对比较复杂的，而且需要配合着参数项 0 来使用。图 3.5 为参数字 1 的各位域的解析图，其主要的作用是在 DMA 进行某些特殊的数据传递模式时，对一些必需的配置信息进行补充，因此只有在用到这些功能的时候才会对其进行设置。

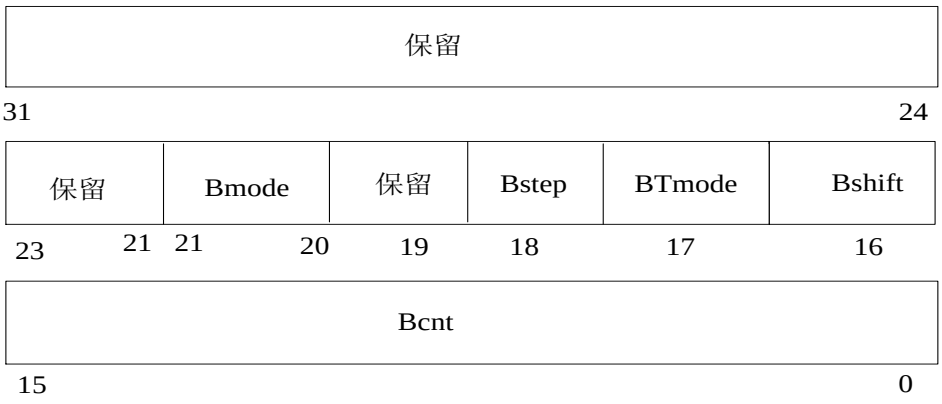

图 3.5 参数字 1 位域解析图

- Bmode 用来标记 DMA 进行矩阵分块传输时首块数据应当传输的目标核，当 Bmode 被设置为“01”的时候表示首块数据应该送到核 0，设置为

“10”时表示首块数据应该送到核 1，单块数据传输完成 Bmode 会循环进行移位；

- Bstep 表示 DMA 进行矩阵分块传输时，当完成一个单端数据的搬移后 Bmode 应该循环右移的位数；
- BTmode 表示 DMA 进行矩阵分块传输和矩阵多核传输的计数方式，设置为“0”表示 DMA 是主机计数，设置为“1”表示 DMA 是从机计数；
- Bshift 表示 DMA 进行矩阵分块传输时传输完首块数据之后需要移位的次数，移位 Bshift 次之后 Bmode 会重新回到它的初始值，由于只有两个 DSP 核，因此在进行分块传输时此位应该置为“1”；
- Bcnt 在不同的传递方式下其代表的含义是不同的，在 DMA 二维矩阵转置传输时 Bcnt 设置为“1”表示转置传输的数据宽度是 64 位，设置为“0”表示转置传输的数据宽度为 32 位；在矩阵分块传输模式下，其表示将整个数据矩阵按照 Bcnt 的大小进行分块来传输到不同的 DSP 核；

3.1.3 DMA 的启动方式

DMA 传输任务的启动方式主要有三种，第一种方式是由标量核（SPU）写“事务置位寄存器”(ESR)直接启动 DMA 进行传输；第二种方式是通过同步中断事务启动 DMA；第三种方式是通过 DMA 参数通道链接启动 DMA。

通过 SPU 配置“事务置位寄存器”直接启动 DMA 是上面三种启动方式中最常使用的。DMA 设置了 16 个参数配置通道分别对应着寄存器的 16 位，当参数配置通道中的 8 个参数完成配置之后，由 SPU 置位相应的 ESR 寄存器就可以启动 DMA 读取该参数通道中的参数进行传输任务。

DMA 通过中断事务来启动任务时，这些中断事务可以是全局的也可以由 DMA 自身的传输事务完成之后发出的中断，DMA 设置的 16 个参数通道分别对应着一个同步的中断事务。参数通道与同步中断事务的对应关系如表 3-2 所示，当参数通道配置完成并且置位相应的“事务使能寄存器”后，一旦检测到对应的同步中断标志就启动 DMA 读取该通道中的配置参数开始传输任务。

DMA 还支持参数通道链接启动，不过使用这种启动方式时需要在当前 DMA 传输事务配置参数中设置 Tcc 的值为“7~15”，同时还需要置位中断许可信号 Tcint。在当前传输任务结束后，DMA 会根据配置参数产生 Tcc 值和置位的“事务使能寄存器”启动 DMA 的参数通道 7~15。

表 3-2 DMA 参数通道和同步事件

参数通道号	同步事件
参数通道 0	DSP0 核全局中断事务
参数通道 1	DSP1 核全局中断事务
参数通道 2	定时器 0 中断
参数通道 3	定时器 1 中断
参数通道 4	保留
参数通道 5	保留
参数通道 6	GPIO 中断
参数通道 7	DMA 传输结束中断码 7
参数通道 8	DMA 传输结束中断码 8
参数通道 9	DMA 传输结束中断码 9
参数通道 10	DMA 传输结束中断码 10
参数通道 11	DMA 传输结束中断码 11
参数通道 12	DMA 传输结束中断码 12
参数通道 13	DMA 传输结束中断码 13
参数通道 14	DMA 传输结束中断码 14
参数通道 15	DMA 传输结束中断码 15

3.2 DMA 的传输方式

M-DSP 中 DMA 部件支持的数据传送方式不仅包含了二维矩阵块到块传输，还包括二维矩阵转置传输[37]、矩阵分块传输以及矩阵多核传输三种特殊的传送方式。后面三种传送方式都是基于高性能科学计算而设计的，这三种传输方式虽然显著丰富了 DMA 的传送功能，但也增加了其设计与验证的复杂性。

3.2.1 矩阵块到块传输

矩阵块到块传输是通过 DMA 作为主动设备发起的其所在核的内核存储设备与核外存储设备之间的数据搬移，两者之间的数据搬移是双向的，即既可以将核内设备的数据搬移到核外也可以将核外的数据搬移到核内。在启动 DMA 按照这种模式传输时，可以根据配置信息完成整个二维矩阵块的搬移，也可以只选择搬移矩阵块的一个数据行。图 3.6 为矩阵块到块传输的示意图，当需要搬移一个完整的矩阵时，可以通过配置目的行计数和行单元数来规定搬移之后的矩阵的形状（如图矩阵①和矩阵②）。在搬移整个二维矩阵时只需要保证传输前后两个矩阵的数据量相等，对每一个阵列行的单元字是否相等没有要求。

在这种块到块传送模式中，根据要完成数据搬移方向的不同可以将传送过程

分为两种不同的类型。一种是“读核内写核外”即把核内的数据资源搬移到核外的存储空间；另外一种是“读核外写核内”即把核外的数据资源搬移到核内设备空间。考虑到 DMA 的整体结构，在具体实现这两种类型的传送过程时所采用的机制也是不完全相同的。

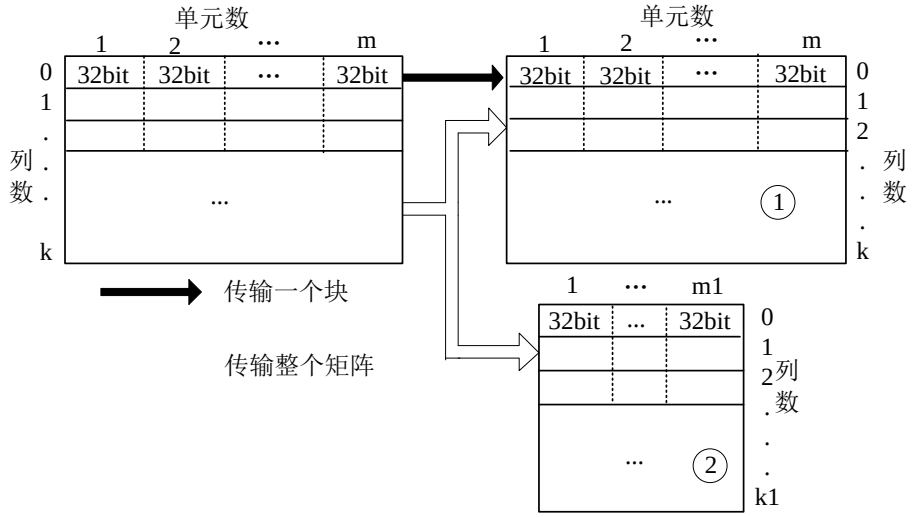

图 3.6 块到块传输示意图

在第一种读核内的传送模式下，DMA 物理通道向核内设备发起读请求等信息之后，核内设备会将读取的数据送回给物理通道，然后由其根据配置的参数计算出写请求等信息后将数据写出到核外的存储设备中。而在读核外的传送模式下，DMA 向核外的存储设备发起读请求等信息之后，核外设备返回的数据不再需要返回到 DMA 的物理通道中，而是由 DMA 的从机专用数据通路写出到目的核内存储空间中。由于这种读写机制的差异，DMA 在进行核内外存储空间之间的数据搬移时，DMA 写核内数据空间时地址是可以乱序，而写核外数据空间时地址必须是顺序的。

3.2.2 二维矩阵转置传输

二维矩阵转置传输是为了方便运算部件进行矩阵乘法运算而设计的一种特殊的数据搬移方式，DMA 在向源端存储设备发出读请求信息之后，数据会返回到 DMA 的转置矩阵存储体中，之后再实现矩阵转置的数据搬移。如图 3.7 为读数据返回给 DMA 后写入转置存储体中，再从存储体中读出数据完成矩阵转置的过程示意图。

由于 DMA 传输数据的总线位宽为 256 位，而一个最小的数据宽度是 32 位的，因此在进行矩阵转置传输时每次转置一个 8*8 大小的矩阵。DMA 从源存储设备中读出数据之后按照图 3.7 中①矩阵所示的数据写入方法，将其按照行写入到转置存储体中。当写满一个 8*8 的矩阵之后，DMA 再根据图 3.7 中②矩阵所示的

方法从存储体中按照列读出数据以此来完成一个矩阵的转置，数据读出之后再由 DMA 的从机专用通道根据写地址将数据写出到目的存储设备当中。

在进行二维矩阵转置进行数据搬移时，还可以在参数字中将最小的数据宽度配置为 64 位，也就是每次需要完成的是一个 4*4 的矩阵转置搬移，其搬移的原理同 8*8 矩阵转置搬移的原理是一样的，只是在数据读出的时候是每次读出两列的数据。

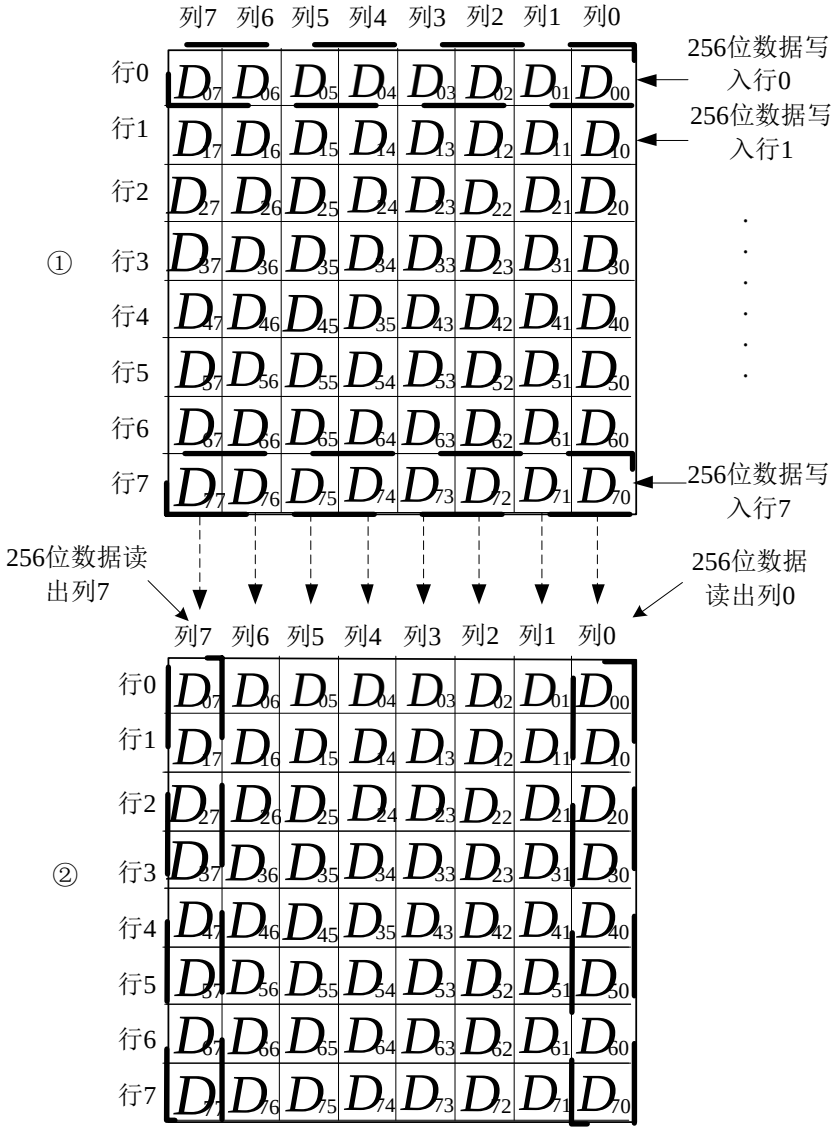

图 3.7 矩阵转置过程示意图

3.2.3 矩阵分块传输

矩阵分块传输是为了满足高性能计算而设计的一种特殊的数据搬移方式，即启动一个核的 DMA 将核外存储设备中的一大块数据矩阵分成数据量相同的小块数据矩阵然后将这些小块矩阵循环搬移到不同 DSP 核的核内存储设备中。这种数

据传送的方向是一个单向的传输，即只能将核外的数据资源搬移到核内的存储设备中，不同内核中的存储设备可以接收不同的小矩阵块。

图 3.8 是 DMA 进行矩阵分块传输搬移数据过程的分示意图。要完成搬移的核外存储设备空间中数据矩阵块是由 4 个数据阵列行组成，每个阵列行中有 48 个单元字。假设启动核 0 的 DMA 进行分块传输，并且每次要完成 48 个单元字的搬移，源数据块矩阵将会被分成相等的 4 个等份，首先将第 1 块数据搬移到核 1 的存储设备中，然后再去搬移数据块 2 到核 0 的存储空间，依次循环完成源矩阵到两个 DSP 核内存储空间的搬移。每个核会得到原来矩阵块的不同部分。如核 1 的存储空间会得到数据块 1 核数据块 3，核 0 的存储空间会得到数据块 2 和数据块 4。

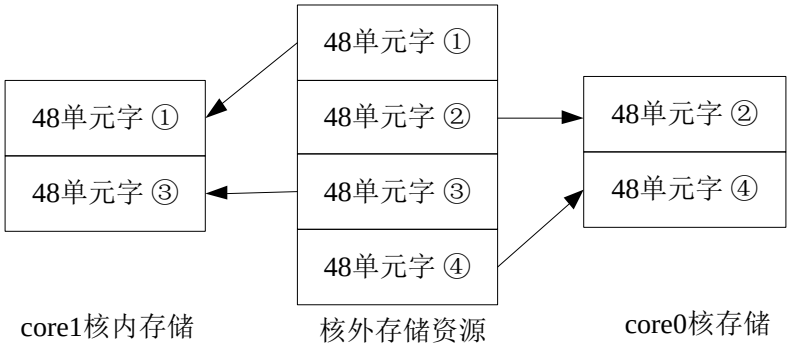

图 3.8 矩阵分块传输传输示意图

矩阵分块传输中有两种不同的计数结束机制：一种是完全由发起分块传输所在核（主核）的 DMA 对要搬移的二维矩阵进行计数，包括不是传送给发起核的数据块，当主核中的 DMA 计数完成之后会发出中断信号并给从核一个特殊的读请求，其他核接收到这个请求之后表示所有的数据全部搬移完成并以此来执行清空内缓冲操作和发出中断信号；另一种是由所有参与分块传输的核对以自己所要接收的数据块进行计数，这种情况下在启动 DMA 进行分段传输之前就需要配置好各个核所需要接收的总数据量，当各自的计数完成之后直接发出中断信号。

3.2.4 矩阵多核传输

矩阵多核传输的功能类似于矩阵分块传输，不同之处在于矩阵分块传输是将一个大矩阵分块搬移到不同的内核，而矩阵多核传输则是将整个大的矩阵搬移到所有的内核存储设备空间中，因此，相比于矩阵分块传输来说矩阵多核传输无论是在设计还是在参数的配置上都要简单很多。矩阵多核传输的计数结束方式也有两种，具体的计数机制和矩阵分块传输是相同的。图 3.9 是矩阵多核传输的示意图，图中核外存储资源中数据矩阵读出后将分别发送给 core0 和 core1 的输入端口，再由各自内核的 DMA 将输入数据矩阵写出到对应存储当中。

