文章编号:1671-4598(2025)03-0323-07 DOI:10.16526/j.cnki.11-4762/tp.2025.03.040 中图分类号:TP391.9 文献标识码:A

基于 UVM 的片上网络验证平台设计

王 鑫,张 畅
(江南大学 物联网工程学院,江苏 无锡 214122)

摘要:片上网络是面向多核片上系统的主要技术组成部分,网络上的通信情景复杂多变,而传统的验证平台存在输入定向,验证层次不规范,验证的覆盖率不足等问题,难以应付此类要求较高的功能验证场景;为解决此问题,构建了一个基于通用验证方法学的验证平台,并设计验证方案,采用约束随机测试为主,辅以定向测试的方式进行验证;此方法既可以保证输入的正确性,又可以提高验证效率;该平台不仅规范了验证流程,还实现了输出结果的自动比对,使验证流程更加自动化;此外,根据设计的功能需求,编写了覆盖组和交叉覆盖组来进行覆盖率的收集,以便量化验证进度;经过仿真后功能覆盖率达到 100%,代码覆盖率达到 87.62%,符合实验预期。

关键词:片上网络;功能验证;通用验证方法学;覆盖率;随机测试

Design of NoC Verification Platform Based on UVM

WANG Xin, ZHANG Chang
(School of Internet of Things Engineering, Jiangnan University, Wuxi 214122, China)

Abstract: Network on chip (NoC) is the major technical component for multi-core on-chip systems, with a complex and changeable communication scenario. However, traditional verification platforms have problems such as input orientation, non-standard verification levels, and insufficient verification coverage, making it difficult to cope with such high requirement of functional verification scenarios. To address this issue, a verification platform based on universal verification methodology (UVM) was constructed, and a verification scheme was designed using constrained random testing as the main approach, supplemented by directional testing. This method can ensure the correctness of the input and improve verification efficiency. This platform not only standardizes the verification process, but also achieves automatic comparison of output results, making the verification process more automated. In addition, according to the functional requirements of the design, the coverage groups and cross coverage groups were written to collect coverage rates for the progress of quantitative verification. After simulation, the functional coverage rate reaches 100%, and the code coverage rate reaches 87.62%, which meets the experimental expectations.

Keywords: NoC; functional verification; UVM; coverage rate; random testing

0 引言

随着半导体行业的持续发展,人工智能(AI, artificial intelligence)芯片因其高效性、灵活性和高并行性等优点在人工智能应用中脱颖而出,具有广泛的应用前景。然而随着对芯片性能的要求不断提高,片上系统(SoC, system on chip)作为 AI 芯片的重要组成部分,其集成了越来越多的功能模块,系统的功耗也迅速增大,但是单处理器的性能回报却逐渐下降[1-2]在这种背景下 SoC 朝着多核系统的方向发展[3-4]。片上网络(NoC, network on chip)作为一种特殊的 SoC,它既具备了 SoC 的优点,又具有良好的复用性和可拓展性,是多核通信的重要组成部分[5]。

本文以人工智能芯片为研究背景,该芯片内部配置了多个核心,采用专用的信息指令和片上网络来实现核心间的数据交互。片上网络作为网络通信技术的一种,具有丰富的信道资源,可以利用网络中不同的物理链路并发执行多个数据通信任务,大大提高了数据处理效率,且显著优于传统总线式系统的性能。但是由于制程工艺的提升,流片的成本也越来越高,这就使得流片前的验证工作更加重要[6]。

传统的芯片验证方法主要依赖于 Verilog 和 Verilog HDL 等硬件描述语言。这些方法在随机化验证方面存在局限,且搭建的验证平台往往维护困难,从而影响了

收稿日期:2023-12-22; **修回日期**:2024-02-02。
基金项目:高等学校学科创新引智计划项目(B23008);未来网络科研基金项目(FNSRFP2021YB11)。
作者简介:王 鑫(1981-),男,博士,讲师。
引用格式:王 鑫,张 畅.基于 UVM 的片上网络验证平台设计[J].计算机测量与控制,2025,33(3):323-329.

