

引用格式:任传宝,崔建国,鲁迎春,等.应用直接编程接口技术提高片上系统的 UVM 验证重用性[J].微电子学与计算机,2021,38(6):20-26. [REN C B,CUI J G,LU Y C,et al. Using direct programming interface technology to improve the reusability of UVM verification of system on chip[J]. Microelectronics & Computer,2021,38(6):20-26.]

应用直接编程接口技术提高片上系统的 UVM 验证重用性

任传宝,崔建国,鲁迎春,黄正峰,易茂祥
(合肥工业大学 电子科学与应用物理学院,安徽 合肥 230009)

摘 要:提出一种提高片上系统的 UVM 验证重用性方案,应用直接编程接口技术,实现通用验证方法学和 C 语言程序的交互通信.该方法不仅降低了通用验证方法学使用的复杂度,而且使得 C 语言测试用例可以在不同的测试层次中移植重用,例如 C 测试代码可以从模块级到片上系统级的重用.以 SPI 控制器验证本方案,搭建其 UVM 验证平台,编写大量的 UVM 和 C 测试用例,使其功能覆盖率达到 100%,并通过虚拟处理器方案保证 C 测试用例从模块级复用到系统级的可行性.实验过程中激励开发简单且调试方便,可以实现 UVM 环境和测试用例的重用,从而提高片上系统的 UVM 验证重用性,达到缩短芯片开发时间的目的.

关键词:通用验证方法学(UVM);重用性;直接编程接口(DPI);片上系统芯片(SOC);虚拟处理器
中图分类号: TN47 **文献标识码:** A **文章编号:** 1000—7180(2021)06—0020—07

Using direct programming interface technology to improve the reusability of UVM verification of system on chip

REN Chuan-bao,CUI Jian-guo,LU Ying-chun,HUANG Zheng-feng,YI Mao-xiang
(School of Electronic Science & Applied Physics, Hefei University of Technology, Hefei 230009, Anhui China)

Abstract: This paper proposes a scheme to improve the reusability of UVM verification for system on chip. The direct programming interface (DPI) technology is applied to realize the interactive communication between universal verification methodology and C-language program, which not only reduces the complexity of using universal verification methodology, but also makes C-language test cases portable and reused in different test levels, for example, C-test code can be reused from module level to system on chip level. The SPI controller was used to verify the scheme, the UVM verification platform was built, a large number of UVM and C test cases were written to achieve 100% functional coverage, and the feasibility of C test cases being reused from module level to system level was guaranteed through the virtual processor scheme. During the experiment, the incentive development is simple and debugging is convenient. It can realize the reuse of UVM environment and test cases, to improve the reusability of UVM verification of system on chip and shorten the development time of chip.

Key words: Universal Verification Methodology(UVM); Reusability; Direct Programming Interface(DPI); System On Chip(SOC); Virtual Processor

1 引言

随着集成电路工艺的快速发展,SOC 的功能和

复杂度呈现爆炸性的增长,芯片验证的难度也越来越复杂.由于芯片的零缺陷容忍度和高昂的流片费用,因此在流片之前需要进行全面的验证.如何缩短

芯片的验证周期和提高验证质量变得至关重要. 传统的验证方法已经无法满足芯片验证的需求,迫切需要一种新的集成电路验证技术和方法学^[1]. UVM 的诞生,是在 2010 年 UVM 发展委员会提出从 OVM(Open Verification Methodology)基础中构建 UVM 的第一个版本 UVM1. 0(Early Adopter),该版本几乎是 OVM 的直接跨接. 随着 UVM 的快速发展,它汲取了 VMM(Verification Methodology Manual)的特性例如寄存器层,以及其它被证明有效的验证理论. 目前,UVM 标准已经制定到 UVM1. 2^[2],并在 2017 年被 IEEE 宣布为正式标准,并且许多 EDA 厂商均支持该方法学. 相较于传统的验证方法,其最突出的优点就是可重用的验证环境、产生随机约束激励和覆盖率驱动^[3],有效的提高了验证的完备性和效率.