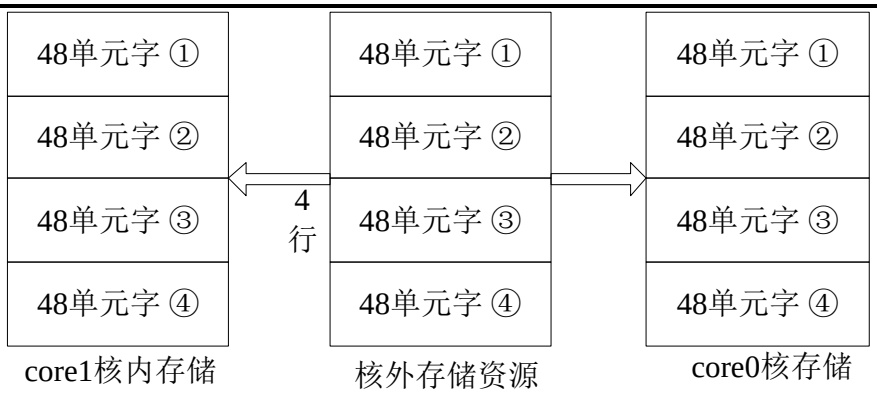

图 3.9 矩阵多核传输示意图

3.3 验证策略及计划

3.3.1 验证策略

根据上面的介绍，DMA 主要功能是负责 DSP 核内存储设备和核外存储设备之间的数据交换，不管采用何种传输方式进行数据的搬移其本质都是将源地址里面的数据对应搬移到目的地址里面。因此在进行 DMA 的功能验证时，可以将 DMA 当做一个黑盒[38]，不去关注 DMA 是怎么样实现其数据搬移的过程，而是关注其数据搬移的正确性，即是否将源存储设备要进行搬移的数据全部搬移到对应的目的设备中，并且保证数据在目的设备中的地址顺序是正确的，图 3.10 给出了 DMA 部件的一个具体验证策略。

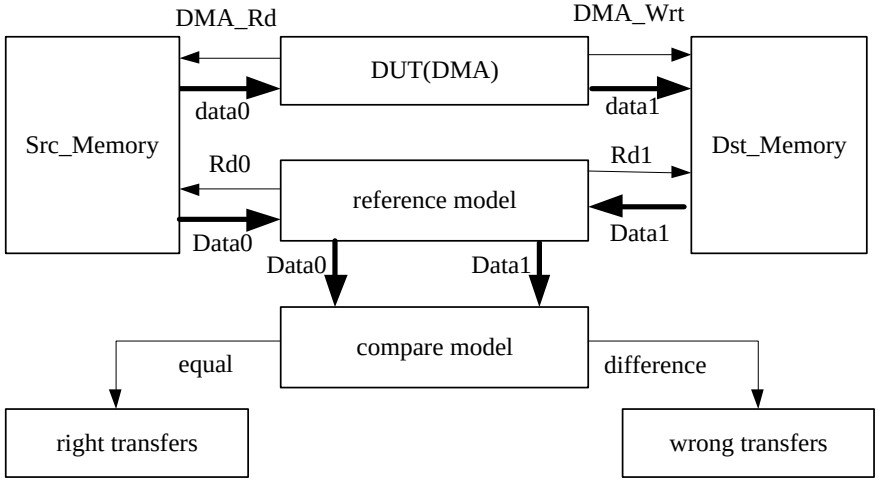

图 3.10 DMA 验证策略

DMA 从源存储设备中读取数据 data0，数据返回给 DMA 后由其根据配置的写信息将数据 data1 写到目的存储设备当中完成一次传输任务。当 DMA 传输完成之后，参考模型根据配置的参数字信息计算出有效的地址，然后根据地址分别从 Src_Memory 和 Dst_Memory 中读出有效的数据 Data0 和 Data1，最后由比对模型

compare model 按照数据的顺序比对 Data0 和 Data1 是否相等。比对结果如果相等则说明 DMA 的这次传输事务是正确的，如果不相等那么我们需要根据仿真信息来定位 DMA 出错的原因，并完成 RTL 代码的修改。

3.3.2 验证计划

验证计划的制定一般都是围绕着设计的功能点来进行的，根据前面对 DMA 功能的介绍可以知道需要验证的内容如下：

1. DMA 进行数据搬移时，根据源存储设备和目的存储设备的不同需要完成不同方向的数据搬移：核外 DDR 到 AM、核外 DDR 到 SM、AM 到核外 DDR、SM 到核外 DDR。
2. 源地址和目的地址变化方式可以进行不同的组合：根据 DMA 传输方式的不同，地址变化方式可以配置为递增、递减以及地址不变三种方式。
3. 源地址和目的地址不同偏移量之间的组合。
4. DMA 采用不同的传输方式完成传输任务：矩阵块到块传输、矩阵转置传输、矩阵分块传输以及矩阵多核传输。
5. DMA 进行一次传输任务时，需要搬移的数据总量。
6. 保证传输数据量相等的情况下，源二维矩阵和目的二维矩阵行列数的不同组合。
7. 同时启动 DMA 两条物理通道进行数据搬移任务：由于两条物理通道是相互独立的，因此验证这个功能点的时候还有 DMA 不同传输方式之间的组合。
8. 同时启动两个 DSP 核中的 DMA 进行传输任务。
9. DMA 传输过程中不同异常情况的验证，例如：地址越界、源目的设备异常等。

上面验证计划中基本上所有的功能点都是围绕着 DMA 作为主机时四种不同的传输方式所进行的，由于验证计划中涵盖内容较多以及 DMA 四种传输方式功能的复杂性，因此要保证上面的所有功能点都能得到完备的验证，一个高效准确并且包含所有传输方式比对功能的验证平台是迫切需要的。

3.4 本章小结

本章首先介绍了 M-DSP 中 DMA 部件的总体结构、基本功能以及其工作方式，并着重分析 DMA 中四种传输方式的特点，最后根据 DMA 的功能需求和工作特点给出了验证的基本策略和计划。

第四章　UVM 环境的构建

前面章节对 DMA 验证策略以及验证计划进行了分析和规划，基于上述内容本章节将详细介绍 DMA 控制器验证环境中各个组件的功能及实现方法并完成整个验证平台的搭建，然后阐述如何运用验证平台完成 DMA 部件各个功能点的验证。本文中验证环境的开发是基于 UVM 验证方法学来完成的，根据方法学分层次建模的思想逐步完成验证环境中各个组件的构建，并对环境进行调试保证其正确性。

4.1　UVM 验证平台简介

相对于使用 Verilog 语言构建的传统验证平台，基于 UVM 验证方法学来建模的验证平台已经有了自己的一套成熟的框架。整个 UVM 验证环境其实可以看成是一个个小的组件来构成的，只要按照其建模的顺序和设计的要求完成小组件的搭建，然后使用 UVM 的通信协议将所有的小组件联系起来就完成了整个验证环境的搭建。每当完成一个组件的搭建之后可以将其加入到环境中进行调试，看其是否达到了要实现的功能要求，这中方法明显方便了整个验证平台的调试过程，缩短了平台的搭建周期。

图 4.1 给出了 DMA 验证平台的总体框架图，根据整个验证环境结构图可以将验证平台看作是由两大部分组成：一部分是 UVM 基本组件部分；另一部分是 DMA 黄金模型。其中第一部分 UVM 环境主要是负责给 DMA 配置参数激励并启动 DMA 开始传输任务，在 DMA 发出读请求之后根据读请求信号按照协议返回 DMA 需要的数据信息并收集 DMA 输入和输出端口的数据信息。第二部分 DMA 黄金模型主要用来模拟 DMA 的数据搬移任务，首先根据配置的源地址从源存储设备中得到将要搬移的源数据，然后再从 DMA 完成搬移任务之后的目的存储设备中按目的地址得到目的数据，最后比对源数据和目的数据是否相等来判断 DMA 传输是否正确。

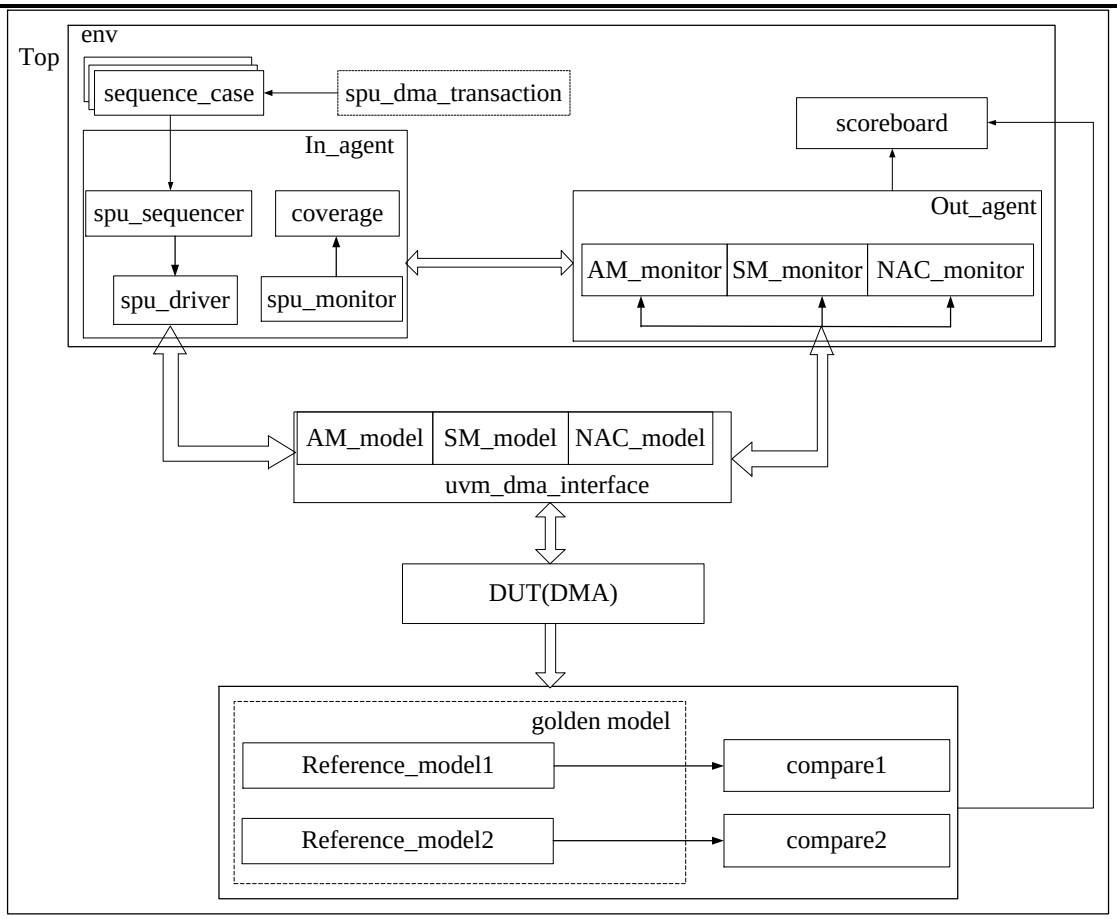

图 4.1 验证平台的总体架构

UVM 基本组件部分通过 uvm_dma_interface 组件与待测部件 DMA 进行连接完成通信，在 UVM 环境的内部也是通过 uvm_dma_interface 来相互通信的。一开始由 spu_driver 组件向 spu_sequencer 申请配置参数数据包，sequence_case 会将产生的数据包 spu_dma_transaction 通过 spu_sequencer 送给 spu_driver，最终数据包中的数据由 spu_driver 按照 SPU 和 DMA 之间的协议驱动到它们的数据配置端口完成 DMA 参数的配置。DMA 启动之后根据配置参数发出读请求，在 uvm_dma_interface 中根据 DMA 发出的请求信息返回对应的数据信息到 DMA 对应的输入端口上，然后由 DMA 按照不同传输任务要求的机制将数据写出到目的设备当中。在 In_agent 中 spu_monitor 负责监测 SPU 到 DMA 的输入端口，然后将配置的参数信息传给 coverage 组件进行功能覆盖率的统计。Out_agent 则负责监测各个存储设备与 DMA 之间的输入输出端口，其中 am_monitor 和 sm_monitor 分别负责核内两个存储设备与 DMA 之间的数据监视，nac_monitor 负责核外存储设备与 DMA 之间的数据监视。Out_agent 中各个监视器收集到的数据最后会传递给 Scoreboard 组件进行数据信息的比对。

Golden model 负责模拟 DMA 数据搬移时的地址变化过程，在顶层的 TOP 环境直接通过实例化黄金模型的环境，然后将 DMA 端口和内部信号通过宏定义连

接到模型的内部实现二者之间的通信。DMA 总共有四种不同的数据搬移方式，根据它们各自传输方式的特点在黄金模型中分为两大块进行 DMA 传输时地址和数据的模拟。其中 reference_model1 负责模拟矩阵块到块和矩阵转置传输时的有效地址及有效数据，然后由 compare1 完成 DMA 数据传输的正确性比对；reference_model2 则负责模拟矩阵分块传输和多核传输时的有效地址和数据，然后由 compare2 进行比对。

4.2 UVM 环境构建

UVM 环境的构建是整个验证平台的核心部分，根据前面介绍的 UVM 环境的建模思想，下面将根据建模的顺序介绍 UVM 环境中各个组件具体的实现和功能。图 4.2 为 UVM 验证环境的树状结构图，UVM 环境中基本功能组件的管理和实现都是按照此结构图的顺序来进行的。其中输入代理（In_agent）负责 DMA 参数项的配置和启动以及输入端口数据的收集，输出代理（Out_agent）负责 DMA 输出端口数据的收集，记分板（scoreboard）负责对平台收集的输入和输出数据按照协议进行结果比对。In_agent 对 DMA 输入端口进行激励配置时，其事务级激励（spu_dma_transaction）由序列生成器（spu_sequence）负责生成，再由 sequence 机制实现激励向驱动器的注入。UVM 平台结构按照层次化的方式进行建模，从最底层的 sequencer 中生成激励的数据结构开始，逐渐完成整个树形结构的生长。

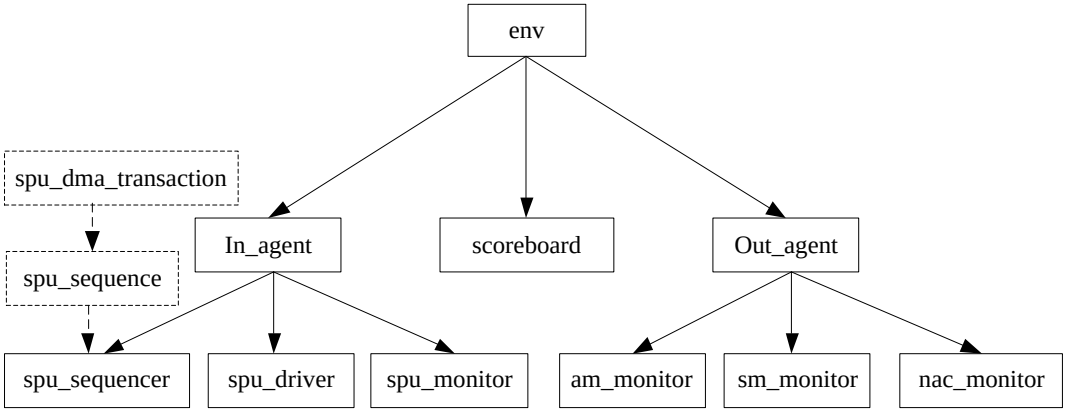

图 4.2 UVM 树状结构图

4.2.1 事务级激励

事务级激励（spu_dma_transaction）是按照 DMA 参数输入协议来构建的，负责生成 DMA 端口能够识别的事务级激励。SPU 与待测试部件 DMA 之间物理协议中的数据交换主要是由八个配置参数项组成的，平台根据端口的协议将物理数据抽象成事务级数据之后使其能够在 UVM 环境中流通。为了方便对事务数据流的定义，根据 DMA 参数字的组成结构在 spu_dma_trans.svh 头文件将其定义成新的

通用数据类型结构。图 4.3 所示为定义数据结构的部分源代码，可以看出参数项 para1_data 是由若干不同的变量组合到一个结构当中，这些变量的含义可以对应着 3.1.2 小节中 DMA 参数字的结构图进行查看。spu_model_e 则是定义的一个枚举数据类型，对应着 SPU 与 DMA 之间的控制信号端口，当 spu_model_e 取 W 时表示 SPU 向 DMA 写参数；当 spu_model_e 取 R 时表示 SPU 从 DAM 里面读参数。

```
typedef struct packed {
      bit [31:28] src_addr_35_32;
      bit [27:24] dst_addr_35_32;
      bit [15:14] damode;
      bit [13:12] samode;
      bit [9:8]   tmode;
        …
     bit        tcint;
     bit        ts;
     bit        link;
    }para1_data;              //  congfig para1
…
typedef enum {W,R} spu_model_e; //(R=1/W=0)
```