验证的效率。为了克服这些挑战，本文引入了通用验证方法学（UVM，universal verification methodology），这是一种基于开源验证方法学（OVM，open verification methodology）和验证方法学手册（VMM，verification methodology manual）的先进验证框架[7]。

UVM的核心优势在于其对约束随机测试（Constrained Random Testing）的支持。约束随机测试是一种强大的方法，能够在预定约束下生成大量随机输入，从而模拟各种可能的操作场景，包括那些设计时未能预见的边缘情况[8]。这种方法极大地增强了验证的有效性。同时，UVM还结合了定向测试（Directed Testing），这是一种更传统的测试方法，侧重于对已知问题和特定功能的精准测试。在UVM框架下，约束随机测试和定向测试相辅相成，确保了测试的全面性和针对性[9]。

UVM借助事务级建模（TLM，transaction level modeling）的标准来实现组件之间的通信交互，并为此提供了一套基于System Verilog语言的库文件[9]。这套库文件包含了众多常用的基类和方法，为验证工作提供了强有力的支持。利用这些基类和方法，可以快速构建出验证环境，显著提高工作效率，且支持构建复杂的随机化测试环境，以自动生成和管理这些随机测试案例[10]。

本文基于UVM构建了验证平台，提出了针对片上网络的验证方案，并编写了不同的测试用例对网络进行功能验证，提升了验证效率，最后通过收集覆盖率来保证验证的充分性。

1 片上网络

本文所研究的对象是一个由64个路由器及其核心（CORE）相互连接而成的片上网络，网络上使用专用的控制指令来实现核心间的交互，其结构采用八行八列的二维网格方式进行连接。以三行三列的二维网格为例，其网络连接方式如图1所示。图1中CR编号为两位8进制数，前一个8进制数表示行号（高三位二进制数），后一个8进制数表示列号（低三位二进制数）。

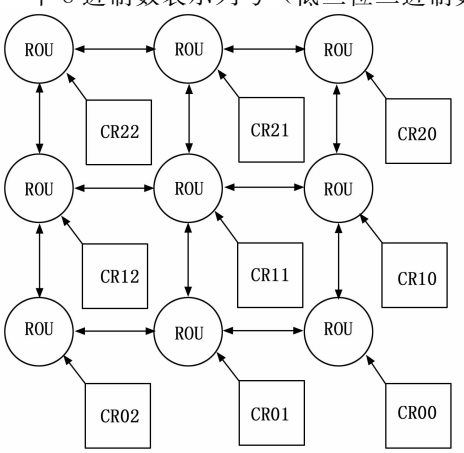

图1 片上网络结构图

在通信开始时，由核心向本地路由器传递控制指令，控制指令中会携带本次通信的操作模式、目标信息。同时控制指令中有3种操作模式：横向传输、纵向传输和节点间传输。节点间传输指定目标核心编号，参与此类传输的核心数量最多为两个；横向传输和纵向传输的目标由8位向量表示，用以标识一组核心中参与传输的核心位置，参与核心的数量可以超过两个。每个操作模式都会选取编号最小核心为控制指令收集者。

本地路由器根据控制指令进行路由计算，并将信息传递给收集者。最后，收集者在接收到所有控制指令后进行判断并完成通信任务。

在通信完成后，所有参与核心输出一个通信完成信号。每个核心内部最多只同时存在一个指令处理流程，且所有核心应当并行控制，某一核心不可被其他核心阻塞。

2 UVM验证方法学

2.1 UVM组件介绍

UVM是作为芯片功能验证环节的一种标准方法，内部设计了许多有助于重用环境的类库[11-13]。由于UVM中所有的组件都是由这些类派生而来，因此通过使用UVM组件类中的组件，开发者可以快速的创建验证环境，提高了验证效率。并将不同的工作内容放在不同的组件中进行，也规范了验证平台的结构和运行过程[14-15]，这样即便是不同的验证人员也方便理解验证环境，便于平台的维护，同时具有可配置性强，可重用性高，封装性好等优点[16-19]。UVM的常见组件在表1中列出。

表1 验证平台的组件及其功能

名称	功能
Test	是验证平台的最顶层，用于配置并构造整个验证平台的测试工作台。
Env	将Scoreboard和Agent、Reference Model封装起来。
Agent	通常将Monitor、Driver、Sequencer等组件封装起来，形成一个容器。
Reference Model	模拟待测设计的功能，并将其输出传输到Scoreboard中进行比对。
Scoreboard	将待测设计的输出数据与Reference Model产生的期望进行自动比对。
Driver	负责将激励驱动到待测设计中。
Monitor	监测接口上的数据，并发送给Reference Model或者Scoreboard。
Sequencer	发送激励给Driver。
Sequence	负责产生激励，并且由Sequencer发送给Driver。

其中，Test是一个关键类，负责设置和控制整个验证环境。每个Test类实例化并定义了一系列的测试用例，这些用例详细规定了在特定环境下执行的测试场景或条件。这是通过设定输入信号的约束或直接操控输