自 UVM 推出以来,国内外就有大量的学者对 UVM 展开了各方面的研究和应用. 田茂源利用 UVM 对 AMBA (Advanced Microcontroller Bus Architecture,片上总线协议)协议转换桥进行了模块验证^[4],不仅缩短了验证周期,而且提高了验证平台的灵活性和可重用性;谢峥等人基于 UVM 对 PKU-DSPII RISC CPU 进行了系统级验证^[5],大幅度提高了覆盖率收敛效果和可重用性;另外,K.

Salah 为 Flash 存储器设计了基于 UVM 的验证结构,并能复用到其他设计中^[6];等等. 但是这些验证平台均只能实现验证环境的复用,而无法实现测试用例的复用. 研究发现,总验证时间的 22%和 23% 分别花费在测试平台构建和测试方案编码上,而 36%花费在调试时间上. 换句话说,准备测试的总时间相对大于实际的验证工作时间;45%比 36%. 这就是为什么要改善组成测试平台和测试用例的重用性的原因. 本文提出一种新方案,该方案引入直接编程接口技术,实现了 UVM 的验证环境和 C 代码交互通信. 这样就能在模块级开发 C 的测试用例并将其重用到系统级,从而缩短 SOC 的总开发时间.

2 基于 UVM 的验证平台

2.1 主要通用验证组件

UVM 是一个以 System Verilog 类库为主体的验证平台开发框架. 基于 UVM 的验证平台就是按照 UVM 提供的标准框架和相应的基类来构建可重用的通用验证组件并建立相应组件之间的通信关系. 这些通用验证组件具有可配置性、封装性、可重用性高等优点. 每个验证组件都有特定的功能,包括生成激励,检查,驱动接口,采集信号等. 构成 UVM 验证平台的主要组件和其功能如表 1 所示.

表 1 主要通用验证组件及其功能

组件名称	组件功能
Sequence	产生受约束的随机激励,需将其挂载到相应的 Sequencer 上
Sequencer	管理着 Sequence,将其产生的随机数据通过事务级传输模型(TLM)传送给 Driver,并带有仲裁特性
Driver	将接收的事务级数据,根据特定的协议转换为待测设计的时序激励
Monitor	监测接口上的数据,通过事务级传输级模型(TLM)传送给 Scoreboard 或其他组件
Agent	将 Driver、Monitor、Sequencer 三个组件进行封装,以便于更高层次的复用
Scoreboard	接收来自多个 Monitor 的监测数据,进行比对和报告. 它决定着测试用例是否通过
Environment	将 Agent、Scoreboard、配置等组件进行封装,以便更高层次的复用
Test	封装特定的 Environment、Sequence
Top_module	实例化待测设计和接口,并将将它们连接起来

2.2 典型 UVM 框架

典型的 UVM 框架能产生待测设计需要的受约束的随机激励,并且自动将期望值和实际值作比较,同时遵循“高内聚,低耦合”的设计原则,使验证框架中各个组件之间界限分明. UVM 的两大特点,一是可重用组件和标准化框架,二是 System Verilog 基类库. UVM 框架中有两个主要分支,一个是定义验证组件,如 Environment、Agent、Scoreboard、Driver、Monitor、Sequencer 等,它们都是派生于 uvm_

component 基类,是框架的基本构成. 另一个是流动在各个组件之间的数据对象,如 Sequence、Transaction 等,它们是派生于 uvm_object 基类,不属于框架中的组件. 典型的 UVM 框架如图 1 所示,在 Top_module 层利用特定的 Interface 将软件世界的验证环境和硬件世界的待测设计相连接,其中 Test 层是验证平台运行的起点. 关于数据通路,Sequence 是信息流的起点,它产生事务级的数据将其发送给 Sequencer. Sequencer 接收到事务级数据后将其放

入仲裁队列里,然后根据 Driver 是否申请事务级数据来发送数据. Driver 则是将接收到的数据驱动到相应的 Interface 上,将事务级数据转换为时序信号来作为待测设计的激励.同时 In_agent 中的 Monitor 组件会对 Interface 上的信息进行采样,然后做广播,传给 Reference Model 组件得出期望值,最后和 Out_agent 中的 Monitor 组件采样到的实际值作比较.以上表明,UVM 平台框架功能分层,各组件功能明确且相互独立,提高了验证平台的规范性和可重用性^[7].