图 4.3 数据结构类型的定义

完成上面的数据类型定义之后，可以在 SPU 输入给 DMA 的事务级激励类 spu_dma_transaction 中直接定义使用这些参数项类型。一般的输入数据类型中会包含有两种不同作用的数据包，一种数据包里面是纯粹的激励数据流，一种数据包的数据是用来控制前面数据包中的激励在平台的流动。根据 SPU 与 DMA 之间的端口配置协议，八个参数项是通过相同的配置端口按照地址顺序先后完成写入的，其写入过程虽然是相对独立的但是在使用时八个参数项却是缺一不可，所以在处理参数项的配置过程时应该将它们当做一个整体。因此在 SPU 的输入类 spu_dma_transaction 中只作了纯粹的激励数据包，而对于控制的端口数据包没有进行定义而是在后面的驱动器中直接进行激励包的控制。

spu_dma_transaction 中的参数变量包含了配置需要的 8 个参数以及寄存器，由于 spu_dma_transaction 在后面的 UVM 环境中会使用 sequence 机制，因此其应该从 uvm_sequence_item 这个基础类中派生而来。在定义的参数变量中 para1-para8 分别对应着 DMA 的八个配置参数项，其他的参数则是对应着 DMA 的配置寄存器。spu_dma_transaction 中使用 rand 将参数定义成随机变量，在仿真时只要对这个类进行随机，里面的参数变量就会产生随机激励，以此来给验证平台提供大量的随机测试激励。

DMA 参数项以及寄存器的配置都有着各自规定的协议，并且各个参数项变量也不是完全独立的，他们各自之间还存在着相互制约的关系。要使得上面定义的参数变量驱动给 DMA 之后能够正常运行，还必须对参数变量进行约束，使其

在一定的规则控制下才能产生正确的随机激励。图 4.4 给出了对参数变量施加约束的部分源程序，在使用 constraint 对参数施加约束条件时，这些条件并不是自上而下来进行解析求解的，而是类似于一种声明性的条件并行执行，其中所有的表达式都是同时有效的。由于 DMA 丰富多样的传输方式以及错综复杂的参数配置关系，在对随机参数变量施加约束之前必须对 DMA 各种传输模式下参数项以及寄存器的配置关系有一个清楚的认识。比如，DMA 在进行二维矩阵转置传输时，根据可配置的最小数据传输位宽的不同，每次进行矩阵转置的单位矩阵只能是 8*8 或者 4*4，因此在对源设备和目的设备的行计数及单数计数进行约束时，可以通过条件操作符（if-else）来规定行计数和单元计数是 8 或者 4 的倍数。同样 DMA 寄存器的配置也是有其相应的规则，启动 DMA 的配置寄存器 ESR 其低 16 位分别对应着 16 个参数通道的地址，故寄存器 ESR 的随机值应该与参数通道的地址之间存在着一个等价的约束。

对事务类 spu_dma_transaction 中随机参数变量的约束是整个类中的核心部分，其约束添加的好坏不仅关系到验证平台中激励的可靠性问题，同时也影响到后续 DMA 覆盖率收集的效率。孰知 DMA 中各种错综复杂的配置条件之后，根据验证计划中的功能点逐步完成对随机变量的约束，这样既能保证验证激励约束的正确性，又能保证功能点验证的完备性。

```
constraint para4_c{
                para4[31:16] inside{[16'h0:16'h03fe]};
                para4[15:0] inside{[16'h0001:16'hffff]};
                tmp==(para4[31:16]+16'h1)*para4[15:0];
                (para1[9:8]==2'b10)->(tmp>para2[15:0]);
               (para1[9:8]==2'b10)->(tmp%para2[15:0]==0);
             if((para1[9:8]==2'b01)&&(para2[15:0]==16'h1))
                  {para4[31:16]%8==0;
                   para4[15:0]%8==0;}
}
…
constraint esr_c {
              esr[31:16] == 16'h0;
              esr[15:0] ==(16'h0001<<addr[8:5]);
 }
```

图 4.4 受约束的随机激励

完成类中所有随机变量约束条件的添加后，使用 uvm_field 系列宏来对类中的随机变量进行注册，方便后面 UVM 的功能组件对随机变量进行操作，图 4.5 给出了生成 DMA 四种传输方式的随机激励示例，图中①~④分别为矩阵块到块传输、矩阵转置传输、矩阵分块传输和矩阵多核传输的随机激励具体示例。根据生成的随机激励以及 DMA 各传输参数字所代表的具体意义，可以判断上述约束条件是否满足当前验证的需要。

①

```
-----------------------------------------
Name       Type           Size  Value
-----------------------------------------
spu_data   spu_dma_data   -     @33562760
  cier     integral       32    'hffffffff
  para1    integral       32    'h8000504e
  para2    integral       32    'h0
  para3    integral       32    'h2c104
  para4    integral       32    'h3bf1
  para5    integral       32    'h4000066c
  para6    integral       32    'he03ff
  para7    integral       32    'h400
  para8    integral       32    'h0
  prir     integral       32    'h0
  esr      integral       32    'h400
  cipr     integral       32    'h0
-----------------------------------------
```

②

```
-----------------------------------------
Name       Type           Size  Value
-----------------------------------------
spu_data   spu_dma_data   -     @33607115
  cier     integral       32    'hffffffff
  para1    integral       32    'h8000514e
  para2    integral       32    'h1
  para3    integral       32    'h40000040
  para4    integral       32    'h400040
  para5    integral       32    'hf3840
  para6    integral       32    'h400040
  para7    integral       32    'h800
  para8    integral       32    'h4
  prir     integral       32    'hffffffff
  esr      integral       32    'h800
  cipr     integral       32    'hffffffff
-----------------------------------------
```

③

```
-----------------------------------------
Name       Type           Size  Value
-----------------------------------------
spu_data   spu_dma_data   -     @33623709
  cier     integral       32    'hffffffff
  para1    integral       32    'h8000525e
  para2    integral       32    'h250e90
  para3    integral       32    'h0
  para4    integral       32    'he80030
  para5    integral       32    'h40000060
  para6    integral       32    'h3a3000c
  para7    integral       32    'h2
  para8    integral       32    'h3a40
  prir     integral       32    'hffffffff
  esr      integral       32    'h2
  cipr     integral       32    'hffffffff
-----------------------------------------
```

④

```
-----------------------------------------
Name       Type           Size  Value
-----------------------------------------
spu_data   spu_dma_data   -     @33562760
  cier     integral       32    'hffffffff
  para1    integral       32    'h80005346
  para2    integral       32    'h0
  para3    integral       32    'h1bd20
  para4    integral       32    'h1ec0
  para5    integral       32    'h40006198
  para6    integral       32    'h1eb0010
  para7    integral       32    'h4000
  para8    integral       32    'h0
  prir     integral       32    'hffffffff
  esr      integral       32    'h4000
  cipr     integral       32    'h0
-----------------------------------------
```

图 4.5 DMA 随机激励示例

4.2.2 输入事务代理

输入事务代理 In_agent 是整个 UVM 验证平台的起始部分，主要负责给 DMA 配置传输参数并且通过写 ESR 寄存器来完成 DMA 的启动。构成 In_agent 的组件主要有三个部分，一是序列发射器 spu_sequencer 为驱动器提供激励，二是驱动器 spu_driver 按照 SPU 和 DMA 之间的端口协议将激励输入给 DMA，三是监视器 spu_monitor 负责监视 SPU 与 DMA 之间的信号端口，下面将依次介绍这三个组件的具体设计实现。

4.2.2.1 序列发射器实现

根据 2.2.3 章节中关于 UVM 工作机制的介绍，sequencer 必须配合着 sequence 一起工作实现 sequence 机制才有意义，sequence 负责生成事务级激励，sequencer 负责仲裁并传输生成的激励到驱动器。sequence 机制在平台中由两大组件来实现，一个是 spu_sequence，一个是 spu_sequencer，二者分工用来产生 DMA 所需要

的激励。在驱动器对事务数据类进行驱动之前，先由 spu_sequence 机制根据不同的测试用例来完成对序列场景的定义，然后由 spu_sequencer 完成事务数据和驱动器之间的通信。

1. spu_sequencer 的实现

spu_sequencer 主要负责将生成的 DMA 事务级激励传输到驱动器当中，其代码定义如图 4.6 所示。从 UVM 基本功能库 uvm_sequencer 中派生出 spu_sequencer 完成类的定义，spu_sequencer 会自动继承 uvm_sequence 基本类中的所有功能，这种方式很明显方便了验证人员对组件的建模。

```
class spu_sequencer extends uvm_sequencer #(spu_dma_data);
function new(string name, uvm_component parent);
    super.new(name, parent);
endfunction
`uvm_component_utils(spu_sequencer)
endclass
```

图 4.6 spu_sequencer 实现代码

2. spu_sequence 的实现

对 spu_sequence 进行构建时，根据 3.3 节中验证计划的内容提炼出主要的几种测试用例，然后再根据测试用例的功能分别完成序列场景的设计，以此来产生能满足测试用例的激励。图 4.7 为整个序列场景的结构图，图中每个测试用例中都包含了两个部分机制，一个 default_sequence 和一个 sequence_case，其中 default_sequence 机制负责对 sequence 进行启动，在整个平台中只需要在某一处进行一次设置之后就可以自动对 sequence 完成启动。在 sequence_case 中来完成对要测试的功能点进行定义，由于在事务类中已经完成了对 DMA 不同传输模式下的参数变量的定义，因此在 sequence_case 中只需要定义 DMA 使用何种方式来完成数据搬移就可以完成一个序列场景的定义。例如在 sequence_case0 中定义 DMA 传输模式是矩阵块到块传输，sequence_case2 中定义 DMA 传输模式是矩阵分块传输。

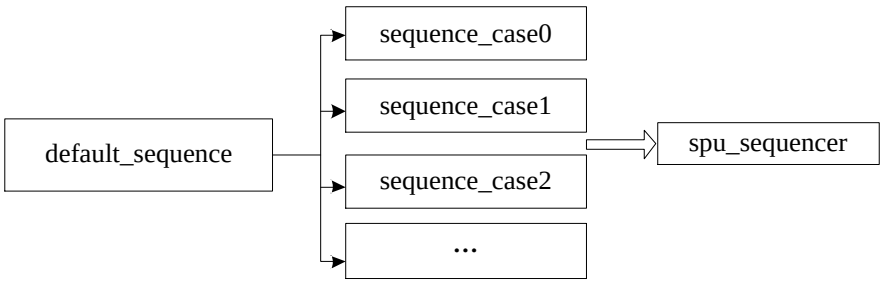

图 4.7 序列场景结构图

在不同的 sequence_case 中虽然其中定义的约束场景不同，但是都有着相同的数据激励生成的流程，图 4.8 为 sequence 中产生事务激励流的流程图。

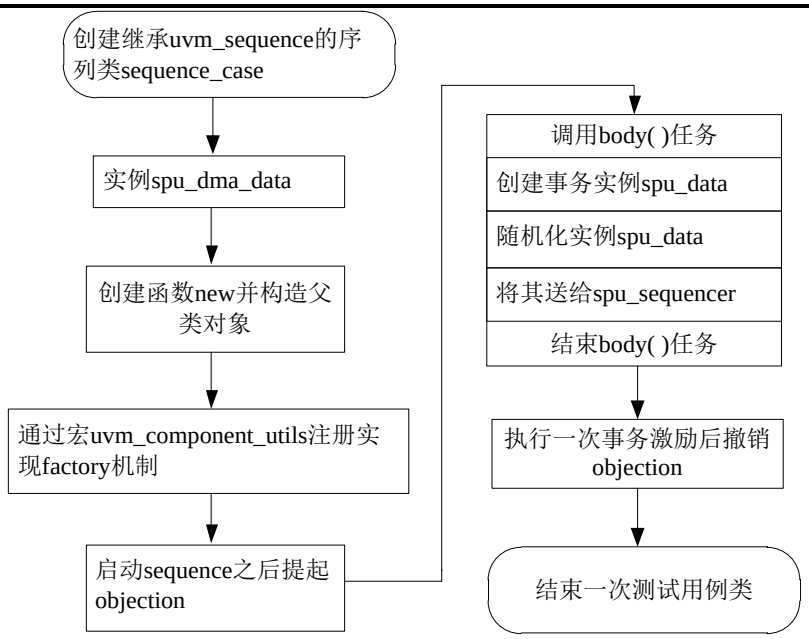

图 4.8 传输事务激励产生流程图

测试用例都是按照上面所示的流程图来设计的，针对不同的测试点，只需要在序列生成器 sequence_case 中进行约束定义即可，因此 sequence 机制的使用显著增加了激励流生成的灵活性，也方便了对测试用例的定义。sequence_case 中最关键的部分是类中 body 任务的设计，当一个测试用例启动之后会自动执行任务中的代码完成激励的产生和发送。在 sequence 中实现激励的产生和发送时，UVM 库中提供了丰富的宏以方便对事务激励进行设置，完成对不同情况下测试用例的设计。以上面矩阵分块传输的场景 sequence_case2 为例，图 4.9 给出了定义此序列场景的部分代码。其中宏 uvm_do_with 包含了创建事务类实例、约束类中变量、随机化事务实例以及发送事务包到 spu_sequencer 这四个步骤，由此可见 UVM 宏的使用大大减少了平台设计的代码量。在 repeat 语句中可以设置序列器中 transaction 的个数，当序列器器将所有的 transaction 送给 spu_sequencer 发送之后即结束生命周期，同时也表明一次 test_case 仿真完成。

```
virtual task body();
          if(starting_phase != null)
     starting_phase.raise_objection(this);
        repeat(1000)
              begin
`uvm_do_with(spu_data,{spu_data.para1[9:8]==10;})
               end
if(starting_phase != null)
starting_phase.drop_objection(this);
endtask
```

图 4.9 事务包产生以及发送

在驱动器和测试用例之间 spu_sequencer 相当于起到了一个开关的作用，当驱动器需要配置激励时向 spu_sequencer 发出一个申请 spu_dma_transaction 的请求，

然后 spu_sequencer 会从 spu_sequence 处得到生成的测试激励并将其送过驱动器。因此，只有当驱动器向 spu_sequencer 发出了请求之后事务包和驱动器之间的通路才会被打通，否则 spu_sequencer 会一直处于等待请求的状态。

4.2.2.2 驱动器的实现

当事务激励包通过 spu_sequencer 传递到驱动器（spu_driver）之后，驱动器按照 SPU 与 DMA 之间的通信协议对激励包进行解析，将事务包中的参数激励对应成端口可以接收的物理信号。spu_driver 是整个 DMA 验证环境中的核心组件，同时也是 DMA 能正常开始工作的提前，图 4.10 为 spu_driver 配置 DMA 端口激励的具体流程。

整个流程可以分为两个部分，第一部分是建立 spu_drive 与 DMA 之间的连接关系，第二部分是驱动激励并管理平台运行。为了保证平台中激励能正确驱动到 DMA 的输入端口，spu_driver 采用 UVM 的 phase 功能将 spu_driver 的工作分为两个 phase 来进行。其中 build_phase 机制通过虚接口建立 spu_driver 和 DMA 输入输出端口之间的连接关系，而 main_phase 则专门负责激励的驱动并管理整个环境的运行进程，环境启动后两种 phase 按顺序自动执行各自的任务。

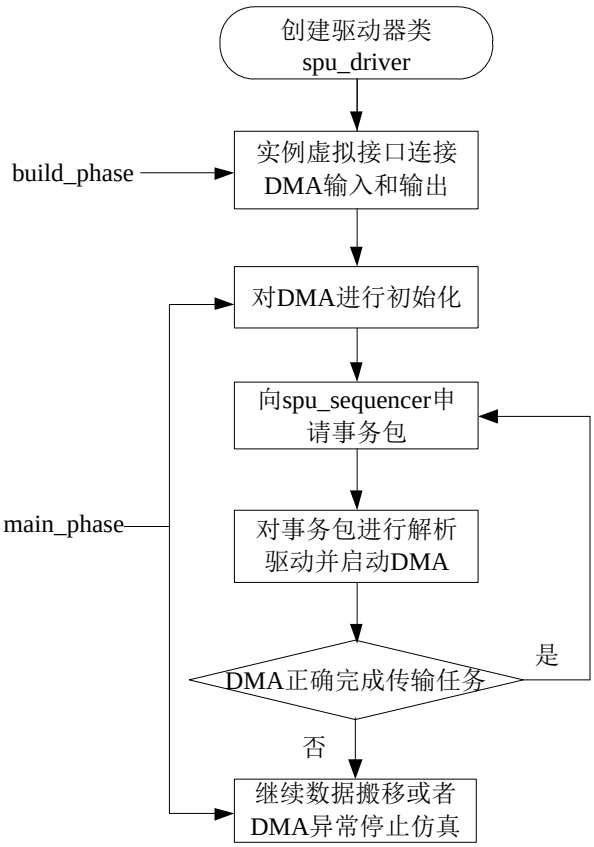

图 4.10 驱动器执行流程图

完成 spu_driver 类的定义以及组件与 DMA 之间的连接关系之后，首先需要对

整个 DMA 进行一个复位操作，给所有的输入端口一个初始值，这个过程是通过一个 reset 任务来完成的。完成 DMA 的初始化之后就可以向 spu_sequencer 申请事务激励，对事务包的解析和驱动是整个驱动器的核心部分。上述过程都是在 main_phase 机制中实现的，图 4.11 为 spu_driver 驱动事务包的部分程序代码，DMA 从配置到启动的过程由四个步骤来完成：第一步配置“通道中断使能寄存器”（CIER），这样 DMA 完成数据搬移任务之后能发出结束中断信号；第二步将事务类中打包数据进行解包，并通过 get_and_driver 任务计算出参数配置地址，然后将参数写入到 DMA 对应的参数通道地址中；第三步选择处理 DMA 配置参数的物理通道，根据需求随机选定通道 0 或者是通道 1 来进行数据搬移；第四步通过配置 ESR 寄存器的方式来启动 DMA 进行数据传输。

```
spu_trans_dma(CIER,req.cier);  //       1. config CIER
get_and_driver(req);                 //       2. config transaction
spu_trans_dma(PRIR,req.prir);  //       3. config PRIR
spu_trans_dma(ESR,req.esr);    //       4.  config ESR
task spu_driver::get_and_driver(spu_dma_data seq);
…
      for(int i=0;i<data_size;i++)
      begin
       @(posedge vother_in.clk);
     vother_in.SPU_DMA_ReqAddr[0] = seq.addr+4*i;
     vother_in.SPU_DMA_WrtData[0] = spu_data[i];
     vother_in.Peri_PBUSConfRw[0] = 1'b0;
      vother_in.Peri_PBUSConfReq[0]= 1'b1;
     end
endtask
```

