文章编号:1671-4598(2024)10-0201-07 DOI:10.16526/j.cnki.11-4762/tp.2024.10.029 中图分类号:TP391.9 文献标识码:A

基于 UVM 的片上网络路由器验证平台

王 鑫,翟周伟
(江南大学 物联网工程学院,江苏 无锡 214222)

摘要: 路由器是片上网络的关键组件,其性能对于整个网络的性能具有重要影响;针对片上网络路由器进行功能验证,采用 SystemVerilog 和自动化脚本搭建了基于通用验证方法学(UVM)的验证平台,简化了验证流程;在验证平台中,通过划分多个 agent 向路由器的每个端口发送受约束的随机激励和定向测试序列,并创建了多个独立的测试用例,对路由器的功能进行充分的验证;通过运用覆盖率驱动策略,对验证进程进行了量化;根据路由器的设计要求,编写了覆盖组和交叉覆盖组以收集覆盖率数据;此验证平台已应用于人工智能芯片的验证工作,平台中的组件和测试用例均可实现更高层次的复用;此外,通过 VCS 和 Verdi 的联合仿真,实现了 100%的功能覆盖率和 95.6%的代码覆盖率的目标。

关键词: 片上网络路由器;验证平台;覆盖率;SystemVerilog;人工智能芯片

Verification Platform for On-chip-network Router Based on UVM

WANG Xin, ZHAI Zhouwei
(School of Internet of Things Engineering, Jiangnan University, Wuxi 214222, China)

Abstract: A router is a key component of on-chip network, and its performance has an important impact on the performance of whole network. For the functional verification of the on-chip-network router, a verification platform based on the universal verification methodology (UVM) is constructed by using the SystemVerilog and automation scripts, simplifying the verification process. In the verification platform, the router's functionality is adequately verified, which sends the constrained random excitation and directional testing sequence to each port of the router by dividing multiple agents, and multiple independent test cases are built to fully verify the functionality of the router. Verification process is quantified by applying the coverage-driven strategies. According to the design requirements of the router, coverage groups and cross-coverage groups are written to collect the coverage data. This verification platform is already applied in the verification of AI chips, and the components and test cases in the platform can be achieved the reuse at a higher level. In addition, the system achieves the goal of 100% functional coverage and 95.6% code coverage through the joint simulation of VCS and Verdi.

Keywords: on-chip-network router; verification platform; coverage; SystemVerilog; artificial intelligence chip

0 引言

随着半导体制造工艺和互联网技术的飞速发展,高可靠性、高性能的人工智能(AI, artificial intelligence)芯片[1]已经成为各国竞争和博弈的重要领域。随着 AI 芯片上集成的知识产权模块(IP, intellectual property)数量的增加,IP 之间通信和互连的方式已成为影响数字系统性能的关键因素。在现代数字系统中,功率损耗常常发生在导线驱动的情况下,而导线延迟问题在芯片运行过程中导致了大量的时间延迟[2]。

片上网络(NoC, network on chip)是 AI 芯片中连接各个 IP 的重要通信结构,它通过提供并行通信通道的方式降低了 IP 之间的通信延迟和能耗,实现了更高效的 IP 间通信。同时,NoC 为现代片上系统(SoC, system on chip)应用提供可扩展、灵活且节省带宽的解决方案,并逐渐替代了传统的总线架构,例如:AMBA、AXI、CoreConnect、STBus 等等,这使得 NoC 成为解决 SoC 内部构建互连的新方案[3]。路由器作为 NoC 的重要组件,是 IP 间通信的关键节点,负责接收来自输入端口的数据包并根据路由表进行转发,其内部的转发机制直接影响通信时延。优化路由器设计和配置可降低通信时延并提高 AI 芯片的整体性能和效率。因此,解决 AI 芯片中 IP 之间的通信时延问题与路由器的设计和配置密切相关。

目前,国内芯片工艺的研发成本和流片成本极其高昂,这使得路由器的设计不容许出现差错。在芯片设计人员设计好路由器之后,验证工作也是不可忽略的。验证的目的是找出待测设计(DUT, design under test)中的错误,这个过程通常是把 DUT 放入一个验证平台中实现的。验证工程师通过搭建验证平台,编写测试用例对路由器进行详细的测试,确认功能正确无误之后,才能进行后续的流片工作。在芯片研发阶段,验证占据的时间已经达到 70%~80%[4],因此,采用快速且高效的验证方法显得尤为重要。

收稿日期:2023-09-20; **修回日期:**2023-10-23。
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,翟周伟.基于 UVM 的片上网络路由器验证平台[J].计算机测量与控制,2024,32(10):201-207.