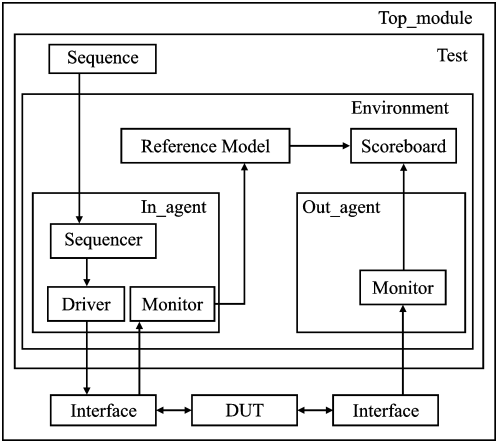


图 1 典型的 UVM 框架

3 直接编程接口

随着方法学和验证的进步,出现 UVM 与其他语言混合验证的新要求,直接编程接口的应用越来越广泛,在硬件和软件跨边界通信场景应用中扮演着重要角色.直接编程接口是 UVM 和其他编程语言的媒介,在验证平台中集成外部程序,如 C/C++ 程序,其重要目的在于重用一些现成的,稳定的模型,例如经验证过的物理层编解码模块的 C/C++ 算法模型,可以作为 RTL 验证平台中的参考模型,提供期望的数据结果^[8];还有就是本文的重点,通过直接编程接口实现 C 测试用例的更高层次重用.直接编程接口如何灵活的重用多语言验证组件,搭建 SOC 多语言协同验证平台已然成为热点^[9].现在多语言的解决方案分为两种:一种是基于 DPI 直接引用外部语言程序的方法^[10];另一种是使用 EDA 厂商提供的多语言库的方法^[11].使用直接编程接口可以很方便地连接 C 代码与 UVM 验证环境,基于该方法可扩展多种接口,有效提高验证效率和验证平台的可重用性^[12].如图 2 所示,描述了 UVM Sequence 和 C 之间的转换层,UVM 验证环境的 Se-

quence 组件可以通过该转换层调用 C 测试用例,然后将激励,配置信息挂载到 Agent 组件中,从而发送给待测设计(DUT),以达到使用 C 测试用例目的.通过这种方式降低 UVM 的使用复杂度;使 UVM 的验证环境支持 C 测试,从而便于将 C 的测试用例在不同的测试层次中移植复用.

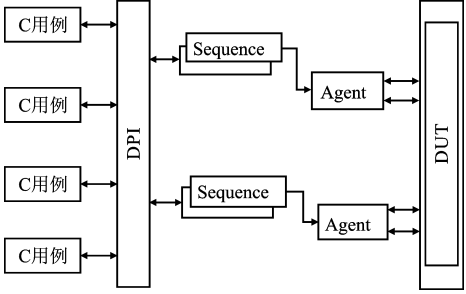


图 2 C 和 UVM Sequence 之间转换层

4 基于 UVM 和直接编程接口的 SPI 控制器的验证方案

4.1 基于 APB 的 SPI 控制器简介

SPI 是由摩托罗拉公司在 20 世纪 80 年代中期针对短距离通信提出的一种同步串行通讯接口^[13].本文研究的 SPI 控制器基于 APB(Advanced Peripheral Bus)总线,基于 APB 的 SPI 控制器是 SOC 或 MCU(Micro Controller Unit)的重要组成部分,通过其可实现高速通信的系统^[14].APB 总线是 AMBA 总线的一种,主要应用在低带宽的外设上,如 SPI、UART(Universal Asynchronous Receiver/Transmitter).MCU 或 SOC 可通过 APB 总线对 SPI 模块进行寄存器配置、数据读写和中断处理等操作,SPI 控制器则能够使 MCU 或 SOC 与外设接口进行数据串行通信.基于 APB 的 SPI 控制器的结构框图如图 3 所示,包括发送数据寄存器(TX),接收数据寄存器(RX),控制寄存器,主设备时钟生成模块,以及数据串并转换模块等.CPU 或者 DMA(Direct Memory Access,直接存储器访问)可以通过 APB 总线对 SPI 从设备进行读写.