入信号来实现的，以此产生多样化的影响。因此，更换不同的测试用例能有效地改变输入信号的特性，从而提供对待测设计的全面验证。

而参考模型（Reference Model）在 UVM 中扮演着一个模仿设计逻辑的行为模型的角色，其主要目标是复现与待测设计相同的功能。参考模型的存在类似于一面镜子，它反映出待测设计的功能正确性。通过与参考模型的对比，验证人员可以评估被测设计的行为是否符合预期。

2.2　平台工作原理

由基类直接创建出的组件是各自独立的，为了使各个组件之间的通信更加有序，就需要在组件之间建立专门的通道，让数据信息只能在这个通道中传递。UVM 采用了事务级建模（TLM）的机制来完成各个组件之间的通信[20-21]，而事务就是 TLM 中信息传输的基本单位，其对象包括变量、约束以及操作数据的方法。在进行通信时，首先在各个组件之间建立通信端口与先入先出（FIFO，first input first output），并将这些端口与 FIFO 相连接，如此便完成了通道的搭建，也确定了数据传输的方向。在数据发送端，通过 write 操作将数据写入 FIFO 中，再通过 get 操作在接收端将数据取出进行操作。

本次实验所搭建的验证平台的主要框架如图 2 所示，图 2 中展示了大部分的组件与其中的连接关系。验证开始时，Sequence 会根据定义好的约束来产生输入激励，后由 Sequencer 发送给 Driver，最后 Driver 将产生的激励发送到平台与待测设计（Design Under Test，DUT）的接口上。这时 Monitor 会监测到接口上的输入信号，并且通过通信端口将这些信号传输给参考模型，而 DUT 则可以直接从接口上获得输入数据。当参考模型与待测设计接收到输入数据后经过运算产生输出数据，参考模型会通过通信端口将数据传输给 Scoreboard，而待测设计的输出则是通过 Monitor 的监测后传输给 Scoreboard，最后在 Scoreboard 中完成对双方输出数据的比对。

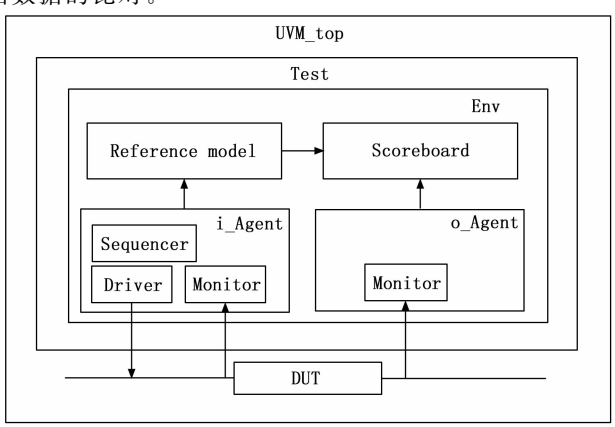

图 2　UVM 验证平台结构图

2.3　验证的指标

在完成平台的搭建之后，对验证过程的进度和质量进行细致的评估至关重要。在大量的测试中，模型与 DUT 输出的比对，可以在一定程度上说明设计功能的正确性。但是，已有测试是否成功遍历到了所有使用情景，在所有使用场景中比对是否都能通过，DUT 的代码是否被全部执行过。只有解决了这些问题，才可以明确知道验证是否全面、结果是否收敛，为此引入了以下两个概念[22-23]。

代码覆盖率：此指标评估了验证过程中 DUT 代码的执行范围。代码覆盖率的测定通常由验证工具自动完成，无需验证人员编写额外代码。高代码覆盖率意味着测试用例已经执行了 DUT 的大部分或全部代码段，从而揭示了未经测试的代码区域，这些区域可能隐藏着潜在的错误或非预期行为。值得注意的是，在大型的设计中，代码覆盖率通常很难达到 100%，而未能覆盖的区域需要验证人员进行检查并确认是否符合规范[24]。

功能覆盖率：与代码覆盖率不同，功能覆盖率着眼于验证 DUT 的所有预期功能是否得到了充分测试。由于功能覆盖率不会自动由工具生成，因此验证团队需依据 DUT 的功能需求制定相应的功能覆盖策略，并手动编写测试用例以确保所有关键功能均得到验证[25-27]。如果覆盖率达到预期目标，则验证基本完成。如果未达到预期覆盖率，则可能需要调整测试用例或增加新的约束条件，以确保更全面的验证。

除此以外，为了有效地执行对待测设计的验证，需进一步细化各组件的具体工作内容。接下来的关键步骤包括明确验证的关键要素，并制定一套周密的验证方案。这个方案将为各组件提供详细的操作指南，从而优化和完善平台组件的功能和性能。