UVM（Universal Verification Methodology）验证方法学为验证工程师建立了一套标准组件库[5]。它使用事务级建模（TLM，transaction level modeling）标准实现模块之间的通信，并使用 SystemVerilog 语言来创建组件。其特点包括可重用性高、支持随机化约束、随机生成测试用例等，这些优点大幅缩短了验证时间。因此，该语言被广泛应用于数字集成电路（IC，integrated circuit）的验证领域[6]。SystemVerilog 支持受约束的随机激励和覆盖率分析，并支持面向对象的编程结构[7]。本文使用 UVM 来搭建验证环境，通过使用 Python 脚本来生成多个测试用例，对路由器的功能进行详细的测试，提高了验证效率和验证平台的重用性。

1 片上网络路由器的结构以及原理

在验证工作开始之前，首先需要充分理解路由器的结构和工作原理，了解了结构和原理之后才能对路由器进行详细的验证。将数据从一个 IP 传输到另一个 IP 的行为就是“路由”，在 NoC 中执行此操作的部件被称为路由器。本文的待验证对象如图 1 所示，路由器有 5 个输入端口和 5 个输出端口，分别对应东方向、南方向、西方向、北方向和本地方向。每个节点的端口方向信号用 3 比特编码来表示，本地方向的编码是 3'b100，北方向的编码是 3'b011，东方向的编码是 3'b010，西方向的编码是 3'b001，南方向的编码是 3'b000，这 5 个传输方向分别支持 2 个虚通道，以微片（flit）为基本传输单位，flit 的大小决定了网络的带宽[8]。在传输过程中，每个数据包的 flit 之间不可被其它数据包的 flit 插入。每个路由器之间使用信用交换协议进行信息的交互。

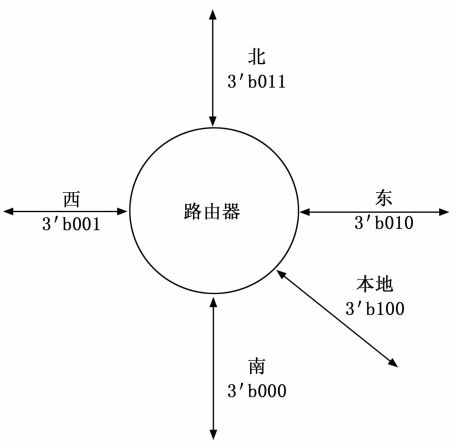

图 1 路由器的结构

路由器由缓冲写入模块、路由计算模块、开关分配模块和开关传输模块组成，每个输入方向拥有两个虚拟通道。缓冲写入模块负责接收数据包并将其存储在路由器的缓冲区中，该缓冲区独立于 5 个输入端口。每个数据包中的路由信息指示了其应沿着网络路径到达的目标位置，路由计算模块根据该路由信息确定传输方向。当路由计算模块确定数据包的传输路径后，它会向开关分配模块请求分配传输资源。开关分配模块根据目标虚拟通道的信用情况对请求进行仲裁和分配。若某个虚拟通道的信用良好，开关传输模块会根据开关分配模块的指示控制数据包的传输。当多个数据包同时申请通过同一虚拟通道传输时，路由器采用信用仲裁机制以确定优先级较高的数据包可获得传输权限，而优先级较低的数据包则需要等待。这种机制确保了数据包的公平传输。一旦某个数据包被批准通过某个虚拟通道进行传输，开关传输模块将负责将该数据包从正确的输入端口传送至正确的输出端口。路由器的具体功能如下：

1）路由器在收到单播包和广播包时，它会根据包中的目标信息进行路由计算。对于单播包，路由器将计算出单个目标位置并发送数据包到对应的方向。对于广播包，路由器将计算出所有目标位置并将数据包发送到对应的方向。

2）路由器在收到广播包之后，多个目标端口会在同一时刻输出广播包。

3）当数据包在路由器内部滞留时间超过设定阈值时，这些数据包的输出优先级高于未超时的包。这意味着当超时的数据包与其他数据包竞争传输资源时，它们将获得更高的优先级并成功传输。超时的数据包应与其他数据包进行仲裁，然后从相应的输出端口输出。

2 UVM 验证结构

2.1 UVM 的基本组件

UVM 是一个库，它提供了一套用于芯片验证的标准和规范。在这个库中，几乎所有的东西都是使用类来实现的，类是用户自定义的，用于封装数据和对数据的操作[9]。uvm_object 是 UVM 中最基本的类，简称基类，它定义了一些基本的属性和方法，用于创建其他类。从基类做扩展并创建新的派生类（子类）的过程，是类的派生。当一个类被扩展产生一个派生类的时候，派生类（子类）就会继承基类（父类）的所有属性和方法。UVM 验证平台中的所有组件都是由 UVM 预先定义好的类派生而来[10]。这些组件通过特定的方式组合在一起，形成了一个完整的验证环境。UVM 验证方法学将芯片验证流程标准化，基于 UVM 方法学的不同验证平台结构可能有所不同，但它们都遵循类似的架构模式。芯片验证人员可以按照 UVM 方法学提供的大致框架，编写继承 UVM 中各个组件，建立各个组件间的通信，完成 UVM 验证平台的搭建。下面将介绍 UVM 中各个验证平台组件的具体功能：

1）transaction 组件：产生受约束的随机激励，这些激励以事务的形式被发送到验证环境中。

2）sequence 组件：将激励装载到 sequencer 上，以便 sequencer 可以管理和控制事务的顺序。

3）sequencer 组件：对 sequence 进行管理，将 sequence 生成的事务级数据发送给 driver 组件。

4）driver 组件：向 sequencer 索要事务级数据，将索要的事务级数据转换成 DUT 能够接收的信息，通过 interface 传给 DUT[11]。