图 4.11 事务包驱动部分程序

驱动器配置并且启动 DMA 之后，在 DMA 进行数据搬移的过程中是不能对之前的物理通道进行配置启动操作的，只有在其完成上一传输任务之后驱动器才能向 spu_sequencer 申请一个新的事务来进行下一次激励的解包以及配置。图 4.12 给出了 spu_driver 中平台运行管理机制部分代码，spu_driver 通过 try_next_item 语句不断向 spu_sequencer 申请新的事务激励。当 spu_driver 完成一次 DMA 的参数配置并启动传输任务，平台开始循环检测任务传输结束信号直到 DMA 自动比对结束之后，spu_driver 会向 spu_sequencer 发出 item_done 信号指示本次激励执行结束可以开始传送新的事务激励。

```
task spu_driver::main_phase(uvm_phase phase);
while(1)
  begin
    seq_item_port.try_next_item(req);              //向sequencer申请激励
   while(vother_in.SPU_DMA_ReqAddr[0] == 36'h0401bfff8)
   begin
     @(posedge vother_in.clk);
    if(vother_out.DMA_pointint[0])                 //DMA传输结束中断
    begin
    spu_trans_dma(CIPR,req.cipr);
    @(posedge vother_out.compare_end0)      //比对结束
    break;
    end
  end
   seq_item_port.item_done();                      //激励完成反馈信号
   end
endtask
```

图 4.12 平台运行管理机制

DMA 传输没有结束之前驱动器不会向 spu_sequencer 发送上一次事务正常接收的握手信号，只有当 DMA 完成之前的数据搬移任务并且传输正确时，驱动器才可以申请新的激励。因此驱动器通过握手信号可以很好的控制前后两个事务之间的流水，保证所有的事务激励都能得到正确的处理。

4.2.2.3 配置激励监视器

驱动器（spu_monitor）将打包的事务类数据激励驱动到 DMA 端口并启动后，DMA 根据配置的 8 个参数项来执行其数据搬移的行为，对配置激励的监测即是对 DMA 即将要进行的行为的一种监测。由于 SPU 与 DMA 之间所有的数据参数配置与寄存器配置物理接口只有一套，驱动器在对输入的激励进行处理时是按照一个串行的先后顺序完成的，所以在监视器监测时需要根据其配置的地址才能区分出每次配置 DMA 的数据具体代表的意义。图 4.13 和图 4.14 分别是监视器通过配置地址进行寄存器数据和配置参数激励采集的部分程序。

```
@(posedge spu_in.Peri_PBUSConfReq[0])
begin
data         = spu_in.SPU_DMA_WrtData[0];
data_addr = spu_in.SPU_DMA_ReqAddr[0];
 end
case(data_addr)
  CIER  : spu_data.cier  = data;
  PRIR  : spu_data.prir  = data;
  ESR   : spu_data.esr   = data;
  CIPR  : spu_data.cipr  = data;
  default: spu_data.prir = 'hffff;
endcase
```

图 4.13 配置寄存器数据采集

```
case(data_addr[4:0])
  PARA1 : spu_data.para1 = data;
  PARA2 : spu_data.para2 = data;
  PARA3 : spu_data.para3 = data;
  PARA4 : spu_data.para4 = data;
  PARA5 : spu_data.para5 = data;
  PARA6 : spu_data.para6 = data;
  PARA7 : spu_data.para7 = data;
  PARA8 : spu_data.para8 = data;
  default: spu_data.para1 = 'h0;
endcase
```

图 4.14 配置参数激励采集

在对配置激励的采集过程中，SPU 配置 DMA 的数据激励按照功能可以划分为两个部分，其中一部分是寄存器的配置，另外一部分是控制 DMA 数据搬移行为的参数项。对于寄存器配置数据的采集，由于在 DMA 设计中功能寄存器都已经分配好了特定的地址空间，所以只需要按照全地址进行数据的采集。在对配置参数项进行采集时，由于 DMA 有 16 个参数通道并且分别对应着 16 个不同的地址空间来进行激励参数的配置，因此在对传输参数进行采集时需要以通道地址的低 5 位来作为数据采集的条件。

监视器完成输入激励的采集之后，生成的事务激励将通过 TLM 通信机制传递到平台其他各个组件中。在 spu_monitor 中通过 uvm_analysis_port 参数类中封装的 write 函数进行激励的收集，此时 spu_monitor 组件被设置为 TLM 机制中的数据发送方（PORT），对应的接收方只需通过 uvm_blocking_get_port 接收数据就可完成监测的激励数据在平台中的通信。

4.2.3 DMA 虚拟接口设计

UVM 验证平台与 DMA 之间的连接通信是通过虚拟端口 uvm_dma_interface 来实现的，虚拟组件 uvm_dma_interface 相当于在平台和待测设计 DMA 之间架设了一条桥梁。平台中的数据可以通过 uvm_dma_interface 传递给 DMA 来作为其输入端口的数据激励，DMA 的输出数据也可以通过虚接口传递到平台组件当中作为判断条件来使用。

一般的 interface 主要是对要验证的设计中所有的输入输出端口信号进行定义，将 DMA 中所有的端口信号定义成 logic 类型变量即可。事实上，interface 能够实现的功能远不是简单的接口信号定义以及实现验证环境和 DMA 之间的通信，在 interface 的一些更高级应用中可以使用任务和函数来完成一些功能的定义，还可以在 interface 中使用 Verilog 语言中的 always 语句和 initial 语句。DMA 在启动之后，会解析配置参数项中的信息向源存储设备发出数据请求包，当源设

备接收到数据请求包之后会根据请求中的地址等信息按照协议将数据返回给DMA 的数据输入端口。要在 driver 中处理上面的 DMA 读请求会显得很烦琐，而且也不利于在仿真时对数据的调试。在处理 DMA 读请求操作时，只要是有请求过来源存储设备就得按照请求的信息将 DMA 需要的数据返回到 DMA 的输入数据端口，也就是说这个处理的操作其实也是一个“always”的语句块操作，因此可以将驱动 DMA 数据返回这部分操作在 interface 中使用“always”语句来完成。

图 4.15 所示为 uvm_dma_interface 中根据读数据请求包驱动数据返回到 DMA 输入端口的部分程序。根据 DMA 源设备地址空间大小和地址空间划分，使用一个定宽数组 mem_src 来模拟源存储器给 DMA 的读返回提供有效的数据，当 DMA 发出的读请求有效时，根据对应好的读地址从数组模型中取出需要的数据并驱动到对应的输入端口。完成所有的源设备数据返回处理之后，由于在 uvm_dma_interface 中是直接对其中定义的端口变量进行驱动的，因此只需要在顶层 Top 互连中将 interface 中驱动的变量和 DMA 对应的输入端口连通就可以实现 DMA 读数据返回的输入。

```
always @(*)
begin
if(DMA_AM_RdReq[0])
 begin
  AM_DMA_RdMask[0]      =  DMA_AM_RdMask[0];
  AM_DMA_RdDataRdy[0]  =  1;
  AM_DMA_RdShift[0]      =  DMA_AM_RdAddr[0][4:0];
  AM_DMA_RdData[0]       =
{mem_src[{DMA_AM_RdAddr[0][23:5],3'b111}],mem_src[{DMA_AM_RdAddr[0][23:5],3'b110}],
  mem_src[{DMA_AM_RdAddr[0][23:5],3'b101}],mem_src[{DMA_AM_RdAddr[0][23:5],3'b100}],
  mem_src[{DMA_AM_RdAddr[0][23:5],3'b011}],mem_src[{DMA_AM_RdAddr[0][23:5],3'b010}],
  mem_src[{DMA_AM_RdAddr[0][23:5],3'b001}],mem_src[{DMA_AM_RdAddr[0][23:5],3'b000}]};
 end
end
```

图 4.15 读数据返回输入驱动程序

在 uvm_dma_interface 中实现 DMA 读返回数据的输入驱动，不仅减小了平台中 spu_driver 组件的工作量，有利于验证环境的维护，同时，uvm_dma_interface 中的信号在仿真时可以直接查看输出波形，有利于验证环境的调试。

4.2.4 输出事务代理

输出事物代理主要任务是负责监视 DMA 的输出数据端口，DMA 在收到源设备返回的数据之后会根据配置的参数信息将返回的数据写出到目的设备对应的地址当中，只要 DMA 与目的设备之间的物理接口有数据写出，监视器就将监测到的数据根据其地址打印到对应的输出文本当中。

DMA 进行数据搬移时目的设备有三个，核内的 AM、SM 以及核外的 DDR，

根据目的设备的不同，分别有三个监视器对这三个目的设备进行监测。考虑到 DMA 每次进行数据搬移时，要么源设备是核内存储目的设备是核外存储，要么源设备是核外存储目的设备是核内存储，因此在进行监视器设计时将每个设备的输入和输出端口的监测分别集中到各自监视器中进行统一管理。其中 am_monitor 负责 AM 存储设备输入和输出的数据监测，sm_monitor 负责 SM 存储设备输入和输出的数据监测，nac_monitor 负责核外存储设备与 DMA 之间的数据监视。

由于三个存储设备监视器执行的步骤和设计的原理是相同的，在这里只对其中的 am_monitor 进行详细的介绍，图 4.16 为监视器执行数据收集的过程示意图。首先通过 config_db 机制以虚接口 uvm_dma_interface 作为桥梁将 DMA 的输入和输出端口与监视器连通，然后再通过配置的参数信息判断 AM 在本次数据传输中是源设备还是目的设备。若 AM 是源设备那么就执行输入端口数据的监视任务，是目的设备那就执行输出端口数据的监视任务，否则就说明 AM 不参与此次数据搬移任务。在对端口数据进行收集时需要不停的对其进行监测，一旦发现有数据输入或者输出就按照传输协议将监测到的数据写入到模拟 AM 存储器的数组当中保存，等待 DMA 完成此次传输任务之后，再将数组中保存的数据按照顺序打印到对应的文本当中以方便后面的组件 Scoreboard 对传输的数据进行正确性比对。在使用数组模拟 AM 存储器时，由于 DMA 每次搬移的数据矩阵大小不同其所占的存储空间地址也是不一样的，根据这种特性可以将数组定义成一个关联数组，以减少存储空间的浪费，提高验证平台仿真的效率。

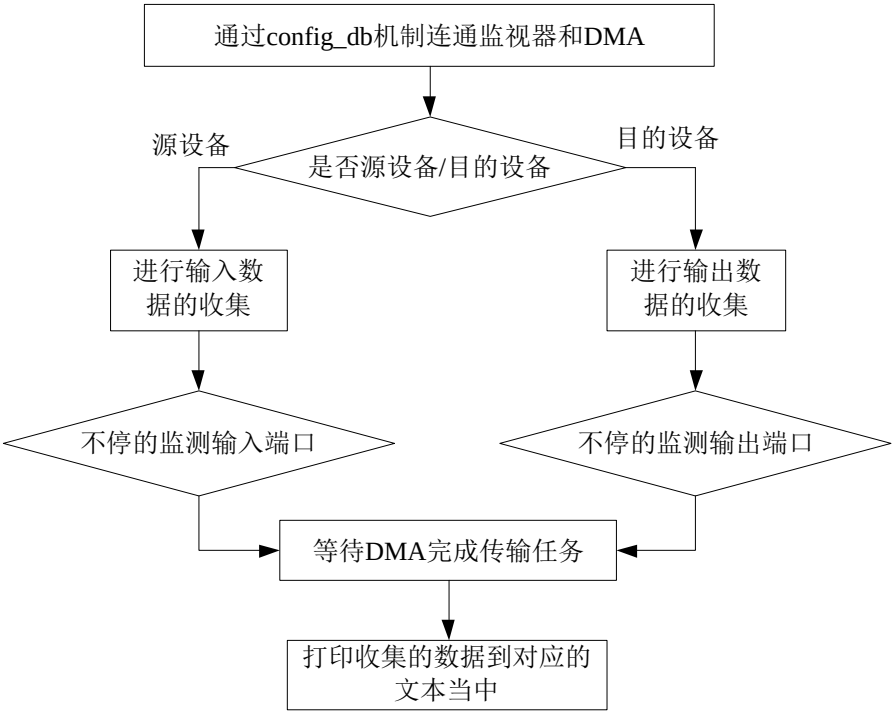

图 4.16 监视器监测流程图

4.3 黄金模型

黄金模型的构建在整个验证环境的搭建当中占据着很重要的地位，验证平台在判断DMA完成一次数据传输任务的正确性时需要将DMA的输出和黄金模型输出的预期值进行比对，因此黄金模型不仅要能模拟出 DMA 的功能，还必须保证其结果的正确性。考虑到前面章节中介绍的 DMA 传输方式的复杂性以及某些传输模式下的特性，要想通过参考模型完全模拟出 DMA 的这些传输功能显然是很困难的，同时也会极大的增加验证平台的复杂性。比如 DMA 在进行核外到核内的矩阵块到块传输时，输出的数据可以乱序写出到核内的存储设备当中，显然要在黄金模型中模拟出DMA这种乱序功能是困难的。按照3.3小节的验证策略来构建参考模型，不仅很好的避免了复现 DMA 传输的复杂设计过程，同时黄金模型可以直接复用到DMA芯片级验证环境当中[38]，明显缩短了DMA芯片级环境建模时间。

图 4.17 为参考模型完成 DMA 传输功能模拟并得到预期值的一个基本流程。其中输入数据存储体中的数据要与UVM平台中给DMA提供读返回数据的源设备中数据保持一致，输出数据则需要通过 DMA 的输出端口根据传输协议将有效数据保持到目的存储体中。完成输入数据和输出数据的收集之后，还需要根据配置的激励计算出输入和输出数据矩阵的有效地址信息。对于输出来说，根据目的有效地址信息得到的数据即为 DMA 完成搬移任务之后的预期值，而输入数据则还需要根据不同传输方式的特点对源地址进行处理才能得到预期值，这两个预期值最后的结果必须是相同的。

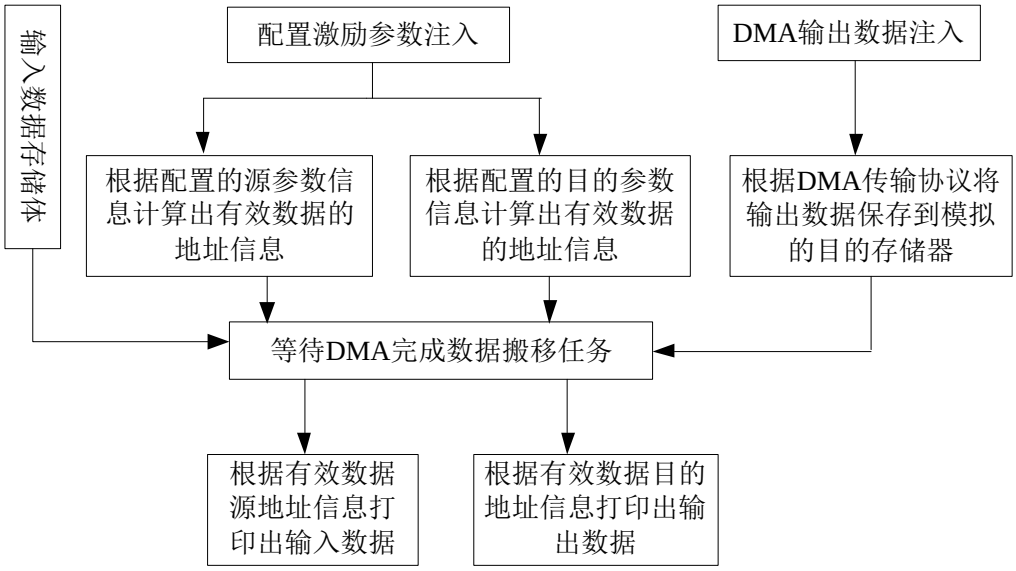

图 4.17 参考模型基本流程图

DMA 在使用不同的传输方式进行数据搬移时，对输入数据需要根据传输方

式的特性进行特别处理，对于其他的流程部分采用相同的方法完成数据的处理即可。根据 DMA 四种传输方式的特点构建黄金模型时，将矩阵块到块传输和二维矩阵转置传输的模型设计封装在一起组成 reference_model1，将矩阵分块传输和矩阵多核传输的模型设计封装在一起组成 reference_model2，两种模型根据各自传输方式的特点将输入和输出数据分别打印输出到对应的源数据文本和目的数据文本当中。