4.2 多语言协同验证方案

SPI 控制器这个待测设计里没有处理器单元,无法验证 C 语言的测试代码复用到系统级的可行性.一种可供选择的方案是挂载一个小而轻便的硬件处理器系统,但是该方案有一些限制,比如硬件处理器的体积过大,接口的连接非常复杂,仿真耗时等.针对上述真实处理器的不足,本文采用虚拟处理器的方案,既能保证 C 在子系统到系统级

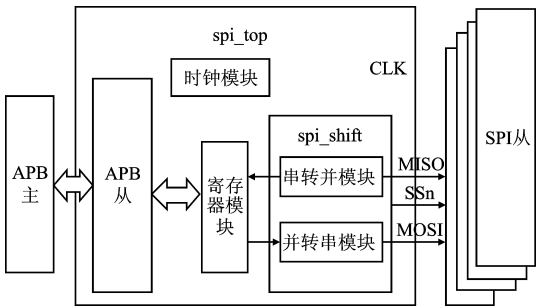


图 3 APB-SPI 结构框图

的复用,又能模拟处理器的行为. 主要模拟处理的行为:各种总线接口的读写、中断响应、时间等待、多核并行处理等. 方案如图 4 所示,Virtual Processor 由两个 core、中断处理模块和 APB UVC(Universal Verification Component)组成,通过 APB UVC 可以实现虚拟处理器与待测设计读写的功能,即总线访问. 中断处理模块接受外设的中断信号模拟处理器的中断响应处理. 工作流程首先通过 gcc 编译器将 C 的代码编译成动态链接文件(.so 文件),这样. so 文件和 UVM 环境在仿真时一起被加载进来,执行时通过直接编程接口为桥梁来模拟真实处理器执行 C 程序的场景. 此方案将 C 和 UVM 的便捷都吸收进来,在虚拟处理器正确地模拟真实处理器的基础上,保证了 C 测试代码实现垂直复用的可行性.

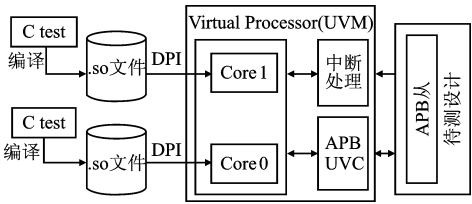


图 4 虚拟处理器的 C 测试框图

4.3 SPI 控制器的验证平台设计

结合图 1 的典型 UVM 框架和图 4 的虚拟处理器方案,本文设计的 UVM 验证平台结构如图 5 所示. 将 APB 总线和 SPI 控制器的相关信号分别封装在 APB Interface 和 SPI Interface 中,通过这两个接口将硬件世界(DUT)和我们的软件环境(验证环境 Test)连接,实现设计和验证环境的交互. 验证环境 Test 包括激励发生器(Sequence)、代理(Agent)、记分板(Scoreboard)、寄存器模型(Reference Model)、功能覆盖率模块、配置模块等 UVM 组件. 其中设计 APB 和 SPI 的 agent,用来模拟真实环境下的 APB 主端和 SPI 从端设备的基本功能,包括读写数