5）monitor 组件：监控数据，通过 interface 监控待测试模块，将监测的数据转换成事务级数据，分别传给 score-

board 组件和 reference model 组件[12]。

6）agent 组件：相当于一个抽象容器，将 monitor 组件、sequence 组件和 driver 组件封装在一起[13]。

7）scoreboard 组件：进行功能检查和错误检测，通过比较 reference model 组件和 DUT 的输出数据来判断 DUT 的功能是否正确[14]。

8）reference model 组件：模仿 DUT 的功能，产生期望的输出值。

9）env 组件：完成所包含组件的创建和接口的对接，所容纳的组件包括 agent 组件、scoreboard 组件和 reference model 组件。

10）interface 组件：传输输入和输出信号，例如数据输入和输出、时钟信号等等。

11）test 组件：封装 env。

2.2　UVM 的运行机制

UVM 将整个验证平台运行流程分成如图 2 所示的各个阶段（phase），不同 phase 的执行顺序是自上而下的[15]。验证平台的各个组件不仅具有可重用性等优点，而且还具有 phase 自动执行特性。

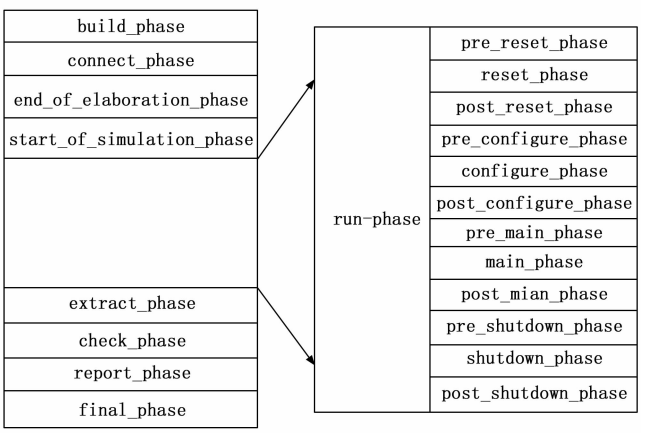

图 2　UVM 中的 phase

在 UVM 验证平台中，所有组件的实例化操作通常都在 build _ phase 中进行，而各个组件之间通过 FIFO（先入先出，first input first output）对接的部分放在 connect _ phase 中完成，reset _ phase、configure _ phase、main _ phase 和 shutdown _ phase 是核心的 phase。这 4 个 phase 通常用于模拟 DUT 的正常工作方式，在 reset _ phase 中，通常会对 DUT 进行复位和初始化等操作，在 configure _ phase 中，DUT 的配置工作将得以进行，包括设定初始条件，配置特定的寄存器等。main _ phase 是核心的运行阶段，大部分的测试在此阶段执行，在此阶段，向 DUT 发送测试向量，并收集 DUT 的响应，以验证其功能是否正确。在 shutdown _ phase 中会进行一些与 DUT 断电相关的操作，包括关闭与 DUT 相连的电源，释放占用的资源等，通过细分实现对 DUT 更加精确地控制。另外，check _ phase 则用于检测验证平台中各个组件的行为的准确性。在运行验证平台的过程中，当执行 final _ phase 时，需要检查 driver 组件中用于模拟信用机制的自定义数组是否恢复到初始值，以及检查 scoreboard 组件中所有用于存放输出数据的队列是否被清空，这些步骤对于提高验证平台的正确性和完整性具有重要意义。

这种 phase 机制的使用让整个验证平台的构建和执行变得非常有序和可控，将复杂的验证流程分成了一系列有序的阶段，确保了每个组件都在正确的时间和条件下执行，在很大程度上解决了因代码顺序不当可能会引发的问题，提高了验证的准确性和效率，极大地方面了用户。

此外，objection 机制的引入，进一步增强了 UVM 的灵活性和可扩展性。UVM 通过 objection 机制来控制验证过程的开始和结束。每个 phase 在执行过程中，都会检查是否有 objection 被举起。如果存在举起的 objection，那么当前 phase 会停止执行，等待该 objection 被撤销。只有在 objection 被撤销后，当前 phase 才会继续执行。如果某个 phase 没有遇到举起的 objection，那么这个 phase 就会顺利执行完毕，然后进入下一个 phase。待所有的 phase 均执行完毕之后，验证平台通过调用 finish 任务来结束仿真。通过这种方式，UVM 能够实现复杂的控制流逻辑，对验证过程进行精确的控制，确保只有在满足特定的条件或者事件后，验证过程才会继续进行，也使得验证过程更加灵活和可控。

UVM 的 phase 和 objection 机制的协同作用使得整个验证过程能够自动、有序地进行，从而避免了运行顺序混乱可能带来的问题。这种机制不仅提高了验证过程的效率和准确性，还简化了代码结构，使其更易于理解和维护。