4.3.1 源目的地址计算

DMA 进行数据矩阵搬移时，其整个矩阵的有效地址空间可以通过配置的激励参数计算得出，图 4.18 所示为数据矩阵地址计算原理图。第 0 行初始地址是激励参数中配置的源初始地址或者目的初始地址，第 0 行初始地址与第 0 行末地址之间的间隔即为行单元数所占的地址空间大小。矩阵的行与行之间还存在着一个地址偏量，若不考虑地址偏量的因素，那么矩阵下一行的初始地址即为上一阵列行的末地址；若考虑到阵列行之间的地址偏量，那么第 1 行的初始地址即为第 0 的末地址加上行偏量。以此类推，根据配置激励参数中的初始地址、单元数和行偏量就可以计算出源数据矩阵和目的数据矩阵所有阵列行的初始地址和末地址信息。

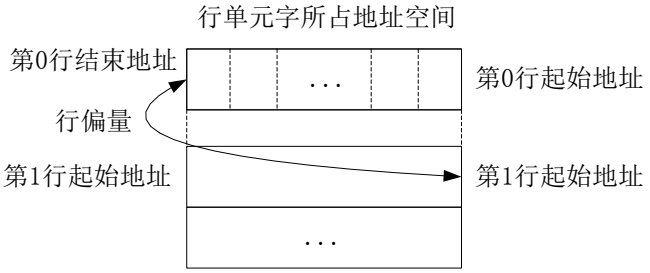

图 4.18 矩阵地址计算原理图

4.3.2 DMA 输出数据处理

当 DMA 的输出数据注入到黄金模型后，模型根据传输的端口协议将数据保存到模拟的目的存储设备当中，其具体实现的部分代码如

图 4.19。图中的部分代码表示当目的设备是 AM 时，将要输出到 AM 中的数据通过模型保存到模拟的 AM 存储器 mem_AM 当中。其中 AM_WrtMask 指示了输出数据信号 DMA_AM_WrtData 中有效的单元字，若此时的单元字有效就将其保存到数组当中否则就不做保存处理。有效单元字在数组中是按照 DMA 输出的地址 DMA_AM_WrtAddr 顺序来存放的，因此通过上面的方法就可以将输入到 AM 中的有效数据按照地址顺序保存到目的存储器 mem_AM 中。类似的，当目的

设备是 SM 或者 DDR 时处理输出数据的思想和 AM 是一样的。

```
always @(*)
begin
if(DMA_AM_Bk0or8Wrt[0])
  begin
mem_AM[DMA_AM_WrtAddr[0][23:2]+0]= AM_WrtMask[0][0] ?  DMA_AM_WrtData[0][32*1-1:32*0]
                                                        :mem_AM[DMA_AM_WrtAddr[0][23:2]+0];
mem_AM[DMA_AM_WrtAddr[0][23:2]+1]= AM_WrtMask[0][4] ?  DMA_AM_WrtData[0][32*2-1:32*1]
                                                        :mem_AM[DMA_AM_WrtAddr[0][23:2]+1];
mem_AM[DMA_AM_WrtAddr[0][23:2]+2]= AM_WrtMask[0][8] ?  DMA_AM_WrtData[0][32*3-1:32*2]
                                                        :mem_AM[DMA_AM_WrtAddr[0][23:2]+2];
…
mem_AM[DMA_AM_WrtAddr[0][23:2]+5]= AM_WrtMask[0][20] ? DMA_AM_WrtData[0][32*6-1:32*5]
                                                        :mem_AM[DMA_AM_WrtAddr[0][23:2]+5];
mem_AM[DMA_AM_WrtAddr[0][23:2]+6]= AM_WrtMask[0][24] ? DMA_AM_WrtData[0][32*7-1:32*6]
                                                        :mem_AM[DMA_AM_WrtAddr[0][23:2]+6];
mem_AM[DMA_AM_WrtAddr[0][23:2]+7]= AM_WrtMask[0][28] ? DMA_AM_WrtData[0][32*8-1:32*7]
                                                        :mem_AM[DMA_AM_WrtAddr[0][23:2]+7];
  end
end
```

图 4.19 输出数据处理部分代码

4.3.3 reference_model1 的构建

矩阵块到块传输和二维矩阵转置传输这两种传输模式在功能上的主要区别是块到块传输直接将要搬移的数据矩阵按照配置的目的行计数和单元数要求输出到目的设备当中，而转置传输则是根据配置的要求对矩阵进行一个转置操作之后再输出到目的设备当中，这种差别使得模型在对输入数据文本进行打印处理时也存在差异。图 4.20 为模型一中两种传输方式下输入数据的打印处理顺序，其中①表示按照矩阵行顺序打印块到块传输输入数据，②表示按照矩阵列顺序打印矩阵转置传输。

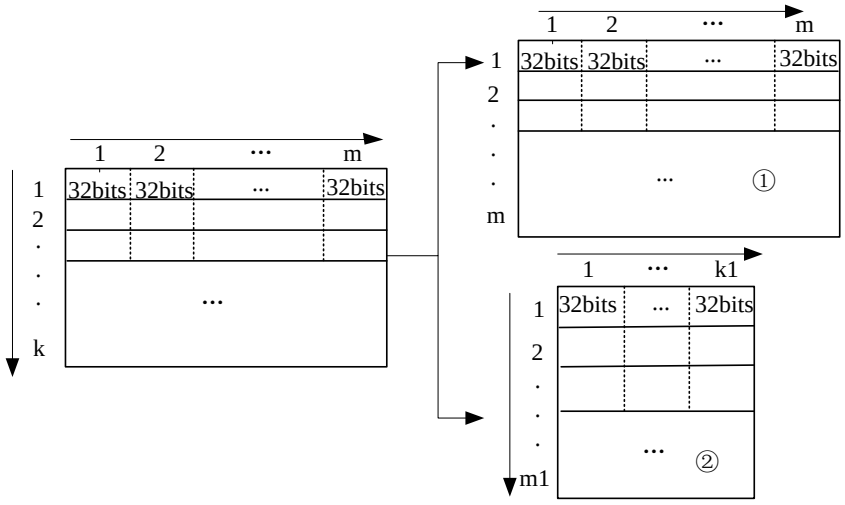

图 4.20 模型一中输入数据处理顺序

根据块到块传输输入数据处理示意图，按照输入矩阵行的顺序先将第一行的

数据按照地址顺序取出，然后再顺序取出第二行的数据，以此类推直到完成整个输入矩阵的数据打印，通过这种数据打印方法来模拟出 DMA 进行矩阵块到块传输的最终预期值。同理对于矩阵转置来说，按照输入矩阵的列顺序先将第一列的数据取出按照行顺序摆放，然后再顺序取出第二列的数据摆放到第二行，直到完成整个输入矩阵的取数就可以模拟出矩阵转置的最终预期值。

在输出数据的处理上，由于 DMA 的输出都是 DMA 按照各自传输方式对输入进行变化后再输出到目的设备的，因此在对输出数据进行打印时直接按照地址顺序取数即可完成预期值的打印。

4.3.4 reference_model2 的构建

矩阵分块传输和矩阵多核传输相比于上面两种传输方式会显得更加复杂，这两种传输方式在输出的目的设备上会涉及到两个核的核内存储，因此在构建黄金模型时需要对两个核的 DMA 进行模拟。矩阵多核传输的特点是通过一个核的 DMA 将源设备中一块数据矩阵搬移到两个核的核内存储中，这在本质上可以看成是两个 DMA 对同一块数据矩阵进行矩阵块到块传输。矩阵分块传输则是通过一个核的 DMA 将数据矩阵的不同数据块搬移到两个核的核内存储当中，每个核会根据配置信息得到不同的矩阵块。图 4.21 为模型二中两种传输方式下输入数据的打印处理顺序，图中①表示按矩阵多核传输特点处理输入数据，图②表示按照矩阵分块传输特点处理输入数据。

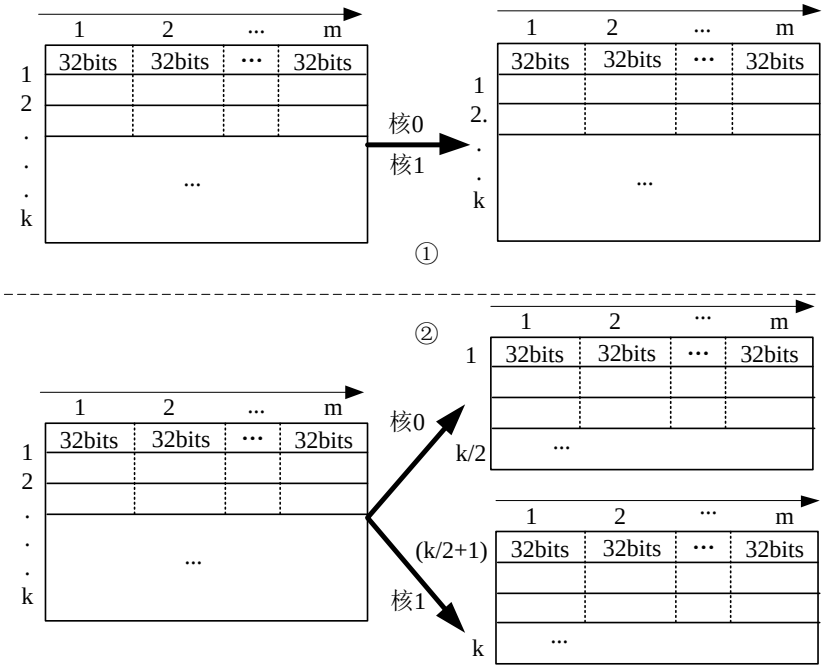

图 4.21 模型二中单核输入数据处理

矩阵分块传输和矩阵多核传输虽然在数据搬移过程中涉及到两个核的核内存

储，但是就单核来说其数据搬移相当于一次矩阵块到块传输，因此在对输入数据进行处理时每个核按照块到块传输的方式将数据以矩阵行的顺序取出即可。根据矩阵多核传输的特点，在模型中只需要按照矩阵行完成整个输入的数据打印即可得到其预期值。对于矩阵分块传输来说，由于各个核会得到输入矩阵的不同部分，故还需要根据配置激励分离出各个核的有效数据再进行打印来完成预期值的模拟。

reference_model2 中输出数据的处理方法和 reference_model1 中是一致的，都是根据计算的目的地址信息从模拟的目的设备存储器中取出有效数据得出预期值，只是相对于 reference_model1 来说，reference_model2 需要处理两个核 DMA 输出端口的数据。

4.4 比对策略

为了保证平台验证的充分性，在对 DMA 的输出值与预期值进行判断上采用了两种比对机制，第一种是通过黄金模型模拟输入之后的预期值与输出值进行比对，第二种是通过 UVM 平台监视器监视的输入值和输出值分别与黄金模型模拟的输入和输出值进行比较。图 4.22 给出了两种机制联合比对结构图，其中模型比对 compare 对应着第一种比对机制，通过这种比对机制不仅可以保证 DMA 完成了整个配置数据矩阵的搬移，同时也可以确定其传输的正确性。输入比对和输出比对组成了第二种比对机制 scoreboard，通过这种机制的比对可以确保 DMA 输入端口和输出端口数据的正确性，方便比对出错时快速定位输入端口或者是输出端口的问题。若是两种比对机制都比对通过，则说明通过 UVM 监视器得到的 DMA 端口数据和黄金模型模拟出的预期值数据完全一致，DMA 输入数据组成的矩阵是按照参数项配置要求变换并完成数据输出。

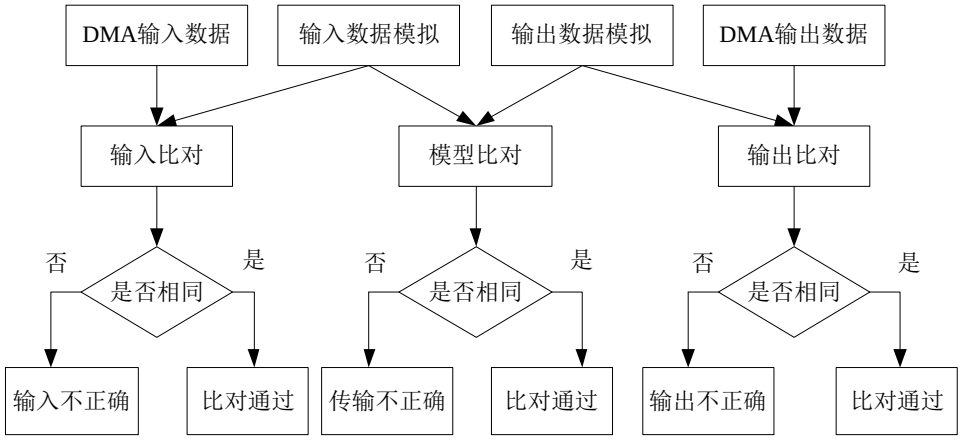

图 4.22 平台联合比对结构图

4.5 平台调试

UVM 验证平台构建完成之后，需要根据 DMA 的功能点对平台进行调试以保证验证环境的正确性，验证环境的正确可靠是进行 DMA 功能验证的前提条件。为了确保整个验证环境符合要求，本文按照 4.1 小节中的平台结构划分分别对 UVM 环境组件和 DMA 黄金模型的功能正确性进行调试。考虑到调试工作的复杂性和重要性，首先构建一个基于 Verilog 的调试环境来保证 DMA 黄金模型的正确性，然后将黄金模型添加到 UVM 环境组件当中，完成对 UVM 环境组件的调试。

4.5.1 DMA 黄金模型

黄金模型模拟的是 DMA 数据搬移结果的预期值，预期值的正确性关系到最后 DMA 功能验证结果的正确性，因此保证黄金模型功能可靠是整个调试工作的重点和难点，图 4.23 给出了对黄金模型进行调试的结构图。

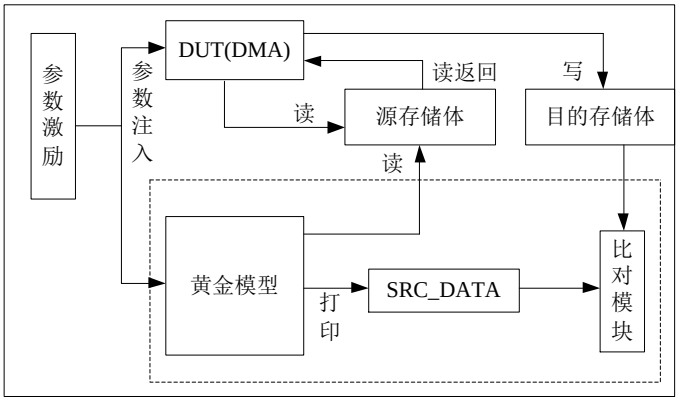

图 4.23 黄金模型调试结构图

在上面的调试环境中，参数激励的注入是通过 Verilog 语言来实现的，参数来源于人工手写的定向激励，增加了调试工作的可控性。同时，黄金模型具有良好的可重用性，很方便将其集成到调试环境当中。DMA 完成配置并启动后，从源存储体中取得读返回数据并完成数据输出，最后将输出的数据保存到目的存储体当中，数据保存操作使用了 4.3.2 节介绍的输出数据处理方法。与此同时，黄金模型对参数激励进行解析，然后根据配置信息将源存储中要搬移的数据矩阵完成相应的输入变化之后打印到SRC_DATA 文本当中，最后由比对模块完成黄金模型预期值和 DMA 输出数据的自动比对。若比对结果正确，则说明黄金模型预期功能是正确的，否则，需要不断的对模型中各模块进行调试直到满足功能要求。

根据上面的调试环境，以 DMA 的四种传输方式为基本功能点，编写大量可控的参数激励对黄金模型进行调试。图 4.24 给出了四种传输模式下黄金模型的调试结果，其中①、②、③、④分别对应着块到块传输、矩阵多核传输、矩阵转置传

输以及矩阵分块传输这四种数据搬移方式，保证了模型功能的正确性。

```
//////////ptp//////////                    ①
4735 run_count0= 1，DMA0   pass!

////////////// Broadcast//////////         ②
 5398run_cnt=         1
 5402 DMA broadcast  pass!

//////////trasnspot/////////               ③
18771 run_transport0=   6，DMA0   pass!

//////////mutlti_broadcast//////           ④
 22520run_cnt=         2
 22525 DMA  mutlti_broadcast pass!
```

图 4.24 黄金模型调试结果

4.5.2 UVM 组件调试

UVM 环境中各个组件的结构和功能在 4.2 小节进行了详细的介绍，UVM 环境中各组件功能是相对独立的，对组件进行构建的同时即可完成其功能的调试，图 4.25 给出了 UVM 环境中组件调试顺序。