3　网络验证方案的设计

3.1　验证要点

根据对设计功能需求的分析，对片上网络的验证以其功能的正确性为目标，主要难点如下。其一，要保证约束后产生的随机数据符合控制指令的规范，否则无法完成通信。其二，控制 64 个核心并行发送激励，参与通信的核心之间既要相互关联，又要保持相对独立。每个核心内部同时只能存在一个指令处理流程，完成后才能开始下次通信。其三，模拟设计的通信功能，对 64 个核心收到的指令信息进行判断并完成通信。其四，对验证进度进行判断，保证验证的正确性。

在对网络功能与验证难点进行分析后，提炼了相应内容的验证要点：

在产生激励时，为了保证验证的充分性要产生随机性更高的输入数据，但是为了能够完成通信，这些随机

数据的输入信息要有一定的关联性。

在激励发送时，并行启动 64 个进程负责各个核心的激励发送工作，并对每个核心的处理进程进行计数，当数字大于 1 时报错。核心只有完成了此次通信后，才可以发出下一次控制指令。

在处理过程中，对每个核心接收到的指令信息进行判断。根据操作模式以及核心关系的不同，判断何时应该完成通信，并输出正确的输出数据。

在数据比对时，对每个核心采用计数比对的策略来判断 DUT 与参考模型完成通信的数量是否一致。

制定功能覆盖策略，编写功能覆盖组并收集覆盖率，根据覆盖率报告的反馈来量化验证进度，判断验证的充分性。

针对以上内容，本文以网络的功能需要和实际使用为基础展开验证。

3.2 测试用例

在本次研究中，为了使输入符合要求的同时能包含尽可能多的情景来对设计进行测试，确保在各种输入情况下功能正常，选择使用随机测试为主，并补充定向测试的方式来进行全面的功能验证。

1）创建随机测试的测试用例。为了随机的效果更贴合实际，在测试用例中将节点间传输，横向传输，纵向传输的逻辑分成 3 个约束块分别进行约束。首先产生一个随机整数 n，范围指定为 0 到 2，当随机到不同数字时，调用不同的约束块产生不同种类的随机数据，此时便完成了一次激励的产生。通过这种方式，便可以控制产生 n 的数量来对随机激励的数量进行有效控制。

对于节点间传输，指定 p_1、p_2 代表着参与通信双方的核心编号，且两个参数均是由随机函数产生的范围为 0 到 63 的随机数，由于目标信息为参与通信的另一方的核心编号，便直接将 p_1 核心的目标信息赋值为 p_2 的编号，将 p_2 核心的目标信息赋值为 p_1 核心的编号。

对于横向传输和纵向传输则是相同的思路，以横向传输为例，由于目标信息中存储的是八位向量，所以行号和八位的向量均由随机函数产生。行号代表这一行的位置，而八位的向量则代表参与横向传输的核心在这一行中的位置。最后以行号为索引，将信息赋值给对应的核心。

按照以上的思路发出大量的随机激励，就会产生各种随机的通信组合，这样不仅可以覆盖到预期的通信功能，也可以找到从未预料到的漏洞。

2）补充定向激励，进行死锁专项测试。当核心接收到错误或者未能成对出现的控制指令时，通信无法完成，产生死锁。该定向测试分为 3 个部分进行，首先是节点间传输时，通信双方未能成对出现。其次是横向传输和纵向传输时，本核心或者其他成员未发出控制指令。

这是对通信功能的反向验证，若是这个网络无论控制指令正确与否都能完成通信，那么这个网络就无任何意义。

3.3 激励的驱动

在产生激励后，由于 64 个核心分别独立，所以要对各个核心进行并行控制。在 Driver 中需要实现对各个核心的单独驱动，每个核心的输入只需考虑处理进程的占用而不应该被其他核心阻塞。为实现对各个核心的单独驱动本文将其分为以下几个部分：

首先，为了确保在核心的内部最多只存在一条控制指令，在 Driver 中声明计数器用于计数，初值为 1，代表每个核心可以接收的控制指令数，当核心发出一条控制指令后，相应核心的计数器数值减一，当接收到通信完成信号后，相应核心的计数器数值加 1。当计数器为零时则代表该核心内已经存在一条控制指令，需要等待上一次的通信完成信号到达后才可以参与下一次通信。