3　片上网络路由器的验证方案设计

本文基于路由器的工作环境和功能，通过搭建 UVM 验证平台的方式，对路由器的 5 个端口分别施加不同的受约束的随机激励，并收集其产生的输出值。然后，根据搭建的参考模型计算出正确的期望值，并将参考模型的期望值与 DUT 的实际值进行逐步对比。此外，需要对路由器进行异常测试、压力测试和性能测试。在大量测试之后，如果参考模型和 DUT 的输出值对比无误，则确认覆盖率是否满足预期。若覆盖率满足预期，则验证工作基本完成。

3.1　验证平台的设计

考虑到路由器的运作环境的重要性，因此本文设计的测试用例主要侧重于请求 flit 和响应 flit 相互交织的情况。根据通信协议的特性来编写 transaction 组件、sequence 组件和 driver 组件，并利用自动化脚本来生成 UVM 基本的系统框架。本文验证工作有两大难点：一是如何让参考模型实现与 DUT 相同的功能；二是如何在验证平台上对路由器的其他两项功能进行验证。为了解决这些难题，本文将功能（2）和功能（3）的验证工作放在 scoreboard 组件中进行，而参考模型所需实现的功能仅仅是根据路由算法生成期望值。一般来说，在平台搭建的过程中，scoreboard 组件和参考模型是不包含时钟和复位信号的。但是，在测试过程中，需要考虑 DUT 复位之后的运行情况。因此，在环境顶层通过使用 config _ db 机制的方法分别将时钟和复位信

号引入其中，便于测试复位之后的 DUT 能否正常工作。

如图 3 所示，在该验证平台中，因为每个方向发送给路由器的激励均不相同，所以根据输入给路由器的数据包所属方向来划分 agent 组件，共 5 个 agent。每个方向的 agent 的作用一致，都是对各自的 sequencer、monitor、driver 组件进行封装，而 o _ agent 中仅封装 monitor 组件。在每个 agent 中，sequence 将激励装载到 sequencer 上，sequencer 将 sequence 生成的事务级数据发送给 driver。driver 组件将激励发送给 DUT，monitor 组件监测激励并通过 FIFO 将其发送给参考模型。参考模型和 DUT 分别根据接收到的数据来进行路由计算，产生各自的输出值。在参考模型中将生成的期望值写入与 scoreboard 对接的 FIFO 中，而 o _ agent 组件封装的 monitor 用于监测 DUT 产生的输出值并将其传至 scoreboard 组件。scoreboard 组件接收两者的输出值并将两者的数据进行对比，统计测试结果。

在验证环境中，通过 interface 来模拟 DUT 的输入端口和输出端口。由于一些接口信号在时序上需要满足特定的情况，采用在 interface 中添加断言的方式来保证信号之间的时序要求[16-17]。

3.2 覆盖组和交叉覆盖组的设计

在 UVM 中，覆盖组和交叉覆盖组是用于评估测试覆盖率的两种重要机制，它们能够准确地报告路由器在性能测试中已经覆盖了哪些方面。以本地输入的数据信号为例，首先，由于数据信号的控制域包括头尾标志域、路由类型域、包类型域和路由信息域，因此，覆盖组需要确保每个控制域都得到了取值覆盖，以及在多个时钟周期内数值的跳转覆盖。其次，为了全面测试路由器的运行模式，需要设计交叉覆盖组。例如，路由器会遇到 5 个端口同时输入广播包或单播包的情况，或者一个端口输入多播包而另一个端口输入单播包等情况。对于每一种运行模式，都需要编写一个交叉覆盖组进行覆盖，以此来确保测试的完备性。

3.3 参考模型组件的设计

在验证平台中，参考模型提供了对 DUT 行为的预期或理想结果。在进行仿真测试时，参考模型可以用于生成期望值，这些期望值是与 DUT 的实际输出进行比较的基础。通过将 DUT 的输出与参考模型的期望值进行比较，可以评估 DUT 的行为是否符合预期，也可以帮助识别 DUT 的任何错误或异常行为，从而发现可能存在的缺陷或错误。

在该组件中，通过开辟多个并行运行的进程和队列来提高参考模型的工作效率。因为在仿真过程中可能会出现不同输入端口的 flit 相互竞争一个输出端口的情况，所以需要使用多个队列来存放每个 agent 输入的 flit，并保留上一轮仲裁失败的 flit，以避免出现丢包的现象。

在仿真过程中，为了防止复位信号拉低之后，参考模型继续处理复位前保留在队列中的 flit，需要在复位信号拉低的时候，将自定义的数组、队列和标志信号进行清零。只有在复位信号拉高的时候，参考模型才会依次处理来自 5 个方向的请求或响应包，并将生成的期望值写入与 scoreboard 组件对接的 FIFO 中。处理流程如图 4 所示，首先，由于只有头 flit 包含路由信息，需要从队列中取出数据包并进行区分。根据 flit 类型的不同，选择相应的处理流程，通过路由计算得出头 flit 需要从哪个输出端口输出。