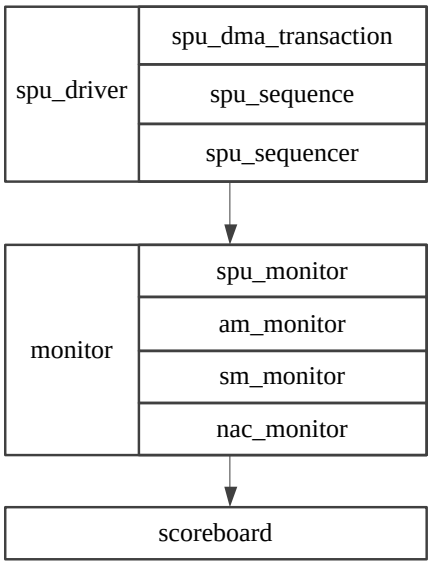

图 4.25 UVM 组件调试顺序

从驱动器 spu_driver 开始调试，然后到各个端口监视器 monitor，最后进行记分板 scoreboard 组件的调试。根据 spu_driver 负责驱动激励和管理平台运行的功能特性，在对 spu_driver 进行调试时必然会涉及到事务级激励 spu_dma_transaction 以及 sequence 机制的正确实现，以保证 DMA 完成配置并启动，因此这个调试过程是保证 DMA 能够正常运行的前提。在进行监视器 monitor 的调试时，只需要查看各个部件对应的监视器收集到的数据是否和 DMA 各端口流过的数据相等即可。记分板 scoreboard 的调试是最后进行的，根据 4.4 节中的比对策略，scoreboard 需

要完成 DMA 输入及输出数据和黄金模型模拟的数据比对。将正确的黄金模型添加到 UVM 组件环境中，保持 scoreboard 的比对结果和黄金模型比对结果一致，就可以保证 UVM 组件环境的正确性。

4.6 本章小结

本章内容是整个课题研究的核心部分，对平台的构建进行了详细说明。首先结合 UVM 方法学的思想和 DMA 的功能特点给出了验证环境的整体框架，随后对平台中各个功能模块的建模过程，特别是 UVM 的基本组件和黄金模型的实现进行了详细的分析和说明，最后介绍了整个环境的调试方法，保证了平台的正确性。

第五章　验证结果及覆盖率

在上面章节中详细介绍了 UVM 平台的实现以及使用平台对 DMA 功能点进行验证的方法，下面将通过仿真工具对 DMA 功能进行仿真验证，并给出基于 UVM 平台的验证结果说明以及 DMA 覆盖率分析，最后对平台性能进行了评估分析。

5.1　测试用例

测试用例是根据待验证部件的验证计划分析提炼出来的，越复杂的部件其对应的测试用例个数也越多。根据第三章描述的 DMA 验证计划，以 DMA 四种传输方式为基本功能点载体，提炼出四种基本的测试用例场景：矩阵块到块传输、矩阵转置传输、矩阵分块传输以及矩阵多核传输。完成上面四种基本用例的提炼之后，以此为基础再进行更为复杂功能点的提炼和规划。表 5-1 对提炼出的测试用例及其对应场景进行了说明。

表 5-1 测试用例说明列表

测试用例	用例名称	场景描述
my_case1	块到块传输	采用块到块传输方式进行不同数据搬移方向、不同数据量、不同地址变化模式等组合情况的验证
my_case2	矩阵转置传输	采用矩阵转置传输方式进行不同数据搬移方向、不同数据量、不同地址变化模式等组合情况的验证
my_case3	矩阵分块传输	采用矩阵分块传输方式进行不同数据搬移方向、不同数据量、不同地址变化模式等组合情况的验证
my_case4	矩阵多核传输	采用矩阵多核传输方式进行不同数据搬移方向、不同数据量、不同地址变化模式等组合情况的验证
my_case5	单核单通道传输	单核 DMA 两个通道分别进行数据搬移，不同物理通道分别完成四种基本用例测试
my_case6	单核双通道传输	单核 DMA 同时用两个物理通道进行数据搬移，不同物理通道采用的传输方式随机组合
my_case7	双核传输	启动两个核的 DMA 进行数据搬移，不同核采用的传输方式随机组合
my_case8	双核双通道	启动两个核的 DMA 进行数据搬移，两个核分别启动两个物理通道进行数据搬移
my_case9	传输异常	根据 DMA 异常寄存器不同位所代表的含义，分别验证 DMA 两个传输通道的异常情况

对于 DMA 来说，每个测试用例不仅包含其本身所带的验证任务特性还必须是一次完整的数据搬移过程，因此 DMA 所有功能点的规划都是建立在四种基本

的传输方式上。四种基本传输方式所形成的测试功能点中包含着很多验证计划中所提到的功能需求，例如不同数据搬移方向、不同搬移数据量、不同的地址空间等等小功能点。然后再以此四个测试用例为基础根据验证计划制定出更加复杂的测试功能点，例如双通道传输和双核传输这两个功能点中所涉及到的组合方式则是以不同的传输方式为基础的，这样不仅保证了不同传输模式组合能够验证到，同时这些基本的传输模式中也会涉及到更加细致的功能需求的验证。

在使用 UVM 平台对 DMA 进行功能验证时，按照验证的顺序首先需要对 UVM 平台进行完备性的调试，完成调试之后才能进行大规模的验证。因此在根据测试功能点准备平台所需要的测试用例时，需要构造两种测试功能点用例以满足平台的需求，第一种是简单的调试测试用例，第二种是复杂的随机验证测试用例。

调试用例负责在验证的初期对整个 UVM 平台环境进行调试，为了方便平台的排错和 DMA 数据搬移行为的观察，调试用例所产生的都是一些很简单的参数激励。随机验证用例则是在整个 UVM 验证环境调试稳定之后才开始使用的大规模测试激励，随机用例中所产生的参数激励应该尽可能复杂以便在验证过程中暴露出更多的设计缺陷。

5.2 DMA 仿真结果

UVM 验证平台搭建完成之后，根据 NC-Verilog 仿真器的使用方法对整个仿真环境进行配置，首先使用简单的调试测试激励来排除 UVM 平台中存在的错误，然后根据随机测试用例对 DMA 进行大规模的功能点验证。整个平台环境是按照分层次设计的思想来搭建的，这种层次结构的划分参考了 DMA 进行数据搬移的顺序，因此在对其进行仿真调试时也是按照顺序一步一步完成的。在使用 NC-Verilog 仿真时主要通过两种手段来进行结果的分析和判断，第一种是根据平台中打印出的各种记事信息来分析仿真结果，第二种是通过仿真的波形信息直接观察平台和 DMA 中各个端口的变化。整个验证环境中 UVM 平台部分的调试是通过查看记事信息来完成的，黄金模型和 DMA 部分则主要是通过仿真波形直接查看端口信号的变化来完成。

5.2.1 UVM 平台信息

NC-Verilog 完成一次仿真之后，会按照 UVM 平台各个组件中设置的信息打印行为将结果输入到仿真器的图形界面上。在各个组件中除了会控制一些基本的运行状态信息的打印之外，还会根据需要将一些重要的事件以及数据信息打印输出到终端中。在本验证平台中主要有三类记事信息将会被显示到终端当中，第一

个是各个组件的中基本信息的输出，第二类是个别组件中重点数据信息输出，第三类是全部仿真完成之后的信息统计，下面将分别展示三类仿真输出信息的结果。

一、基本信息输出

图 5.1 所示为平台各组件仿真结束后基本信息输出结果，通过这些记事日志可以知道平台运行的进度以及各个组件运行的状态等信息。图中第一条信息是整个报告的开始，指明了正在执行的测试用例是 my_case0 并完成其实例的生成。接下来平台按照 UVM 树形的结构依次执行各个组件中的 build_phase，再接着按顺序执行树中各个结点上的 main_phase 等任务完成各自的功能直到仿真结束。平台中这种按照顺序执行任务的流程在输出的记事报告中也得到了体现，因此通过分析报告中信息可以很快的定位平台运行的状态和进度，方便对整个验证环境进行调试。

```
UVM_INFO @ 0: reporter [RNTST] Running test my_case0...
UVM_INFO dma_env.sv(18) @ 0: uvm_test_top.env [dma_am_env] build_phase
UVM_INFO am_monitor.sv(20) @ 0: uvm_test_top.env.am [am_model] new is called
UVM_INFO nac_monitor.sv(21) @ 0: uvm_test_top.env.nac [nac_model] new is called
UVM_INFO spu_driver.sv(14) @ 0: uvm_test_top.env.agt.spu [spu_driver] spu_driver is called
UVM_INFO spu_monitor.sv(11) @ 0: uvm_test_top.env.agt.spu_m [spu_monitor] spu_monitor is called
UVM_INFO spu_driver.sv(21) @ 0: uvm_test_top.env.agt.spu [spu_driver] build_phase is called
UVM_INFO spu_monitor.sv(17) @ 0: uvm_test_top.env.agt.spu_m [spu_monitor] build_phase is called
UVM_INFO spu_driver.sv(165) @ 0: uvm_test_top.env.agt.spu [spu_driver] main_phase
UVM_INFO spu_driver.sv(40) @ 20500: uvm_test_top.env.agt.spu [spu_driver] reset_phase
UVM_INFO spu_driver.sv(40) @ 20500: uvm_test_top.env.agt.spu [spu_driver] reset_phase
```

图 5.1 各组件基本输出信息

二、数据信息输出

平台各个组件当中除了一些基本信息需要输出外，验证人员还需要对一些重要的数据信息有一个直观的了解。例如配置激励监视器完成对 DMA 输入参数激励的收集之后需要将其打印输出到记事报告当中，图 5.2 所示为配置参数激励的数据输出格式。输出的数据中包含了通过 SPU 配置给 DMA 的参数项和启动信息，验证人员可以通过其分析出此次 DMA 数据搬移的方向、采用的传输方式以及使用的参数通道等传输信息。通过对图中打印的参数激励进行解析，DMA 即将采用矩阵块到块传输将核外 DDR 中 29KB 大小的数据量搬移到核内 AM 存储设备当中，其中源设备 DDR 的数据初始地址是 8_0004_8978，数据存储到目的设备 AM 中的初始地址是 0_4000_0000。

每一个这样的事务数据格式代表着一个数据包，考虑到在进行 DMA 全芯片验证时所需要提供的汇编激励和数据包格式类似，只需要将数据包转化成系统能够识别的汇编指令即可完成配置，因此这些参数信息可以很好的复用到 DMA 芯片级验证当中。

```
--------------------------------------
Name       Type           Size  Value
--------------------------------------
spu_data   spu_dma_data   -     @33562760
  cier     integral       32    'hffffffff
  para1    integral       32    'h80005016
  para2    integral       32    'h0
  para3    integral       32    'h48978
  para4    integral       32    'h380082
  para5    integral       32    'h40000000
  para6    integral       32    'h5e004e
  para7    integral       32    'h10
  para8    integral       32    'h0
  prir     integral       32    'h1
  esr      integral       32    'h10
  cipr     integral       32    'hffffffff
--------------------------------------
```

图 5.2 配置参数激励输出格式

三、 仿真结束信息统计

UVM 平台根据一个功能点用例设置的循环次数不停进行仿真验证，等待平台所有组件中的 phase 执行完毕后仿真器会结束仿真，并统计所有的输出记事信息形成一个最终的报告，如图 5.3 所示为仿真结束之后的总结记事报告。报告会统计此次测试用例中出现的所有信息类型，一旦其中出现了 UVM_ERROR 类型或者是 UVM_FATAL 类型则说明平台或者设计中存在错误需要进行调试和修改，若统计的信息类型中无上面两项，则表示此次仿真验证的功能点测试通过。

```
TEST CASE PASSED
--- UVM Report catcher Summary ---
Number of demoted UVM_FATAL reports :  0
Number of demoted UVM_ERROR reports :  0
Number of demoted UVM_WARNING reports:  0
Number of caught UVM_FATAL reports :  0
Number of caught UVM_ERROR reports :  0
Number of caught UVM_WARNING reports :  0
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO : 4010
UVM_WARNING :  0
UVM_ERROR :  0
UVM_FATAL :  0
```

图 5.3 仿真信息汇总

5.2.2 DMA 仿真波形

对于仿真波形的观察和分析在整个功能验证过程中占据了很重要的地位，不同的测试激励作用到 DMA 之后其真实的运作场景都会丝毫不差的体现在仿真的波形上面，因此对于波形的分析过程即等同于对 DMA 的逻辑进行一次解析。DMA 完成一次数据搬移任务可以大致分为三个步骤，第一步是完成参数项的配

置并启动 DMA，第二步是发出读请求信息并接收源设备返回的数据，第三步是将返回的数据按照要求作出相应的变换之后将其写出到目的设备当中，下面将按照这三个步骤通过波形分析 DMA 的完成数据搬移的具体实现。

1)第一步：配置并启动 DMA

DMA 的激励配置是通过 UVM 平台的驱动器组件 spu_driver 来驱动的，其配置端口的数据波形如图 5.4 所示，通过观察波形中端口得到的数据与配置激励监视器打印输出的数据一致性来判断驱动是否成功。

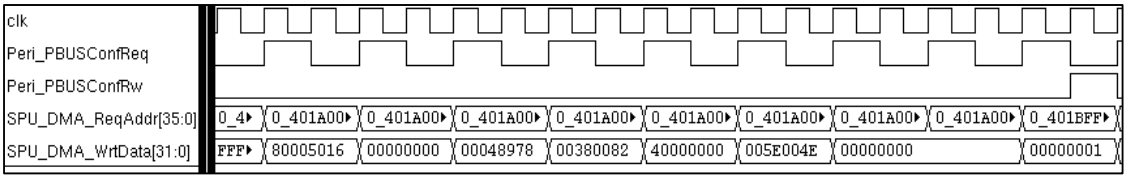

图 5.4 DMA 配置端口波形图

2)第二步：读请求与数据返回

DMA 正常启动之后会根据配置激励向源设备发出读请求信息，根据 5.2.1 节对配置参数的分析，DMA 会向核外的存储设备 DDR 发出读请求并且数据读取的初始地址是 8_0004_8978，图 5.5 所示为 DMA 读请求端口输出波形图。根据 DMA 读请求的计算规则，其读地址的变化和每次读取的有效数据个数保持着固定的规律，即每个有效字对应的占据着 4Byte 地址。在下面波形图中 XX_RdMask 信号表示每次请求读取的有效数据个数，XX_RdAddr 表示每次取出数据的存储地址，当信号 XX_RdMask 中有 2 个 F 时表示本次请求会读取 2 个有效字，相应的其下一次读地址会在本次地址的基础上增加两个字的地址空间大小。

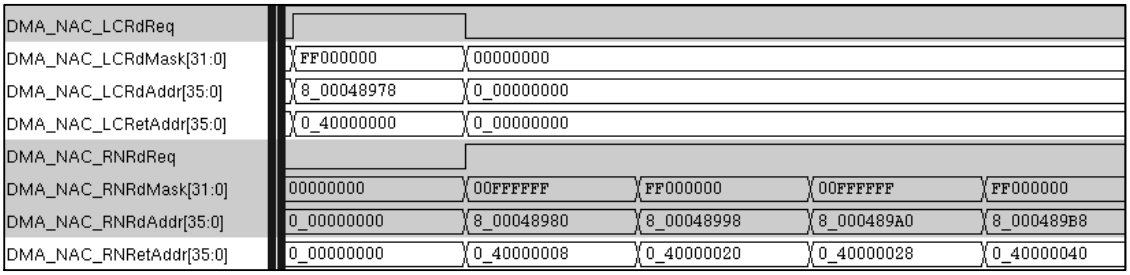

图 5.5 DMA 读请求端口输出波形图

源设备接收到读数据请求之后会根据地址等信息将对应的数据返回给 DMA 的输入端口，图 5.6 所示为 DMA 数据返回端口的输入波形图。当读取的源设备为核外存储时，取出的数据会由三条通路返回到 DMA 的输入端口上，在对返回数据端口的波形进行分析时最重要的是保证在读取的地址空间上返回的数据有效个数和读请求中需要的有效数据个数应该保持一致。例如图 5.5 中所示，当读取的存储地址为 8_00048980 时其对应的数据有效个数为 6 个，在图 5.6 中相应的该地址返回的数据有效个数也应该为 6 个。