其次，为了并行控制 64 个核心，需要实现一个并行控制逻辑，它能够同时向所有核心发送控制信号，以便同时启动或停止每个核心的工作。在核心准备发出控制指令时会先检查计数器的数值，如果处理进程被占用则等待计数器的数值重新置 1 后将指令发出。

过程中需要对计数器数值进行监测，当数值大于 1 时报错并停止仿真。这是为了及时发现错误并定位到错误来源，可以有效提升验证效率。

3.4 参考模型的编写

在通信过程中，参考模型会接收到来自所有核心的控制指令。为了完成通信，首先判断每个核心控制指令中的操作模式，再根据操作模式的不同进行分类处理。参考模型的处理流程如图 3 所示。

对于节点间传输，由于目标信息中存储的是通信的另一方的核心编号，首先声明容量为 64 的目标数组用以存放目标核心编号，将所有目标数组的初值设置为，表示对应的核心处于空闲状态，内部无正在处理的通信流程。当收到信息后，第一时间检查此核心的数组值是否为，若不是，则表明该核心同时发出两条控制指令，应当报错。否则，就将目标核心编号存入对应数组中。随后，对所有数组进行遍历检查，若双方目标数组内存储的编号与对方的编号一致，则表示通信成功，将数组复位为，并且输出通信完成信号。若未完成通信，则继续遍历剩余核心，并等待通信完成。

对于横向传输与纵向传输则思路相同，以横向传输为例，由于在横向传输时，通信参与者数量可能大于 2，因此要收集齐所有参与通信的核心的控制指令后才算通信完成。为解决此问题，将通信过程分成以下两步进行。

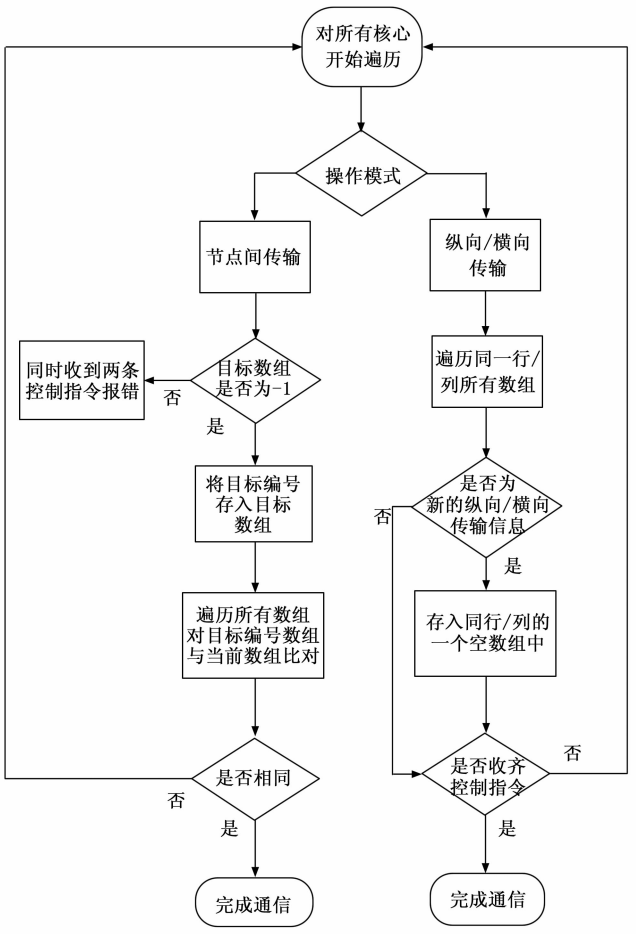

图 3　参考模型通信流程图

1）判断接收到的控制指令是否是同一次横向传输通信。预定义数组，用于存放本行的横向传输信息，在收到第一次横向传输信息时便存入数组中。再收到本行的横向传输信息时，便与数组中的信息做比较。若相同，则表示收到了这次横向传输中一个参与者的控制指令。若不同，则表示不属于同一次横向传输，并将信息存入同行的其他数组中。由于每行有 8 个核心，极端情况下会出现 8 个不同的横向传输信息（此时均为核心自己与自己横向传输），所以每行声明 8 个数组来解决这个问题。

2）判断是否接收到了所有本次横向传输参与者的控制指令。由于横向传输时，目标信息表示为 8 位二制进数组成的向量，因此采用置零法来判断何时收齐控制指令并完成通信。首先，将目标信息存入一个数组中，当收到了一个属于本次通信的控制指令时，便根据其核心编号，将数组中属于该核心位置上的二进制数置零。当数组中存储的八位向量全部为零时，表示收齐了所有控制指令，通信完成，输出通信完成信号。而当出现错误无法通信完成时，根据数组中未能置零的数据便可以快速找到问题所在。所有的通信完成信号皆作为输出信号写入与 Scoreboard 相连接的 FIFO 中。