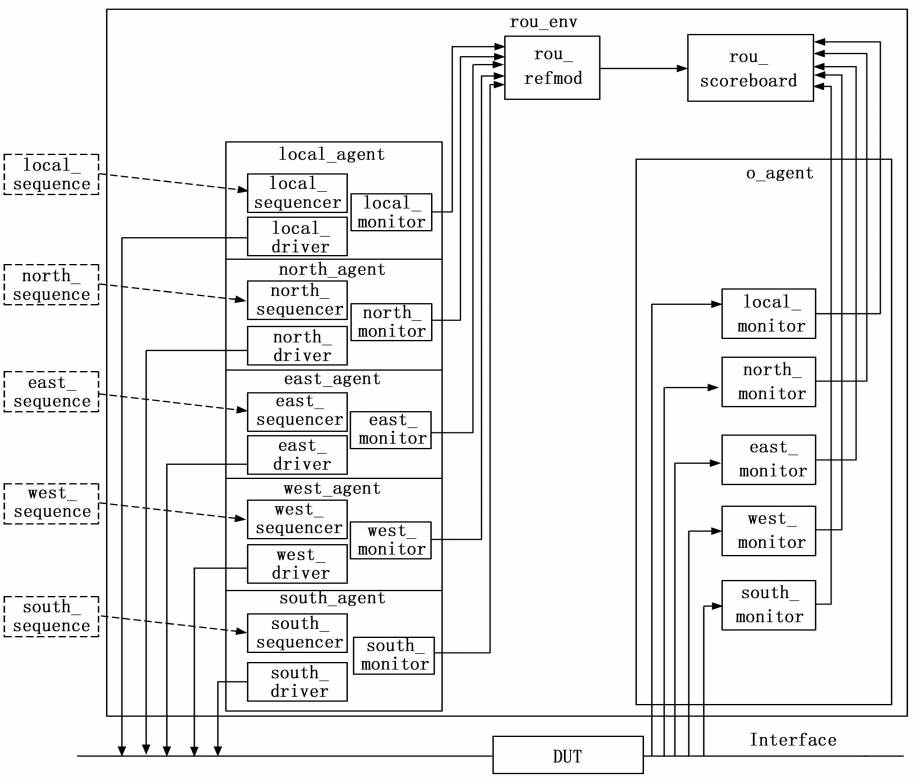

图 3 路由器的 UVM 验证平台

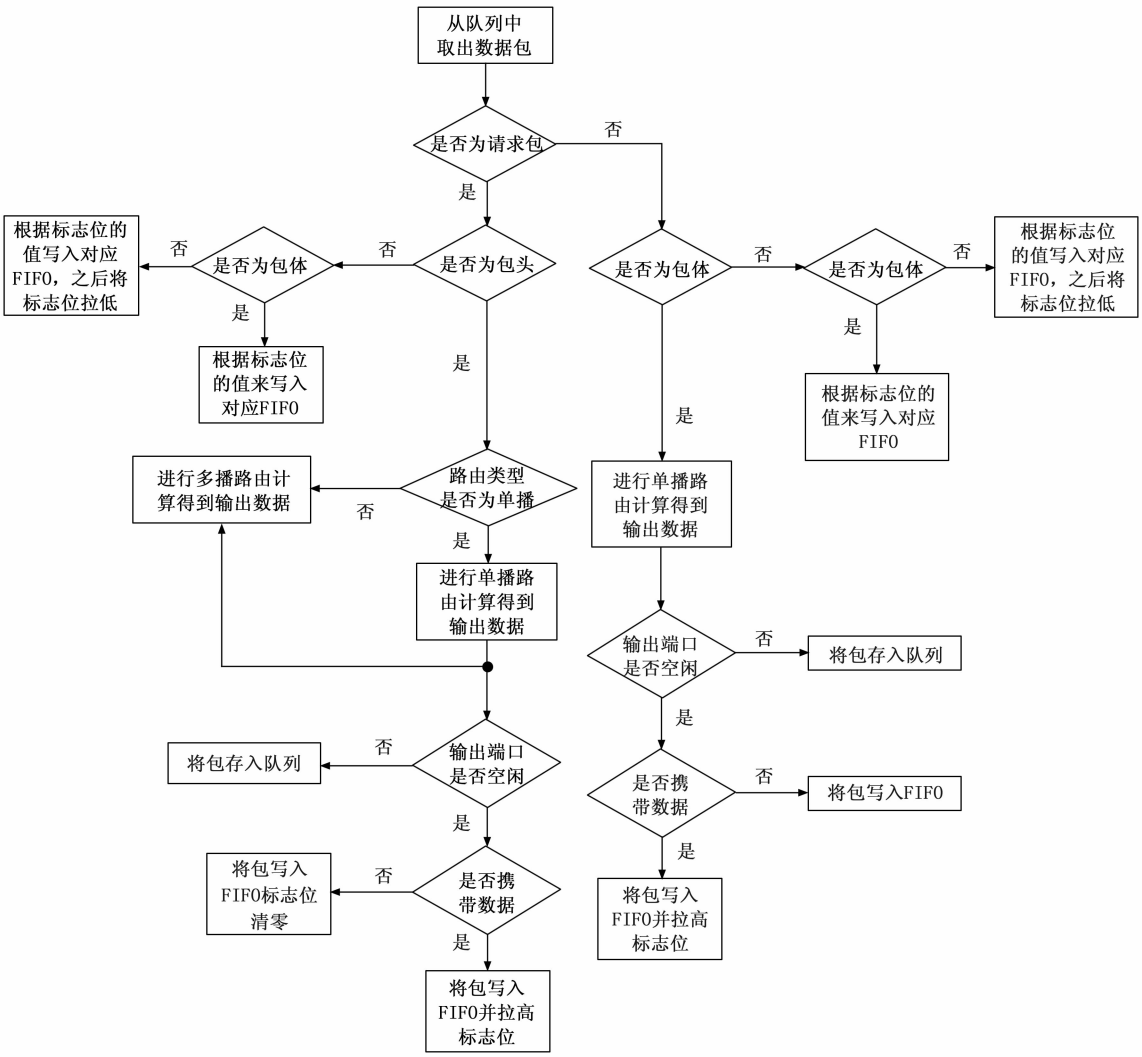

图 4　参考模型的主要工作流程

其次，在对头 flit 进行相应的路由运算之后，需要查看其目标输出端口的空闲情况。由于体 flit 和尾 flit 并不包含路由信息，因此当头 flit 的目标输出端口处于空闲状态时，需要将头 flit 写入与 scoreboard 对接的 FIFO 中，并将标志位置为高。然后根据该标志位将体 flit 和尾 flit 写入同一 FIFO 中。如果头 flit 需要输出的端口处于被占用的状态，那么就将该 flit 重新存入队列中，直到目标输出端口处于空闲状态后再将其写入 FIFO。

最后，因为体 flit 和尾 flit 不包含路由信息，所以二者根据 5 个输出端口标志位的状态来写入正确的 FIFO 中。在尾 flit 处理完毕之后，将端口的标志信号拉低，以避免对下一周期的路由结果造成影响。

3.4　scoreboard 组件的设计

在该组件中，将参考模型和 DUT 的输出数据进行比对，这些数据分别来自本地输出端口、北输出端口、东输出端口、西输出端口和南输出端口。因为来自 5 个方向的输出值对比流程是相同的，所以以本地端口为例来说明 scoreboard 组件的设计流程。

由于每个 flit 都有各自的编号，且在路由的过程中编号不会改变，所以为了准确地跟踪每个 flit 的路由路径，以每个 flit 的编号为索引使用多个关联队列来存放每一个从 FIFO 中取出的 flit。当 scoreboard 收到来自参考模型的输出数据的时候，先根据 flit 的编号来查看存放 DUT 输出 flit 的相应队列是否收到数据，若存放来自 DUT 输出 flit 的同编号的队列不空，则从队列中将数据取出和来自参考模型的输出数据进行比对。若存放 DUT 输出同编号的包的队列为空，则根据参考模型输出包中的编号来存进相应的队列中，待 DUT 收到同编号的数据包之后，再将参考模型的数据从队列中取出进行对比，如果比对通过，则功能（1）验证完毕。