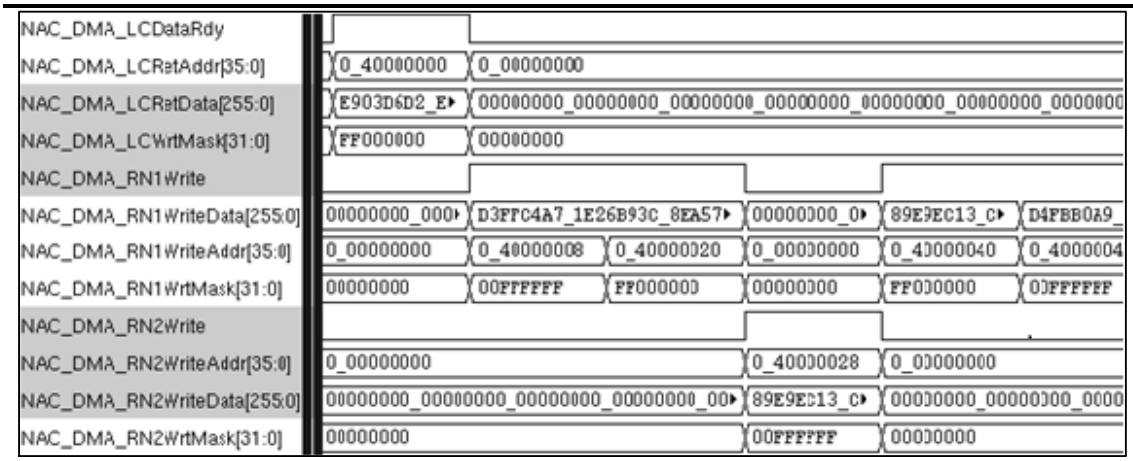

图 5.6 数据返回端口输入波形图

3)第三步：数据写出到目的设备中

读取的有效数据输入到 DMA 之后，DMA 会根据传输方式的要求对数据矩阵做出特定的变换，然后将其输出到目的设备对应的地址当中。图 5.7 所示为变换之后数据矩阵输出到目的地址的波形图，根据 DMA 矩阵块到块传输的特点返回的有效数据会根据配置目的地址要求直接写出到目的设备当中。

图 5.7 数据矩阵输出波形图

5.2.3 平台比对结果

DMA 完成一次配置激励的传输任务之后，平台中的数据比对模块就会根据各自的工作原理对输入和输出的数据进行正确性校验，这种比较方式在某种程度上代替了通过肉眼观察仿真波形来判断 DMA 的传输是否正确。比对校验模块最终的数据处理结果只有两种情况，一种是数据比对通过仿真器继续运行，另一种是数据比对出错停止仿真。表 5-2 所示为比对结束后两种检验结果说明以及处理方式，对于校验成功的激励只需要等待仿真结束即可完成一个测试用例的验证，一旦检验失败则需要根据打印出的错误信息并结合波形首先查看 DMA 设计是否有问题，当排除掉 DMA 设计问题之后再仔细分析 UVM 的平台定位出错的位置。

表 5-2 比对校验结果说明

校验结果	结果说明	处理方式
校验成功	DMA 传输数据比对通过	仿真器继续仿真直到平台执行完所有的 phase 后结束仿真
检验失败	DMA 传输数据比对不通过	结合错误信息和仿真波形分析错误原因

5.3 覆盖率分析

传统芯片验证过程中，对于验证完备性的保证主要依赖于验证人员的测试经验。随着芯片的设计规模及功能复杂性程度不断增加，这种仅仅依靠人工经验来判断验证完备性的方法显示是不可靠的，因此随着验证方法学的发展以覆盖率[39]为标准来判断验证完备性的方法占据了验证工作中的主导地位。

根据覆盖率描述对象的不同，在验证过程中涉及到的常用覆盖率类型主要有功能覆盖率、数据流覆盖率以及代码覆盖率等，不同的覆盖率类型对应到设计当中所体现的功能也不同，因此在对验证完备性进行评估时需要结合不同类型的覆盖率综合分析。本文在对 DMA 验证完备性进行判断时，综合分析考虑了代码覆盖率和功能覆盖率[40]，在整个验证过程中二者相辅相成，既保证了验证进度同时也保证了验证的充分性。图 5.8 所示为验证进行的进度和覆盖率之间的关系说明图，在验证刚开始的阶段主要是通过验证平台根据测试用例对 DMA 进行大规模的测试以便能发现设计中的缺陷，此时对于覆盖率的收集意义不大。当验证进行到一定的程度时可以对覆盖率进行收集，分析未达到覆盖率要求的原因，通过增加随机测试用例或者改变约束等方法来加快验证的收敛速度，直到最终功能覆盖率和代码覆盖率均到达预订的目标值。

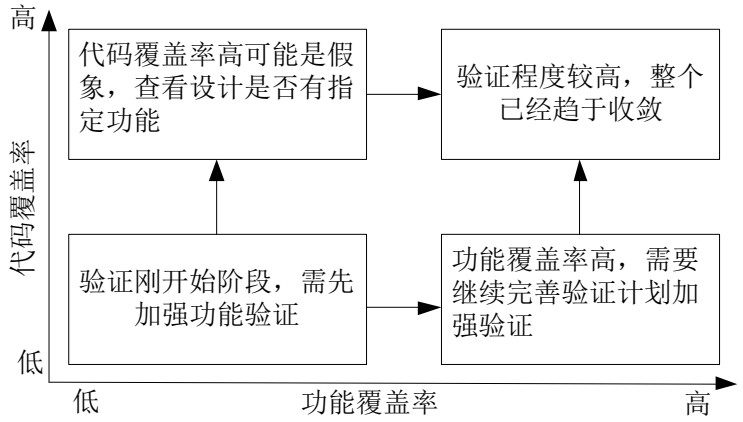

图 5.8 验证进度与覆盖率关系说明

5.3.1 功能覆盖率

功能覆盖率是用来检查待验证部件的行为是否满足设计需求中所要求的全部功能，在对覆盖率进行收集时需要根据验证计划中罗列的测试点自定义收集条件，构建出自动收集功能覆盖率的规范模型。

在根据 DMA 的验证计划构建功能覆盖率收集规范模型时，可以按照 5.1 小节提炼出的测试用例为基准列出需要采集数据的覆盖点，然后根据覆盖点的数据来

源对覆盖率收集的规范模型进行分类处理。DMA 参数项中包含了完成传输任务需要的各种信息，这些配置信息分别对应着验证计划中不同的功能点，因此在事务级激励（spu_dma_transaction）中可以直接对数据搬移方向、地址增减方式、不同传输方式等覆盖点进行收集。对于 spu_dma_transaction 中不能进行数据收集的情况，比如地址偏量的组合、寄存器值变化等覆盖点则需要在平台与 DMA 监视器接口组件中进行收集。

5.3.1.1 覆盖点设置

功能覆盖率的模型是由一个个覆盖点组成的，对于模型中不同覆盖点的数据收集方式也存在差别，下面将就 DMA 中四个覆盖点来说明其编写方式。其中 dma_para1 覆盖点对应着 DMA 四种传输方式，src_addr_offset 对应着源地址偏量，dst_addr_offset 对应着目的地址偏量，addr_offset 对应着源目的地址偏量组合方式：

1) dma_para1 覆盖点

根据 3.1.2 节中关于 DMA 配置参数的介绍，参数字 1 中 tmode 取值 0、1、2、3 分别指示 DMA 四种传输方式，图 5.9 给出了覆盖点 dma_para1 数据收集方式。在收集 dma_para1 覆盖点数据时，通过 bins 设置四个独立的仓 p2p、matrix、multicast、broadcast 来分别记录 tmode 出现 0、1、2、3 四种取值的次数。

```
//////// dma_para1/////////////////DMA传输方式
dma_para1: coverpoint para1.tmode {
                        bins p2p =       {2'h0};
                        bins matrix =    {2'h1};
                        bins multicast = {2'h2};
                        bins broadcast = {2'h3};
                      }
```

图 5.9 dma_para1 数据收集

2) src_addr_offset 覆盖点

DMA 数据总线位宽是 256 位的，输入和输出总线数据位宽决定了 DMA 一次最多读入或者写出 8 个有效单元字。图 5.10 给出了源地址偏量 src_addr_offset 数据收集方式，一个单元字占据着 4byte 地址，因此取源地址 para3[4:2]表示传输单元字的偏量情况，src_addr_offset[]会自动生成 8 个仓来分别收集地址偏量出现 0~7 的次数。

```
////////src_addr_offset/////////////////源地址偏量
    src_addr_offset: coverpoint para3[4:2] {
            bins src_addr_offset[] = { [0:7] };
    }
```

图 5.10 src_addr_offset 数据收集

3) dst_addr_offset 覆盖点

目的地址偏量覆盖点 dst_addr_offset 的数据收集情况和源地址类似，图 5.11 给出了 dst_addr_offset 数据收集方式。数据收集时取目的地址 para5[4:2]表示地址偏量，为了方便分析地址偏量组合覆盖情况，通过 bings 分别设置 8 个仓来对应收集地址偏量 0~7 出现次数。

```
//////// dst_addr_offset/////////////////目的地址偏量
    dst_addr_offset: coverpoint para5[4:2] {
                bins dst_addr_offset_0 = { [0:7] };
                bins dst_addr_offset_1= { [0:7]};
                bins dst_addr_offset_2 = { [0:7] };
                bins dst_addr_offset_3= { [0:7]};
                bins dst_addr_offset_4 = { [0:7] };
                bins dst_addr_offset_5= { [0:7]};
                bins dst_addr_offset_6 = { [0:7] };
                bins dst_addr_offset_7= { [0:7]}; ;
    }
```

图 5.11 dst_addr_offset 数据收集

4) addr_offset 覆盖点

DMA 进行传输任务时，对于地址之间偏量不同组合情况的处理方式也存在着差异，图 5.12 给出了地址偏量组合 addr_offset 的数据收集方式。通过 cross 来实现源偏量 src_addr_offset 和目的偏量 dst_addr_offset 的组合情况，在使用 cross 之前两种地址偏量必须已经完成定义。

```
//////////addr_offset/////////////////地址偏量组合
//////////////////src_addr_offse源偏量，dst_addr_offse目的偏量////////
addr_offset:   cross  src_addr_offse,  dst_addr_offset;
```

图 5.12 addr_offset 数据收集

5.3.1.2 结果分析

覆盖率的收集必须在激励验证通过的情况才有效，对于验证不通过的测试激励收集覆盖率是没有意义的，因此覆盖率收集的时间点应该是在平台比对成功之后。根据覆盖率采集时间一致性，本文将所有覆盖点集成到一个覆盖组中进行规范模型的构建，图 5.13 为覆盖组数据收集部分代码，cg 中里面包含了 DMA 所有功能覆盖点以及覆盖点之间交叉组合数据收集。

```
Covergroup   cg;
//////// /DMA传输方式////////
dma_para1: coverpoint para1.tmode {
                              bins p2p =       {2'h0};
                              bins matrix =    {2'h1};
                              bins multicast = {2'h2};
                              bins broadcast = {2'h3};
                             }
//////// 地址偏量//////////////
Src_addr_offset: coverpoint  para3[4:2] {
                bins src_addr_offset[] = { [0:7] };
     }
…
//////////////////组合////////
 cnt_offset:   cross Src_addr_offset,  Dst_addr_offset;
…
endgroup
```

图 5.13 覆盖组数据收集

完成对功能覆盖率规范建模之后，将其添加到 UVM 验证平台中，在对 DMA 不同功能进行仿真验证时，仿真器 VCS 会根据覆盖点定义的收集条件，自动完成覆盖率的统计。为方便对生成的覆盖率进行分析，本文中数据报告以 htlm 的文件形式输出，根据此报告可以直观的查看各个覆盖点命中与缺失的情况。图 5.14 给出了 DMA 覆盖率收集情况，其中总覆盖率为 97.63%由所有单个覆盖点以及交叉覆盖点组成，通过查看覆盖点数据收集情况分析其未满足覆盖条件的原因并改变事务激励生成的约束条件加强验证。

Group : dma_pkg::spu_mon_sb_trans::cg

SCORE	WEIGHT	GOAL
97.63	1	100

Summary for Group dma_pkg::spu_mon_sb_trans::cg

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	141	2	139	99.08
Crosses	2947	189	2758	95.08

图 5.14 DMA 功能覆盖收集报告

报告还给出了单个覆盖点以及交叉覆盖点具体覆盖率情况，根据报告可以查看各个功能点的覆盖情况是否满足规范模型中各个覆盖点的数据收集。下面将按照 5.3.1.1 中提到的四种覆盖点顺序分别给出其覆盖结果。

图 5.15 给出了 DMA 四种传输方式的覆盖结果，dma_para1 覆盖点中设置的四

个独立仓 p2p、matrix、multicast、broadcast 都收集到了数据，表示 DMA 在验证过程中完成了四种传输方式的数据搬移。

Summary for Variable dma_para1

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	4	0	4	100.00

User Defined Bins for dma_para1

Bins

NAME	COUNT	AT LEAST
broadcast	10	1
multicast	20	1
matrix	10	1
p2p	10	1

图 5.15 DMA 传输方式

图 5.16 以及图 5.17 分别给出了 DMA 传输源和目的的地址偏量数据收集结果，源覆盖点 src_addr_offset 和目的覆盖点 dst_addr_offset 都完成了 8 种情况的数据收集，达到 100%覆盖。

Summary for Variable src_addr_offset

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	8	0	8	100.00

User Defined Bins for src_addr_offset

Bins

NAME	COUNT	AT LEAST
src_addr_offset_0	31	1

图 5.16 源地址偏量

Summary for Variable dst_addr_offset

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	8	0	8	100.00

User Defined Bins for dst_addr_offset

Bins

NAME	COUNT	AT LEAST
dst_addr_offset_0	5	1

图 5.17 目的地址偏量

由于源和目的地址偏量分别有 8 种情况，因此二者地址偏量组合情况为 64

种，图 5.18 给出了 DMA 传输 64 种地址偏量数据收集报告，地址偏量组合覆盖点 addr_offset 达到 100%覆盖。

Summary for Cross addr_offset

Samples crossed: src_addr_offset dst_addr_offset

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Automatically Generated Cross Bins	64	0	64	100.00

Automatically Generated Cross Bins for addr_offset

Bins

src_addr_offset	dst_addr_offset	COUNT	AT LEAST
src_addr_offset_0	dst_addr_offset_0	627	1
src_addr_offset_0	dst_addr_offset_1	663	1

图 5.18 地址偏量组合覆盖报告

对于功能覆盖率未满足的覆盖点需要着重分析不能覆盖的原因，图 5.19 给出了源矩阵单元数 src_element 覆盖点的数据收集报告，其中 src_element 为 0 的情况不能覆盖。由于 DMA 的矩阵单元数最小值为 1，因此当 src_element 为 0 时，DMA 传输处于异常状态，这种情况下是不进行覆盖率收集的，故不能满足覆盖条件。其中交叉覆盖率只有 95%，主要原因是 DMA 传输矩阵行计数和单元数组合的情况太多，需要增加仿真时间完成覆盖。

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	31	1	30	96.77

User Defined Bins for src_element

Uncovered bins

NAME	COUNT	AT LEAST	NUMBER
src_ele_0_0000	0	1	1

Covered bins

NAME	COUNT	AT LEAST
src_ele_1_0010	181	1

图 5.19 源单元计数覆盖报告

通过分析功能覆盖率，对于未达到覆盖要求的功能点可以通过增加仿真次数或者修改激励约束行为来增加提高覆盖率，这样有利于加快 DMA 仿真验证的收敛速度。表 5-3 给出了 DMA 所有测试用例对应的部分功能点覆盖情况，验证 my_case3 和 my_case4 测试用例时，由于 DMA 矩阵分块传输和矩阵多核传输方式下，数据搬移方向只能是核外向核内进行，因此收集到的数据搬移方向覆盖率只有 50%。其中主要是矩阵行计数和单元计数组合方式需要收集大量的数据，这个

阶段需要不断修改激励约束条件或者增加仿真时间来完成覆盖，最终完成 DMA 总的功能覆盖率为 100%。对于 my_case9 测试用例，可以在最后进行验证，其主要是完成 DMA 异常寄存器所指示的 12 种 DMA 异常情况的处理。

表 5-3 DMA 功能覆盖率统计

DMA 测试用例	地址偏量组合	数据搬移方向	源行计数和单元计数组合	目的行计数和单元计数组合	地址改变模式组合	地址空间覆盖	异常情况
my_case1	100%	100%	99%	99%	100%	100%	
my_case2	100%	100%	98%	100%	100%	100%	
my_case3	100%	50%	95%	95%	100%	100%	
my_case4	100%	50%	100%	100%	100%	100%	
my_case5	100%	100%	98%	98%	100%	100%	
my_case6	100%	100%	97%	97%	100%	100%	
my_case7	100%	100%	96%	98%	100%	100%	
my_case8	100%	100%	96%	98%	100%	100%	
my_case9							100%

5.3.2 代码覆盖率

代码覆盖率所检查的目标不是验证计划本身，而是针对 RTL 代码在进行仿真验证过程中有没有被执行到。通过对代码覆盖率的分析可以检测出设计的代码是否存在冗余，设计规范要求的要点是否全部实现。代码覆盖率的收集是通过仿真工具来实现的，仿真时所使用的工具不同，对于代码覆盖率的收集也存在着差异，本文中所采用的 NC-Verilog 仿真器主要支持分支块覆盖率、状态机覆盖率以及表达式覆盖率这三种类型的代码覆盖率。

通过 UVM 验证平台对 DMA 进行仿真验证时，当其功能覆盖率完成所有功能点的覆盖之后再开始分析 DMA 的代码覆盖率是否满足要求。针对三类覆盖率中确实无法满足覆盖条件的语句需要给出详细的分析说明，对于能够覆盖到的语句需要添加一定的定向激励或者增加功能覆盖点加强验证。图 5.20 所示为 DMA 主机模块对应的三种类型代码覆盖率情况，其中分支块覆盖率平均达到 97%以上，表达式覆盖率中除了 GREGIntf 模块外平均达到 95%以上，状态机覆盖率平均 90%以上。