据,指令的产生和传输等. 它们可水平复用,即在其他功能类似的验证平台上稍加修改便可直接进行移植,来减少了平台开发的难度. 寄存器模型是一组高级抽象的类,用来对 DUT 中寄存器和存储器进行建模^[15]. 它反映 DUT 中寄存器的各种特性,产生激励作用于 DUT 并进行寄存器功能检查. 引入寄存器模型,简单高效的实现对 DUT 的寄存器进行前门或后门操作,并且它本生也提供了寄存器测试的 Sequence,方便用户的直接使用. 该平台使用 TLM (Transaction-Level Modeling)通信机制,使得组件之间可以更快地实现数据通信,并且可以保证相邻组件的通信不再通过显式的句柄引用,减少组件之间的耦合性. TLM 是一种基于事务的通信方式,通常在高抽象级语言 SystemC 或 SV/UVM 中作为模块之间的通信方式. 验证环境则通过 Env_config 组件进行配置,比如相关参数、接口传递等. Sequence 组件这是激励的产生部分,每一个 Sequence 则对应一个测试用例,每一个测试用例对应着 DUT 的一个功能点. 每一个 Sequence 需要挂载到 Virtual_sequencer 组件上,在启动仿真时,Sequence 就能匹配上对应的 Sequencer. 该验证结构不仅有 UVM 的测试用例,还有 C 语言的测试用例,其通过直接编程接口将相应的操作传递到接口上,实现数据传递,并通过 Scoreboard 进行比对. 因为系统级的验证有 CPU 的存在,CPU 可以很容易的执行 C 代码,所以该部分 C 语言的测试用例,可以被复用到系统级,并通过图 4 方案进行验证,从而以减少片上系统芯片的开发周期.

5 SPI 控制器的验证平台测试及实验结果

5.1 平台测试用例说明

图 5 所描述的 SPI 控制器的平台结构是在 linux 系统下用 Mentor 公司的 Questasim 10. 6c 版本搭建完成的,该平台针对待测设计的不同功能编写了不同的测试用例,如表 2 所示. 首先进行带约束的随机测试,根据输出的波形和验证报告来判断 DUT 的功能正确与否,并查看输出的功能覆盖率来检测各功能点的实现情况. 如果覆盖率没有达到 100%,需要增加约束或进行定向测试,这种覆盖率驱动的策略,将验证进度进行量化,提高验证效率和完备性. 编写的测试用例分为基于 UVM 和基于 C 语言的;该部分 C 语言测试用例可以复用到系统级供 CPU 直接执行并用虚拟处理器方案验证其可行性,从而提高 UVM 验证重用性,达到缩短片上系统芯片开发周期目的.