在广播模式下，由于广播包是同一时刻从多个目标端口输出，所以当监测到来自 DUT 的广播包的时候，根据包中的路由信息计算出其他的目标端口并对所有目标端口收到的广播包进行监测，若所有目标端口收到广播包，则将所有输出数据的比对，比对通过则功能（2）验证完毕。

在仿真过程中，为了准确地计算出每个 flit 在 DUT 内

部逗留的时间，需要将输入 flit 通过 FIFO 传递给 scoreboard，在 scoreboard 收到数据包之后，以每个 flit 的编号为索引使用关联数组对每一个 flit 进行计时。当收到来自 DUT 输出的同编号的 flit 时，停止计时并且核实计时的结果是否在限度之内，如果超出设定的阈值，那就说明路由器的超时功能存在问题。若所有数据包都未超过设定的上限，则功能（3）验证完毕。

3.5 测试用例的设计

在正式开始编写测试用例之前，先通过冒烟测试的方法快速验证 DUT 的基本功能是否存在重大缺陷以及整个验证平台能否正常运行。若仿真通过，则可以根据 DUT 的具体工作场景来编写测试用例。

在本次测试用例中，使用随机种子，其中受随机约束的种子个数在 2 000～3 000 之间。在所给的随机数据中包括 i_LValid（本地端口的输入有效信号）、i_NValid（北端口的输入有效信号）、i_EValid（东端口的输入有效信号）、i_WValid（西端口的输入有效信号）、i_SValid（南端口的输入有效信号）、i_LInfo（本地端口的输入数据）、i_NInfo（北端口的输入数据）、i_EInfo（东端口的输入数据）、i_WInfo（西端口的输入数据）和 i_SInfo（南端口的输入数据）。以本地输入端口为例，i_LInfo 接口信号在测试的时候分为控制域和信息域。将这两个部分进行随机约束遍历，进行这一操作是为了对路由器的功能进行详细地测试，并且为后期覆盖率的收集工作提供便利。