最后在仿真结束后检查所有数组，若有不为空的则报错，保证了没有控制指令卡住造成死锁，或者一直没有通信完成。

3.5　自动比对的实现

由于网络的使用场景十分复杂，所以需要大量的数据进行随机测试，面对较多的核心与大量的输出数据如何保证数据比对的效率是需要解决的首要问题。

在网络中，对于每一个核心来说，在完成通信时作为输出信号的通信完成信号是一个位宽为 1 的信号。因此只进行大小比对意义不大，还需要检查每个核心发出的通信完成信号的数量是否相同。但是如果只是简单的做加法运算，那么只有在仿真结束后，进行总数比对时才会发现比对错误。而进行大批量的数据输入时，每次仿真的时间都会大大增加，这样就会造成验证效率的低下且无法判断错误的根源所在。为了解决这个问题，本次的自动比对要在做加法的同时去做减法，具体实现方式如下。

首先为参考模型和 DUT 分别定义了 64 个计数器，初值为 0。随后对参考模型和 DUT 的计数部分进行并行控制。当接收到一个参考模型的输出时，检查所有的核心，若核心完成通信，则将模型计数器中与核心编号相对应的计数器的值加一。当接收到一个来自 DUT 的输出时，同样检查所有的核心，若核心完成通信则将对应 DUT 的计数器数值加一。但是由于参考模型在处理数据时是没有延迟的，而 DUT 中会有各种不同的延迟操作，所以参考模型处理数据、输出数据的速度一定快于 DUT。因此在 Scoreboard 中，DUT 的数据接收速度一定慢于参考模型。当 DUT 的同一次通信完成信号到达时，参考模型的一定已经到达，此时开始做减法。以 DUT 的核心输出结果为参考，每个完成通信的核心其对应的 DUT 数组与模型数组数值减一。同时做出判断，在数值减一后，对应的双方计数器是否有值变成，如果是则表示通信失败，DUT 的输出与模型的输出并不一致，报错并停止仿真。若没有，则本次通信成功，开始下一次的加减比对。

相较于单纯的加法比对，由于 DUT 与模型的输出速度不同，只有在完成所有计数后才能通过数字的大小不同来确定是否出现错误。加减比对的优势就十分明显，因为 DUT 的速度更慢，所以选择 DUT 为标准做减法，当某个数组数值小于 0 时，就表示这个核心在这次通信时比对双方的答案不同，可以及时发现错误，并定位到错误所在，很大程度上方便了验证人员的工作。

最后当所有数据比对完毕且并未出现比对错误时，检查所有 FIFO 中是否有数据残留并未取出。

3.6　功能覆盖策略

依据本次验证目标的功能需求，功能覆盖组需要对

64 个核心的控制指令进行覆盖。而对于每个核心来说，有 3 种不同的操作模式，且不同的操作模式下目标信息的表示方式也有所不同。由于控制指令的强关联性，除了对操作模式与目标信息分别进行覆盖之外，还需要采取交叉覆盖的策略。

本次实验的功能覆盖策略如下：

对于每个核心，对其能否发出控制指令，能否发出所有的操作模式，能否对所有的目标信息发出控制指令进行覆盖。

对于每个核心，在每种操作模式下，能否对所有目标信息发出控制指令进行交叉覆盖。

4 实验结果与分析

在本次仿真实验中，首先采用大规模的随机测试来对网络通信功能的正确性进行测试，并用覆盖率驱动策略来量化验证进度。通过检查覆盖率报告，以识别未被充分覆盖的区域，提高测试的效率和全面性。

使用随机测试的测试用例，生成 100 000 个随机数据样本，并将这些数据并行地驱动到每个核心中。仿真结束后，首先进行了输出波形的详细检查。观察结果显示，输出波形中未发现任何异常情况，表明网络在各种输入条件下的通信表现符合预期。进一步调阅仿真日志，发现大量的自动比对均成功执行，无任何失败案例，这进一步验证了设计的可靠性。