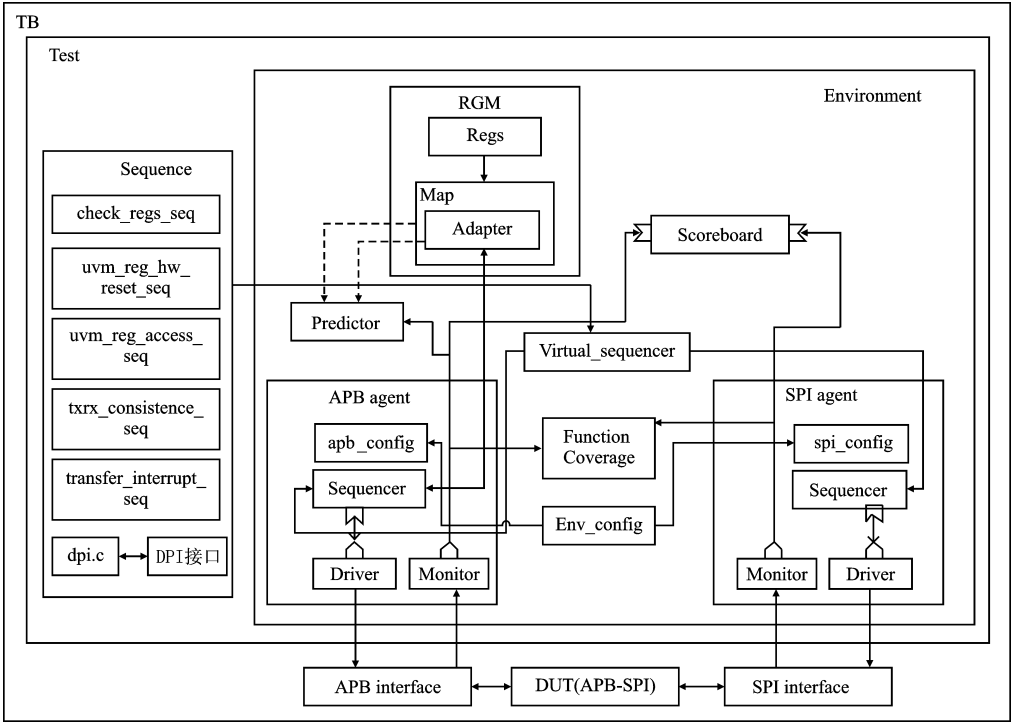


图 5 SPI 控制器的验证平台结构

表 2 测试用例及相关功能说明

用例名称	对应的 sequence	说明
spi_reg_test0	check_regs_seq	利用寄存器模型对寄存器进行前门和后门读写测试,并比对数据是否一致.
spi_reg_test1	uvm_reg_hw_reset_seq	检查寄存器模型复位值是否与硬件实际值一致.
spi_reg_test2	uvm_reg_single_access_seq	先从前门写入寄存器,而后从后门读回数据进行比对;接下来从后门写入寄存器,又从前门读回写入值比对.用来检查寄存器映射的有效性.
txrx_consistence_test	txrx_consistence_seq	检查数据传输的正确性.对发送数据和接受数据的期望值和真实值进行比对.
transfer_interrupt_test	transfer_interrupt_seq	对待测设计的中断功能进行检测.
spi_dpi_c_test	dpi. c	对待测设计进行 C 语言和 UVM 环境的混合测试,并将 C 测试复用更高验证层次.

5.2 实验结果与分析

平台利用 Malefile 脚本,将编译,仿真、测试选择、覆盖率收集等功能设计为自动化流程.在执行脚本时候只需传入测试用例的名称,就能实现自动的编译、仿真和输出验证报告信息.

以表 2 中的 txrx_consistence_test 测试用例为例,进行验证结果分析.该测试用例是检查数据传输的正确性,根据随机的种子会产生 100 组受约束的读写操作指令和数据,并根据待测设计中的时序信息按顺序配置寄存器.第一步,配置时钟分频系数寄存器、控制寄存器除启动域段的其他域段;第二步,配置 TX 寄存器和片选信号;第三步启动,将启动域段配置成 1 可进行数据的传输.第四步,监测启动信号和片选信号,数据传输完毕会释放启动信号和片选信号,则表示传输完成,此时可读取 RX 寄存器.

表 3 中列出几组传输的数据以及期望值和实际

值,第 1 组发送的数据为:0Xc15b5edbc5f6865a025143dac15b5edb,

数据方向为:APB 端到 SPI 端;PWDATA 数据总线上为发送数据的期望值,该期望值:0Xc15b5edb_c5f6865a025143dac15b5ed,mosi 为发送数据的实际值,该值为: 0Xc15b5edb_c5f6865a025143dac15b5edb,期望值与实际值一致. PRDATA 总线为接受数据的实际值,miso 为接受数据的期望值,数据如表 3 第 2 组数据所示,传输方向为:SPI 端到 APB 端.表 3 中第 3,4,5,6 组数据与前两组数据类似且期望值和实际值一致,这里不再赘述.实验平台输出验证报告如表 4 所示,“Test Passed - 100 tansfers occured with no errors”表明 100 组数据传输无错误,所有数据通过 Scoreboard 模块比对正确. “UVM_ERROR:0、UVM_WARN-ING:0、UVM_FATAL:0”表明整个仿真过程中无

语法,数据比对等错误. 验证工作的结束需要功能覆盖率为 100%. 功能覆盖率模块对功能点进行建模,根据接口上的数据仿真器会计算出覆盖率. 本实验中,首先将表 2 中 UVM 的测试用例仿真一遍;然后通过仿真命令将跑过的测试用例的覆盖率合并,得到总的功能覆盖率;查看覆盖率报告,如果功能覆盖率没有达到 100%,此时需要更换随机种子再跑仿真.

随机测试很难达到的场景,采用定向测试,开发了 C 语言的测试用例,如表 3 中第 7 组和第 8 组数据. 运用图 4 的方案,将激励发送给 DUT,配置过程和 txrx_consistence_test 测试用例类似,这里不再赘述. 由表 3 数据,两个方向上的数据,期望值和实际值比对一致,且验证报告上没有错误. 此测试用例

通过虚拟处理器方案运行无误,所以可以保证其复用到片上系统(真实处理器环境)时是可行的,从而达到重用该测试用例,缩短芯片开发时间的目的.