当路由器收到多播包的时候，会根据输入信号 i_Hopvld 是否拉高来选择是否使用多播路由算法来对数据包进行处理。因此，根据路由器对多播包处理方式的不同，编写了 32 个测试用例，以此来对路由器的功能进行详细的测试。根据路由器在片上网络中的位置的不同，编写了 64 个测试用例，分别对应 mesh 网络中的每一个路由节点。

除了上述基础功能测试和焦点测试，还需对路由器进行压力测试、异常测试和性能测试。压力测试是指在路由器的 5 个输入端口的输入 flit 均从一个输出端口输出数据的情况下，输出 flit 的速率是否能达到预期。异常测试是指在路由器收到错误激励的情况下，能否按照预期的设想，输出正确的报错信号。性能测试是指路由器从复位信号拉高到仿真结束，5 个输出端口输出 flit 的平均速率是否达标。

为了在验证平台的顶层中能够更加精细地控制发送给 DUT 的激励类型，以上编写的所有测试用例，均使用宏来控制顶层向 sequence 组件传递的参数值，之后，在 driver 组件中进行数据的拼接，依次发给 DUT。

4 实验结果与分析

在验证平台搭建完成之后，使用 VCS 和 Verdi 进行仿真验证，根据输出的波形和仿真日志来确保测试用例的通过率[18]。在仿真测试的时候，通过编写覆盖组和交叉覆盖组来对输入输出数据进行收集，确保随机遍历的可靠性。以本地数据信号 i_LInfo 为例，随机约束遍历情况如表 1 所示。在测试用例中的随机数据包全部使用完毕并且平台仿真结束之后，需要查看 FIFO 中是否有未处理的数据包残留，以及 scoreboard 组件中的队列是否为空。如表 2 所示，以本地数据端口为例，经过多次回归测试之后，环境中的 FIFO 以及队列均为空。

表 1 本地端口数据信号遍历情况

信号名称	位数	含义	遍历情况
i_LInfo	[1:0]	头尾标志域	遍历 2'b11、2'b10、2'b00、2'b01
	[2]	包类型域	遍历 1'b0、1'b1
	[4:3]	路由类型域	遍历 2'b00、2'b01
	[68:5]	路由信息域	遍历 64'hffff_ffff_ffff_ffff、64'hffff_ffff、64'hffff_ffff_0000_0000、[16'h1-16'hffff]

表 2 各个 FIFO 内数据情况

FIFO 名称	功能	残留数据的数目
local_agt2refmod_fifo	将本地输入端口的数据传递给参考模型	0
local_refmod2chk_fifo	参考模型输出的本地方向的期望值传递给 scoreboard	0
local_agt2chk_fifo	DUT 输出的往本地方向的输出值传递给 scoreboard	0
local_agt2chk_fifo_overtime	将本地方向的输入数据传递给 scoreboard	0

信号名称位数含义遍历情况 i_LInfo [1：0] 头尾标志域遍历 2'b11、2'b10、2'b00、2'b01 [2] 包类型域遍历 1'b0、1'b1 [4：3] 路由类型域遍历 2'b00、2'b01 [68：5] 路由信息域遍历 64'hffff_ffff_ffff_ffff、64'hffff_ffff、64'hffff_ffff_0000_0000、[16'h1-16'hffff]。

功能覆盖率和代码覆盖率是验证评估的标准。代码覆盖率表示待测模块源代码被覆盖程度，功能覆盖率是用户自己定义的一些覆盖组，用于度量仿真时执行了多少设计规范中的待测模块[19-20]。如果代码覆盖率很高，那么大部分代码都已经被测试用例执行过，这表明测试的质量可能较高。但如果功能覆盖率很低，可能意味着虽然代码被测试了，但可能没有满足所有的功能需求，这可能意味着路由器的实现并未完全按照功能规格文档来实现。此时，可能需要检查自定义的覆盖组是否正确，或者激励是否存在问题。另一方面，如果功能覆盖率很高但代码覆盖率很低，这可能表明虽然大部分功能都被测试了，但实现这些功能的代码并没有被充分测试。这可能是因为路由器的设计代码存在冗余项，即有些代码路径在仿真过程中并未被使用到。这种情况下，可能需要审查路由器的设计代码，以确定是否存在冗余项或者无效代码。

依次仿真环境中编写的所有测试用例，对各自的覆盖率进行收集，对于一些不能达到的功能点，通过编写定向的测试用例来覆盖。待每个覆盖率收集完成之后，合并每个测试用例收集到的覆盖率。表 3 所展示的是本地方向的

功能覆盖率数据，每个覆盖组和交叉覆盖组代表路由器的接口覆盖情况。路由器的代码覆盖率为 95.6%，功能覆盖率为 100%。经过追踪检查，发现代码覆盖率没有达到 100%的原因是设计中存在不会被覆盖使用的冗余代码。因为这种情况不会对路由器的功能产生影响，所以代码覆盖率达到了预期。