在仿真平台中，所有的 FIFO 队列均被成功取空。这一观察表明，数据处理流程有效且无堆积现象发生。网络中 64 个核心的代码覆盖率综合后达到 87.62%，展示了代码在仿真过程中得到了高度的执行和检验，而没有达到 100%是因为本次验证的片上网络中的路由器与核心是通用型，为一份代码例化 64 份，而位于网络不同位置的路由器在接收到控制指令时的处理方式不同，这就导致设计的代码在核心和路由器的编号变化时会有不同程度的冗余，这部分代码不会被覆盖但并不影响其功能，经过人为排除后符合设计规范。功能覆盖率的覆盖范围以及收集结果如表 2、表 3 所示。表 2 中，采用二进制数来表示不同覆盖点的可能取值，并利用“cross”关键字标识了不同覆盖点间的交叉覆盖。表 2 的设计是为了详细说明如何对各个覆盖点进行评估和交叉分析。而表 3 则进一步阐述了覆盖组的构成，这些覆盖组实际上是表 2 中定义的覆盖点的集合。在表 3 中，网络中的 64 个核心的覆盖点被分类并分配到不同的覆盖组中。例如，“valid _ cov”覆盖组专门定义了所有核心的“valid”覆盖点，并对这些覆盖点进行了集中收集。在收集到的覆盖率报告中，功能覆盖的实际值达到了 100%，符合预期。这一结果强调了验证环境在覆盖各种功能方面的全面性和有效性。

表 2 每个核心功能点覆盖范围

覆盖点	含义	取值范围
valid	输入有效信号	1’b0:1’b1
mode	操作模式	2’b00,2’b01,2’b11
mask	目标信息	[8’b0000_0000:8’b1111_1111]
mode_cross	输入有效信号与操作模式的交叉覆盖	valid:1’b1,mode:[2’b00,2’b01,2’b10];cross valid,mode;
p2p_mask	节点间传输时与目标信息的交叉覆盖	valid:1’b1,mode:2’b10,mask:[8’d0:8’d63];cross valid,mode,mask;
row_mask	横向传输时与目标信息的交叉覆盖	valid:1’b1,mode:2’b00,mask:[8’d0:8’d255];cross valid,mode,mask;
col_mask	纵向传输时与目标信息的交叉覆盖	valid:1’b1,mode:2’b01,mask:[8’d0:8’d255];cross valid,mode,mask;

表 3 功能覆盖结果

覆盖组	功能描述	期望/%	覆盖率/%
valid_cov	64 个核心的输入有效信号	100	100
mode_cov	64 个核心的操作模式	100	100
mask_cov	各种操作模式下的目标信息	100	100
mode_cross	64 个核心输入有效信号与操作模式交叉覆盖	100	100
p2p_mask_cross	节点间传输时与目标信息交叉覆盖	100	100
row_mask_cross	横向传输时与目标信息交叉覆盖	100	100
col_mask_cross	纵向传输时与目标信息交叉覆盖	100	100

随后使用定向测试的测试用例，进行死锁专项测试，令参与通信的某个核心不发出控制信息。而定向测试结束以后，观察到波形并没有输出数据，FIFO 中存在数据残留。该实验结果表明，当无法收齐所有参与核心发出的控制指令时，通信无法完成，造成数据堆积，并遗留在 FIFO 中无法取出。从反向验证了网络的功能，即控制指令必须配合使用，否则无法完成通信，结果符合预期。

总的来说，本次仿真实验通过细致的验证计划和全面的结果分析，验证了网络的功能，确保了设计的正确性。

5 结束语

本文提出一种基于 UVM 验证方法学的验证平台，对片上网络在复杂通信环境中的可靠性进行验证。通过精心设计的仿真实验，模拟了网络在各种操作条件下的行为。实验结果显示，网络能够有效地处理复杂的通信

场景，且在不同测试条件下均表现出预期的行为，证明了其设计的可靠性和稳定性。

在验证过程中，特别注重了代码覆盖率和功能覆盖率的分析。对网络上所有核心的覆盖率进行综合后，实现了高达 87.62%的代码覆盖率，和 100%的功能覆盖率。确保了代码在仿真过程中的广泛执行和测试，功能覆盖率的细致记录和交叉覆盖策略的应用也进一步加强了在处理复杂的控制指令时测试的全面性。

本研究的成功不仅展示了 UVM 框架在处理高复杂度系统验证中的有效性，而且为网络的应用和优化提供了坚实的基础。同时为类似的复杂系统验证工作提供了有价值的参考。

参考文献：

[1] 余　霞. 面向无线通信的 NoC 架构与映射技术研究 [D]. 成都：电子科技大学，2021.

[2] LEE K. Trends of modern processors for AI acceleration [C] //2021 18th International SoC Design Conference. IEEE，2021：227－227.

[3] 马　彬，刘　威. SoC 芯片上 AXI 总线 IP 验证 [J]. 电子设计工程，2023，31 (13)：56－60.