运行不同测试用例的仿真,或者运行同一个测试用例多次,会生成独一无二的覆盖率数据. 将所有的测试用例的覆盖率合并,然后来判断覆盖率是否达到验收要求. 表 5 所示为总覆盖率结果,由 questasim 工具生成,每个覆盖组对应于 DUT 的一个功能点,并已在验证环境中对它们进行建模. 通过运行大量的测试用例,使得覆盖组中的每一个覆盖点都能命中,如图所示, Hits%=100%、Coverage%=100%,说明验证中每一种情况都已覆盖,验证工作达到目标.

表 3 传输数据的期望值及实际值

组数	传输方向	传输数据(128bit)	期望值(128bit)	实际值(128bit)	比对正确?
1	APB 端到 SPI 端	0Xc15b5edbc5f6865a025 143dac15b5edb	0Xc15b5edbc5f6865a025 143dac15b5edb	0Xc15b5edbc5f6865a025 143dac15b5edb	正确
2	APB 端到 SPI 端	0Xb123c33456bca77bd88 ac7fb56cbad6e	0Xb123c33456bca77bd88 ac7fb56cbad6e	0Xb123c33456bca77bd88 ac6fb56cbad6e	正确
3	APB 端到 SPI 端	0Xdbc6ad67cb789610ad6 db78a5dc90da	0Xdbc6ad67cbd789610ad 6db78a5dc90da	0Xdbc6ad67cbd789610ad 6db78a5dc90da	正确
4	SPI 端到 APB 端	0X123adbc4537adbc8adc 7896b78cadce87	0X123adbc4537adbc8adc 7896b78cadce87	0X123adbc4537adbc8adc 7896b78cadce87	正确
5	SPI 端到 APB 端	0Xfe54e56bf78ae9073567 ef45f67afe378	0Xfe54e56bf78ae9073567 ef45f67afe378	0Xfe54e56bf78ae9073567 ef45f67afe378	正确
6	SPI 端到 APB 端	0Xdab7019f095bfe78e015 f567ef34eadb1	0Xdab7019f095bfe78e015 f567ef34eadb1	0Xdab7019f095bfe78e015 f567ef34eadb1	正确
7	APB 端到 SPI 端(C)	0Xef560034f09049000d 900dbcad09fbc9a	0Xef560034f09049000d 900dbcad09fbc9a	0Xef560034f09049000d 900dbcad09fbc9a	正确
8	SPI 端到 APB 端(C)	0X1200003efa78b98ef 0091af8cb76ad05f	0X1200003efa78b98ef 0091af8cb76ad05f	0X1200003efa78b98ef 0091af8cb76ad05f	正确

表 4 验证报告关键信息

序号	内容	解释含义
1	Test Passed - 100 transfers occurred with no errors	产生的 100 组随机约束的数据传输成功其比对无错误.
2	UVM_INFO : 6590	验证平台共打印 6590 次信息以便代码调试及查看仿真情况.
3	UVM_WARNING : 0	验证平台执行过程中没有警告发生.
4	UVM_ERROR : 0	验证平台执行过程中没有错误发生.
5	UVM_FATAL : 0	验证平台执行过程中没有致命的错误发生.
6	APB_DRV_RUN : 100	APB 端到 SPI 端共产生 100 组随机数据
7	SPI_DRV_RUN : 100	SPI 端到 APB 端共产生 100 组随机数据.
8	SPI_SB_REPORT : 100	Scoreboard 中共进行 100 次数据比对且比对成功.
9	SPI_SB_RXD : 6382	Scoreboard 中共进行 6382 次单 bit 数据比对.