表 3　功能覆盖率的结果

Covergroups	Hit Count	Hits/%	Goal/%	Coverage/%
i_LValid_c	286 562	100	100	100
i_LInfo0	285 662	100	100	100
i_LInfo1	286 562	100	100	100
i_LInfo2	280 218	100	100	100
i_LInfo3	286 772	100	100	100
i_LInfo4	286 562	100	100	100
s_cross1	15 031	100	100	100
s_cross2	14 626	100	100	100
s_cross3	14 776	100	100	100
s_cross4	2 341	100	100	100
s_cross5	2 146	100	100	100
s_cross6	1 456	100	100	100
s_cross7	250	100	100	100
s_cross8	237	100	100	100
s_cross9	189 170	100	100	100
s_cross10	4 573	100	100	100
s_cross11	249 394	100	100	100
s_cross12	862	100	100	100
s_cross13	7 910	100	100	100

5　结束语

本文提出了一种基于 UVM 的片上网络路由器的验证平台的构建方案，针对路由器的功能，设计了验证方案，并通过详细的测试来确认路由器的功能是否有误。该验证环境利用了 SystemVerilog 面向对象的编程结构的优点来提高验证平台中每个组件重用性，通过划分多个 agent 向每个端口发送受约束的随机激励，对路由器进行详细的测试。本文通过运用覆盖率驱动策略，量化了验证进程，通过编写多个覆盖组和交叉覆盖组的方法来收集覆盖率，经过 VCS 和 Verdi 的联合仿真，实现了功能覆盖率达到 100%的验证目标，提高了验证的完备性和准确性。

此外，该验证环境和测试用例均可实现更高验证层级的复用，大幅缩短了后续整个片上网络的验证周期和片上系统芯片的开发周期，对后续整个片上网络的验证平台搭建具有重要参考价值。

参考文献：

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

[2] HO R, MAI K W, HOROWITZ M A. The future of wires [J]. Proceedings of the IEEE, 2001, 89 (4): 490 - 504.

[3] EIAAS A S, IBRAHEM M A, ELMOHR M A, et al. A reusable verification environment for NoC platforms using UVM [C] //IEEE EUROCON 2017-17th International Conference on Smart Technologies. IEEE, 2017: 239 - 242.

[4] 田晓旭，徐庆阳，汤先拓，等. 基于 UVM 的寄存器验证自动化方法 [J]. 集成电路应用，2020，37 (2)：18 - 21.

[5] DENG Q, ZHU P, XI J. The implementation of universal verification platform for sub-model of DBF system based on UVM [J]. Microelectronics & Computer, 2018, 35 (1): 115 - 117.

[6] MELIKYAN V, HARUTYUNYAN S, KIRAKOSYAN A. UVM verification IP for AXI [C] //2021 IEEE East-West Design & Test Symposium. IEEE, 2021: 1 - 4.

[7] SALAH K. A UVM-based smart functional verification platform: concepts, pros, cons, and opportunities [C] //2014 9th International Design and Test Symposium (IDT). IEEE, 2014: 94 - 99.

[8] BJERREGAARD T, MAHADEVAN S. A survey of research and practices of network-on-chip [J]. ACM Computing Surveys, 2006, 38 (1): 30 - 34.

[9] 钟文枫. SystemVerilog 与功能验证 [M]. 北京：机械工业出版社，2010.

[10] 张　强. UVM 实战 [M]. 北京：机械工业出版社，2014.

[11] XIONG T, JIANG J. Self-verification of CAN module based on UVM [J]. Microelectronics & Computer, 2016, 33 (9): 93 - 97.

[12] JIAYI W, NIANXIONG T, YANGFAN Z, et al. A UVM verification platform for RISC-V SoC from module to system level [C] //2020 IEEE 5th International Conference on Integrated Circuits and Microsystems (ICICM), 2020: 242 - 246.

[13] NI W, WANG X. UVM based function coverage-driven SDIO IP verification [J]. Microelectronics, 2017, 47 (3): 392 - 395.

[14] 孙晓东，王治强. 集成电路 UVM 验证环境典型结构设计 [J]. 测试技术学报，2022，36 (2)：135 - 140.

[15] 谢　峥，王　腾，雍珊珊，等. 一种基于 UVM 面向 RISC CPU 的可重用功能验证平台 [J]. 北京大学学报（自然科学版），2014，50 (2)：221 - 227.

[16] SOHOFI H, NAVABI Z. Assertion-based verification for system-level designs [J]. IEEE, 2014: 582 - 588.

[17] 李森森，张立朝，徐金甫. 一种基于断言的高效验证实现方法 [J]. 微电子学与计算机，2014，31 (4)：128 - 266.

[18] 隋金雪，张　霞，郁添林. 基于 UVM 的 AXI4 总线自验证平台设计 [J]. 计算机仿真，2023，40 (1)：345 - 348.

[19] JAIN P, SHAH M V, PATEL B. Automated verification system for functional coverage extraction [C] //2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018: 1870 - 1875.

[20] GOEL A, SUNDARI B T, MATHEW S. UVM based controller area network verification IP (VIP) [C] //2020 International Conference on Smart Electronics and Communication (ICOSEC). IEEE, 2020: 645 - 652.