[4] 包日辉. SoC 设计平台中若干 IP 模块的设计 [D]. 杭州：杭州电子科技大学，2019.

[5] 张大坤，宋国治，王莲莲，等. 三维片上网络拓扑结构研究综述 [J]. 计算机科学与探索，2015，9 (2)：129－164.

[6] 马　鹏，刘　佩，张　伟. 基于 UVM 的 AMBA 总线接口通用验证平台 [J]. 计算机系统应用，2021，30 (7)：57－69.

[7] 刘光宇，马　盼，刘肖婷，等. 基于 UVM 的应答器传输模块验证方法研究 [J]. 铁路通信信号工程技术，2023，20 (9)：7－10.

[8] 李姝萱，卜刚. 基于 FPGA 的 PIE 编码与 UVM 验证平台的设计 [J]. 电子技术应用，2021，47 (6)：110－114.

[9] 张　强. UVM 实战－卷Ⅰ [M]. 北京：机械工业出版社，2014.

[10] MALHOTRA S，PRAKASH N R. UVM-Based verification IP of AMBA AXI protocol showing multiple transactions and data interconnect [M] //Advances in Communication，Devices and Networking. Springer，Singapore，2019：529－536.

[11] 刘　斌. 芯片验证漫游指南 [M]. 北京：电子工业出版社，2018.

[12] WEI N，WANG X. Functional coverage-driven UVM-based UART IP verification [C] //2015 IEEE 11th International Conference on ASIC (ASICON)，IEEE，2015：1－4.

[13] 邓庆勇，朱　鹏，习建博. 基于 UVM 的 DBF 系统模块级可重用验证平台的实现 [J]. 微电子学与计算机，2018，35 (1)：115－117.

[14] 陈琳娜. 基于 UVM 的层次化验证平台研究 [D]. 杭州：浙江大学，2018.

[15] LIU Y，HUANG S，WANG H，et al. System verification scheme based on an MCU [C] //Journal of Physics：Conference Series. IOP Publishing，2022，2166 (1)：012004.

[16] 段一杰. 基于 UVM 的 APB-UART 验证平台的设计与实现 [D]. 成都：电子科技大学，2020.

[17] NAIK A O，KURUVILLA E，CHAVAN A P. Integration and verification of IP cores on SoC [C] //2021 IEEE Mysore Sub Section International Conference (MysuruCon)，IEEE，2021：120－125.

[18] 雷　霆. 基于 UVM 的网络数据包解析器的验证研究 [D]. 成都：电子科技大学，2016.

[19] FOSTER H D. Trends in functional verification：a 2016 industry study [C] // DvCon，San Jose，2017：3－4.

[20] 吴星星. 基于 UVM 的 SPI 接口 IP 核的验证平台设计 [D]. 合肥：安徽大学，2016.

[21] HUSSIEN A，MOHAMED S，SOLIMAN M，et al. Development of a generic and a reconfigurable UVM-based verification environment for SoC buses [C] //2019 31st International Conference on Microelectronics (ICM)，IEEE，2019：195－198.

[22] 刘霜梅，高新宇，谭星亮. 基于 UVM 实现 TCP/IP 协议栈的自动化交互验证平台 [J]. 中国集成电路，2016，25 (3)：19－23.

[23] HUNTER B，CHEN B，LIPON R. Reset testing made simple with UVM phases [J]. SNUG Silicon Valley，2013：1－4.

[24] 刘魁玉. 基于 UVM 的 1553B 总线协议验证平台设计 [D]. 哈尔滨：哈尔滨工业大学，2019.

[25] 张　瑞. 基于断言的形式化验证与 UVM 的综合应用 [D]. 西安：西安电子科技大学，2018.

[26] KASHYAP B，RAVI V. Universal verification methodology based verification of UART protocol [J]. Journal of Physics Conference Series，2020，1716：012040.

[27] 毛国栋. 基于 UVM 的 AHB-APB 桥模块验证 [D]. 太原：中北大学，2023.

(上接第 294 页)

[17] 周戌申. SpaceWire 接口 IP 设计 [D]. 上海：中国科学院研究生院，2008.

[18] XILINX. LogiCORE IP 光纤 8B/10B v10.2 product guide [S]. U.S.A，2014：37－104.

[19] 朱　琛，崔　镭，邵春伟，等. 基于 FPGA 的高速光纤传输卡 [J]. 电子与封装，2018，18 (3)：22－25.

[20] ECSS-E-ST-50-11C. Space enginering spaceWire protocols [S]. European Cooperation for Space Standardization (ECSS)，2008：22－109.