图 5.20 DMA 主机代码覆盖率

对于各个模块中没有覆盖到的语句需要深入到模块内部分析其不能满足覆盖条件的具体原因，例如上面 GREGIntf 模块中表达式覆盖率只有 60%，其未覆盖的相关语句中有两个条件没有达到要求，图 5.21 所示为未满足覆盖要求语句的解析示意图。其中第一条显示不能覆盖的原因是：当 p1CLink_ER 等于 0 的时候，相对应的 Gpip1_Clink 也为 0，此时收集到的覆盖率情形是 T1=0 并且 T3=0 时，对于 T1=0 并且 T4=0 的情况没有收集；第二条显示不能覆盖的原因是：当 Gpip0_Clink 等于 1 的时候 p0CLink_ER 是不为 0 的，因此 T1=1 并且 T2=0 这个条件是不会成立的；

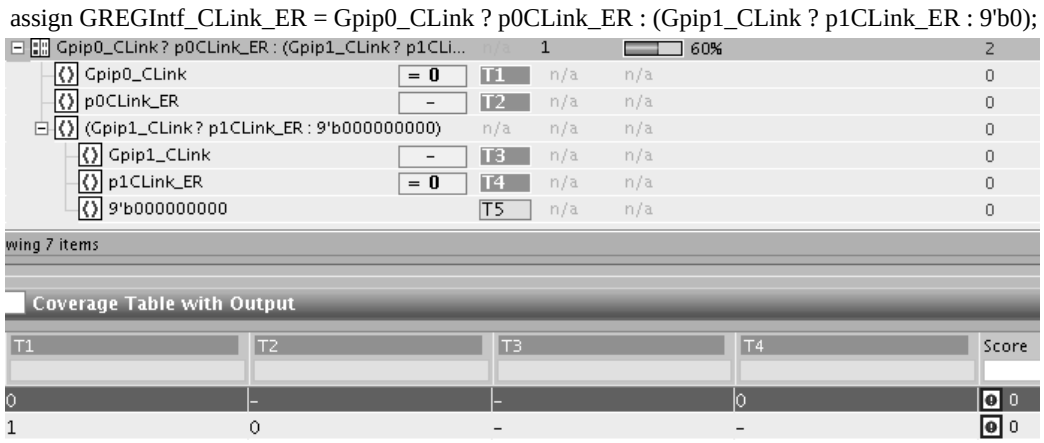

图 5.21 未满足覆盖要求语句示例

5.4 平台性能分析

本文运用 UVM 方法学的思想构建 DMA 验证环境，并根据 DMA 验证计划完成了所有功能点的验证。课题研究从 UVM 验证方法学以及 DMA 功能结构的学习

开始到UVM验证环境的构建，最后运用环境进行DMA功能验证并完成整个覆盖率的收集，整个研究过程经历了较长的时间周期。相比于传统验证方法，UVM方法学的应用明显提高了DMA验证效率，保证了验证的完备性，表5-4给出了实际工程研究中，应用UVM方法和传统方法之间的性能比较结果。其中UVM验证平台的构建以及使用平台完成DMA功能验证总用时为7个月，而采用传统验证方法构建平台并完成DMA的功能验证则需要13个月。

表5-4 UVM方法和传统方法比较

	UVM	传统验证
测试激励	自动随机	人工定向
验证类型	动态	静态
可重用性	重用性好	重用性差
平台维护	容易	困难
功能覆盖率	支持	不支持
代码覆盖率	支持	支持
平台建模时间	3个月	5个月
功能验证时间	4个月	8个月

UVM方法学中提供了各种强大的工作机制和树形管理模式，极大地方便了验证人员构建验证环境，同时，自动随机激励的提供使得验证速度有了质的提高，相对于传统验证，UVM方法将整个验证周期压缩了近一半的时间。UVM方法学还支持功能覆盖率的收集，很好的保证了DMA功能验证的完备性，整个验证环境还具有好的可重用性，为后续项目验证环境的建模提供了基础。

5.5 本章小结

本章主要说明了使用UVM平台对DMA功能点进行仿真验证的过程，并对验证结果进行了分析。在仿真验证之前，首先根据DMA的验证计划提炼出测试功能点，完成平台的调试和随机测试激励的编写，之后着重分析了仿真过程生成的记事报告和波形文件，最后以覆盖率驱动验证收敛性为手段完成验证，达到了课题研究的预期目标。

第六章　结束语

6.1　论文总结

本文的验证对象来源于国防科技大学自主研发的一款 DSP 中的 DMA 部件，通过分析目前国内外较为成熟的各种验证方法手段，提出了采用 UVM 平台的建模思想来构建 DMA 验证环境，并以此平台完成 DMA 模块级的功能验证。根据实际项目中的研究内容，本文详细介绍了 UVM 平台的构建过程和采用平台进行仿真验证的结果分析。在对整个课题进行研究时，主要完成了以下几点工作：

1. 在平台构建之前首先需要深入了解并掌握 SystemVerilog 这门建模语言，然后再以此为基础熟悉 UVM 验证方法学的思想以及 UVM 平台的构建方法，完成了课题研究所需要的知识储备。在基于 SystemVerilog 的三种验证方法学中，UVM 不仅具有 VMM 和 OVM 的优点还针对其缺陷进行了修正，因此采用 UVM 方法学构建的验证环境可以很好的满足 DMA 功能验证的需要。
2. 结合 DMA 的设计规范和 RTL 代码熟悉 DMA 各个功能点的具体实现，然后根据 DMA 的功能需求和特性编写出详细的验证计划以及验证策略。所谓磨刀不误砍柴工，一份优秀的验证计划和良好的验证策略可以帮助提高平台构建的效率和准确性。
3. 根据 UVM 验证方法学的思想，对 DMA 的验证环境进行建模，这是整个课题研究的重点也是难点部分。在对平台进行建模时首先根据验证策略构建出 UVM 平台的整体树形结构图，再按照 UVM 平台的构建方法逐步完成结构图中各组件的建模。平台构建完成之后根据验证计划提炼出测试功能点编写简单的调试用例对平台进行调试，保证平台各个组件以及整体功能的正确性。
4. UVM 验证环境调试完成之后，针对 DMA 各个测试功能点编写大规模测试用例，对 DMA 进行功能仿真验证。为了保证 DMA 验证的完备性，以功能覆盖率和代码覆盖率为判断验证完备性的性能指标，其中要求功能覆盖率达到 100%，代码覆盖率要达到 98%，对于代码覆盖率中未满足条件的语句还要给出不能覆盖的具体原因。

UVM 验证平台的使用不仅缩短了验证所花费的时间，同时对于 DMA 功能验证的完备性也有保证。平台根据功能点要求自动产生随机仿真激励，这种激励生成方式不仅高效准确而且也使得人工编写的定向激励更加有针对性。UVM 平台还具有很好的可重用性，这种重用性主要体现在两个方面：第一，平台生成的参

数配置激励和黄金模型通过简单的修改之后就可以完全复用到 DMA 的芯片级验证当中；第二，对于 DSP 中其他部件 UVM 验证平台的构建，本平台可以提供一个很好的模板，甚至于很多组件只需要小小的修改就可以使用。

6.2 未来展望

UVM 作为一门高级的验证方法学本身就包含了很多强大的功能机制，同时这些功能也在不断的扩大和改进，要想深入了解这门方法学并且熟练的使用它，没有一定的知识储备和时间积淀显然是不可能的。鉴于自身的能力以及时间限制，本文中 UVM 平台的建模仅仅涉及到了 UVM 方法学的一点皮毛知识，后续的工作将主要从下面几个方面加强对 UVM 方法学的研究，使能平台的功能更加完善更加强大。

1. 针对 DMA 部件中的寄存器构建寄存器模型，然后将模型集成到 UVM 验证平台当中，通过模型可以对 DMA 的寄存器进行读操作和写操作，大大简化了平台与 DMA 中寄存器的交互问题。
2. 为了使得 UVM 平台的可重用性功能更加的强大，UVM 方法学中还专门设置了 callback 机制，对于两个项目之间的差异部分可以通过 callback 函数来集成。不仅如此，在 DMA 部件还有很多异常情况需要处理，这种针对异常情况的测试激励也可以通过 callback 来实现。
3. 学习 SystemVerilog 中的断言机制[41]并将其融合到 UVM 平台当中，通过断言机制不仅可以快速定位错误点，还能配合功能覆盖率提高平台工作效率。

致　谢

时间的脚步不会因为你的挥霍而加速，也不会因为你的不舍而停顿，三年的科大生活随着时间的推移也即将画上句号。回首三年的研究生时光，父母的叮嘱、老师的指点、队领导的教诲、同学的帮助以及朋友的陪伴等画面像电影片段一样浮现在脑海中，最终定格成平淡、充实四个字，这两个词也是我研究生生涯的真实写照。虽然我即将离开学校，离开这里的老师、同学和朋友，但他们给我的帮助却会陪着我继续一路前行，在这里我想对他们表示我最衷心的感谢和祝福。

首先，我要感谢我的导师孙永节老师，是您的鼓励和支持让我下定决心从零开始进入这一领域。在研究方向和学习方法上面，孙老师给了我很多很好的建议和指导，经常在百忙中抽出时间关心我的学习状态，解答我的疑惑；在生活上，孙老师像长辈一样给予我们关心和爱护。与此同时，孙老师身上留露出的儒雅风范也深深的影响着我。

两年的实际工程项目中，我要非常感谢我的项目指导马胜老师。犹记得刚刚进入项目组时，那时候什么都不懂，是马老师花费自己休息的时间对我进行指导，为了让我快速成长，在项目任务中给了我很多支持和督促。马老师对待工作认真负责的态度也深深的影响着我，为了找出设计中存在的错误，马老师放弃了陪伴家人的时间和我加班解决问题，我想这些宝贵的品质都值得我去学习。我还要特别感谢项目组的杨柳、胡月安、田玉恒以及王占力师兄，因为你们的帮助我才能迅速的融入到项目当中，并且不断提升自己。

我还要感谢和我一起学习成长的同学朋友，这一路有你们的陪伴我觉得很幸运。我们从刚入学时什么都不懂的新生到逐渐打开这个领域的大门，都付出了很多但是收获的更多，不仅是知识层面还有我们的同学情谊。感谢我的挚友黄皓天、毛昶、刘明、曾嘉兴、齐娟以及我的室友曾臻和宋意良，我们一起经历了学习成长的快乐，也一起经历了写论文时的痛苦，谢谢你们的鼓励和支持。还要感谢我的女朋友邓梦霞，谢谢你在我论文期间的鼓励和理解，希望你的快乐能一直陪伴着你。

最后，我要特别感谢我的父母以及家人，你们的鼓励和支持是我不断前进的动力，同时也让我感到深深地愧疚和自责。这三年来一直是你们在替我负重前行，担负起了早就应该由我来承担的责任，让我能够心无旁骛的在这里学习生活。养育之恩，无以言谢，你们能够身体健康就是我最大的愿望。

参考文献

[1] Duenas C A M. Verification and test challenges in SOC designs[C]//Integrated Circuits and Systems Design, 2014. SBCCI 2004. 17th Symposium on. IEEE, 2004: 9.

[2] Ghosh P, Ghosh S, Singh P,et al. Case study: Re-visiting SOC verification challenges and best practices[C]//VLSI Design and Test(VDAT), 2015 19th International Symposium on. IEEE, 2015: 1-9.

[3] Lin Y, Lee H, Who M.SODA: A Low-Power Architecture for Software Dadio [C]. //Proceedings of the IEEE/ACM International Symposium on Computer Architecture. Boston, MA, USA, June, 2006: 89-101.

[4] Rashmi V S, Somayaji G, Bhamidipathi S. A methodology to reuse random IP stimuli in an SOC functional verification environment[C]//VLSI Design and Test(VDAT), 2015 19th International Symposium on. IEEE, 2015: 1-5.

[5] William K. Lam. 硬件设计验证[M]. 北京: 电子工业出版社, 2013: 6.

[6] 马宁, 李玲, 田泽, 等. ARINC659 总线协议芯片的仿真验证[J]. 计算机技术与发展, 2010, 20(1): 205-208.

[7] 王浩. 基于 UVM 的 Preamp 芯片的验证与分析[D]. 西安电子科技大学, 2014.

[8] Kim L W, Villasenor J D. Dynamic Function Verification for System on Chip Security Against Hardware-Based Attacks[J]. Reliability, IEEE Transactions on, 2015, 64(4): 1229-1242.

[9] Kleeberger V B, Rutkowski S, Coppens R. Design & verification of automotive SOC firmware[C]//Proceedings of the 52nd Annual Design Automation Conference. ACM, 2015: 117.

[10] 谈笑, 王小力. 一种基于 UVM 的模块级可重用随机化验证平台构建方法[J]. 微电子学与计算机, 2015(3): 67-72.

[11] 胡海生. 基于 UVM 的覆盖率驱动自动验证系统[J]. 电子世界, 2014(16): 101-102.

[12] 张强. UVM 实战[M]. 机械工业出版社, 2014.

[13] 刘瑞, 邵智勇, 康春雷, 等. SoC 仿真验证中多核技术的研究与应用[J]. 现代电子技术, 2015, 38(6): 126-168.

[14] Salah K. A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities[C]//Design & Test Symposium(IDT), 2014 19th International. IEEE, 2014: 94-99.

[15] Bergeron J, Gerny E, Hunter A, et al. Verification methodology manual for systemverilog[M]. Beijing: Beihang University Press, 2007.

[16]王文义，王若雨，董绍静．高性能科学计算的特征分析及其实用方法研究[J]. 计算机科学, 2008, (35)9: 217~219.

[17]黄欣．基于 UVM 的高效验证平台设计及运用[J]．电子技术与软件工程, 2014(4): 28-28.

[18]陈海燕，郭阳，刘祥远．集成电路计算机辅助设计与验证实践[M]．长沙：国防科技大学出版社, 2010: 156-157.

[19]姚爱红，孙盟哲，袁莉娜．基于模拟的 SoC 功能验证研究[J]．微电子学与计算机, 2013, 30[5]: 7.

[20]雷霆．基于 UVM 的网络数据包解析器的验证与研究[D]．西安：电子科技大学, 2015.

[21]Chris Spear. SystemVerilog 验证[M]．北京：科学出版社, 2009: 2.

[22]Accellera. Universal Verification Methodology(UVM) 1.1 User's Guide[M]. 2011.

[23]Synopsys, SystemVerilog VMM Workshop. 2006.

[24]Synopsys, OVM User Guide, Version 2.0.2, June 2009: 13-18.

[25]Synopsys, UVM User Guide, Version 1.1, March 2012: 54-65.

[26]吕毓达，谢雪松，张小玲．基于 UVM 的可重用 SOC 功能验证环境[J]．半导体技术, 2015, 3: 014.

[27]张军，常国锋．基于 UVM 的高效 SOC 验证环境[J]．科技通报，2012，28(12): 70-71

[28]程海燕．基于 UVM 架构的 EHIC 验证环境研究与开发[D]．成都电子科技大学, 2014

[29]曹阳，胡越黎．基于 UVM 的存储控制器功能验证[J]．计算机测量与控制, 2015, 23(003): 834-837.

[30]李前勇，申敏．基于 UVM 验证方法学的 FFTbuffer 模块级验证[J]．广东通信技术, 2015, 35(11): 33-36.

[31]孙铮．基于 UVM 验证方法学的 AES IP 验证[J]．工业技术创新, 2014, 1(1): 26.

[32]Mentor Graphics UVM Documentation Verification Methodology online Cookbook, 2012. 56-70.

[33]王飞．基于 UVM 的 RFID 非接触卡系统级验证[D]．西安电子科技大学, 2014.

[34]王占力．面向 GPDSP 科学计算的高性能 DMA 传输方式的设计与实现[D]．长沙．国防科学技术大学, 2011.

[35]胡月安. 32 位高性能 M-DSP 中支持高效数据传输的 DMA 设计与验证 [D]．长沙．国防科学技术大学, 2012.

[36]张帅．一种支持多种传输模式的 DMA 主机模块设计与实现[D]．长沙：国防科学技术大学, 2014: 15-17.

[37]张帅，孙书为，马胜等. 一种支持高带宽矩阵转置传输的 DMA 设计与实现[C]. 第十七届计算机工程与工艺年会. 2013: 114~119.

[38]丁一博，马胜，孙永节，胡月安. 高可复用 DMA 验证平台的实现[J]. 第十九届计算机工程与工艺年会. 2015: 499-505.

[39]Zhang Heng, Shen Hai-hua. Function verification of godson2 processor[J]. Journal of Computer Research and Development, 2006, 43(6): 974-979.

[40]罗莉，何鸿君，窦强，徐炜遐. 覆盖率驱动的芯片功能验证设计与实现[J]. 计算机工程与科学, 2013, 35(1).

[41]王锐，冯煌. 基于断言合成的验证方法学及应用[J]. 中国集成电路, 2013,10.

作者在学期间取得的学术成果

[1] 丁一博，马胜，孙永节，胡月安. 高可复用 DMA 验证平台的实现[J]. 第十九届计算机工程与工艺年会. 2015: 499-505.