表 5 功能覆盖率结果

Covergroups	Total Bins	Hits	Hits(%)	Goal(%)	Coverage(%)
reg_rw_cov	23	23	100	100	100
combination_cov	32	32	100	100	100
cg_spi_reg_illegal_access	12	12	100	100	100
cg_apb_command	8	8	100	100	100
cg_apb_trans_timing_group	4	4	100	100	100
cg_apb_write_read_order_group	4	4	100	100	100

6 结束语

UVM 验证方法学作为主流的功能验证方法, 尽管在前期搭建验证平台的时间较长, 但是执行效率很高, 可以缩短整个研发的验证时间. 本文以 SPI 控制器作为例子, 将 UVM 和直接编程接口相结合开发其验证环境, 实现多语言协同验证, 并通过虚拟处理器方案验证 C 测试用例垂直重用的可行性. 实验运用覆盖率驱动策略, 将验证进度进行量化, 最终达到功能覆盖率 100% 的验证目标, 提高了验证的完备性和准确性; 并且将 C 语言程序通过直接编程接口和 UVM 验证环境对接. SPI 控制器实验结果表明, 本文所提方案不仅能实现验证环境的复用, 而且测试用例也能实现更高验证层次的复用, 可大大缩短片上系统芯片的开发周期.

参考文献:

[1] 倪伟, 袁琳, 王笑天. 基于 UVM 的 I2S 验证 IP 设计[J]. 合肥工业大学学报(自然科学版), 2018, 41(1): 49-54. DOI: 10. 3969/j. issn. 1003-5060. 2018. 01. 010.

NI W, YUAN L, WANG X T. Design of a UVM-based I2S verification IP[J]. Journal of Hefei University of Technology, 2018, 41(1): 49-54. DOI: 10. 3969/j. issn. 1003-5060. 2018. 01. 010.

[2] 刘斌. 芯片验证漫游指南: 从系统理论到 UVM 的验证全视界[M]. 北京: 电子工业出版社, 2018: 36-58.

LIU B. Walking guide to SoC verification[M]. Beijing: Publishing House of Electronics Industry, 2018: 36-58.

[3] 张明望. 基于 UVM 的 CAN 总线控制器的验证方法研究[D]. 西安: 西安电子科技大学, 2018.

ZHANG M W. Research on verification method of CAN bus controller based on UVM[D]. Xi'an: Xidian University, 2018.

[4] 田茂源. 基于 UVM 的 AMBA 协议转换桥验证实现[D]. 西安: 西安电子科技大学, 2014.

TIAN M Y. Testbench implementation for AMBA protocol conversionbased on UVM verification methodology[D]. Xi'an: Xidian University, 2014.

[5] 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. DOI: 10. 13209/j. 0479-8023. 2014. 031.

XIE Z, WANG T, YONG S S, et al. A RISC CPU oriented reusable functional verification platform based on UVM[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2014, 50(2): 221-227. DOI: 10. 13209/j. 0479-8023. 2014. 031.

[6] SALAH K. A unified UVM architecture for flash-based memory[C]//18th International Workshop on Microprocessor and SOC Test and Verification (MTV). Austin, TX, USA: IEEE, 2017: 1-4. DOI: 10. 1109/MTV. 2017. 10.

[7] 张强. UVM 实战[M]. 北京: 机械工业出版社, 2014: 7-9.

ZHANG Q. UVM of actual combat[M]. Beijing: China Machine Press, 2014: 7-9.

[8] 钟文枫. SystemVerilog 与功能验证[M]. 北京: 机械工业出版社, 2010: 188-200.

ZHANG W F. The art of verification with system verilog [M]. Beijing: China Machine Press, 2010: 188-200.

[9] 王纪, 冯志华. SoC 多语言协同验证平台技术研究[J]. 电子设计工程, 2015, 23(20): 130-133. DOI: 10. 14022/j. cnki. dzsjgc. 2015. 20. 042.

WANG J, FENG Z H. Research of SoC multi-language co-verification platform technology[J]. Electronic Design Engineering, 2015, 23(20): 130-133. DOI: 10. 14022/j. cnki. dzsjgc. 2015. 20. 042.

[10] AYNsLEY J. SystemVerilog meets C++: re-use of existing C/C++ models just got easier[C]//Design and Verification Conference & Exhibition (DVCon). San Jose, CA: Accellera, 2010: 255-262.