计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(7):57−69 [doi: 10.15888/j.cnki.csa.008004]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 UVM 的 AMBA 总线接口通用验证平台①

马　鹏, 刘　佩, 张　伟

(华东计算技术研究所, 上海 201808)
通讯作者: 刘　佩, E-mail: 13554183368@163.com

摘　要: 根据摩尔定律的发展规律, 集成电路的规模越来越大, 单颗芯片可集成的电路越来越复杂. 在一个 SoC 芯片的研发周期中, 前仿验证工作随着芯片功能复杂程度验证难度增加, 导致前仿验证时间不可控, 如何在有限时间内可靠的、高效地完成复杂芯片验证工作是目前面对的问题. 针对这一问题, 本文定制一个基于 UVM 方法学的 AMBA 总线接口通用验证平台, 该平台结构具有可扩展性、验证激励具有随机性、验证结果具有可靠性, 能够支持 AMBA-APB、AMBA-AHB、AMBA-AXI 接口类型的待测模块的验证工作.针对目标可以快速地搭建验证平台, 减少前仿验证的准备工作, UVM 平台能够产生带约束随机数据, 验证结果汇成覆盖率报告, 能够保障验证工作的高效以及完备性.

关键词: 验证; AMBA; APB; AHB; AXI; UVM

引用格式: 马鹏,刘佩,张伟.基于 UVM 的 AMBA 总线接口通用验证平台.计算机系统应用,2021,30(7):57–69. http://www.c-s-a.org.cn/1003-3254/8004.html

UVM-Based Universal Verification Platform for AMBA Bus Interface

MA Peng, LIU Pei, ZHANG Wei

(East China Institute of Computing Technology, Shanghai 201808, China)

Abstract: According to the Moore's Law, the scale of integrated circuits is getting bigger, and the integratable circuits within a single chip are increasingly complex. In a research and development cycle of an SoC chip, pre-layout verification becomes harder with more complex chip functions, taking uncontrollable time. How to reliably and efficiently verify complex chips within limited time represents a challenge to be addressed. In response to this problem, this paper customizes a UVM-based universal verification platform for AMBA bus interface. The platform is equipped with a scalable structure and random verification incentives, achieving dependable results. It can verify the modules to be tested in AMBA-APB, AMBA-AHB, and AMBA-AXI interfaces. In addition, the verification platform can be quickly set up for the target, and the preparation for pre-layout verification is simplified. The UVM-based platform produces random data with constraints, and verification results are converted into coverage reports, ensuring the efficiency and completeness.

Key words: verify; Advanced Microcontroller Bus Architecture (AMBA); Advanced Peripheral Bus (APB); Advanced High-performance Bus (AHB); Advanced eXtensible Interface (AXI); Universal Verification Methodology (UVM)

SoC (System on Chip) 产品研发一般采用正向设计方法, 产品的开发流程包括: 项目策划、系统说明以及行为描述、RTL 描述、前端仿真、后端设计、流片阶段、出样等[1]. 项目策划阶段会对市场需求进行调研, 形成产品规格书. 在系统说明及行为描述阶段, 确定设计对象和目标, 明确芯片功能、内外部性能要求等, 从而定制 RTL 代码. 本文以一款工控 CPU 芯片研制为例, 芯片内部需要集成 CPU 核以及大量的外设 (如 SPI、

① 收稿时间: 2020-11-10; 修改时间: 2020-12-12; 采用时间: 2020-12-18; csa 在线出版时间: 2021-06-30

I2C、UART、CAN、SPI、PCIE、USB、DMA 等), 在 RTL 描述阶段将整体任务划分为各个子模块实现, 根据任务书的需求分析, 通用性 IP 配置达不到目标需求. 因此, 这部分模块只能通过自研的方式实现. 而自研代码相比较于成熟 IP 而言可靠性不高, 风险大, 必须依靠前仿验证保证 RTL 代码功能的完善完备性, 前仿任务艰巨繁重.

传统的验证方法[2], 需要验证人员具备丰富的经验, 人的因素很大, 进度不可控, 难以满足复杂的 SoC 验证需求. 目前, 市场上流行的验证方式是 UVM (Universal Verification Methodology) 验证方法学.

UVM 验证方法学[2], 由 Accellera 于 2011 年 2 月推出, 以 SystemVerilog类库为主体的验证平台开发框架, 继承了 VMM 和 OVM的优点, 构建的验证环境具备可重用组件构建标准化和层次化特性, 并且得到了 Synopsys、Mentor 和 Cadence 公司的支持, EDA 环境构建完善. 它目前已经得到市场的验证, 现在市面上很多 IC 设计公司都已经在使用[1]. UVM 方法学的特点是能够产生带约束的随机测试激励、通过 refm 设定可以判定输出结果是否符合预期, 能够进行大量的随机验证, 平台本身具备可靠性, 加入覆盖语句、断言语句, 获取到代码覆盖率、功能覆盖率、断言覆盖率的百分比, 通过分析能够量化的科学的判定验证的程度, 验证进度可控.

AMBA (Advanced Microcontroller Bus Architecture) 由 ARM 公司研发推出, 定义了一种嵌入式高性能微控制器的片上通信标准[3]. 目前, 已被广泛地应用于大型复杂 SoC 系统中[2].

本文以工控处理器 SoC 芯片验证为背景, 目标芯片内部采用 AMBA 总线作为片上通信的总线架构, 搭建基于 UVM 验证方法学的 AMBA 总线接口的验证平台, 平台具备可靠性、可扩展性、通用性、高效性, 能够减少验证准备工作的重复性, 节约验证时间, 可靠的高效完成整个芯片的验证工作, 保障 RTL 代码功能的正确性和完备性, 提高芯片成功率.

1 基于 UVM 的 AMBA 总线接口验证平台搭建

在 AMBA 3.0 版本中, 介绍了 4 种接口类型, AMBA APB (Advanced Peripheral Bus) 总线[3]、AMBA AHB (Advanced High-performance Bus) 总线[4]、AMBA AXI (Advanced eXtensible Interface) 总线[5] 和 AMBA ATB (Advanced Trace Bus) 总线[6].

目标芯片系统结构图, 如图 1 所示, 片上总线采用 AMBA 总线、内部集成大量低速外设包括多路 UART 控制器、I2C 控制器、Timer 控制器、Watchdog 控制器、PWM 控制器、GPIO 控制器等; 高速设备, L2 Cache 控制器、DMA 控制器、PCIE 控制器、USB 控制器、EMMC 控制器、SMC 控制器等; 基于系统性能设计考虑, 目标芯片系统架构设计采用 AMBA 协议中的 3 种类型, AMBA[7] AXI、AMBA AHB[8] 、AMBA APB 总线协议, 选择高速总线 AMBA AXI 总线作为处理器与外设相连的一级总线, 选择 AMBA AHB[9] 总线作为片上本地总线, 选择 AMBA APB 总线, 连接低速设备. 如图 1 所示, AMBA-AXI 总线上挂接 CPU 核以及高速缓存控制器 AXI-L2Cache 控制器、AXI-SRAM 控制器、AXI-PCIE 控制器等, 这类子模块的验证工作需要平台包含 AMBA AXI 总线驱动; 通过 AXI2AHB 桥控制器, 完成 AMBA AXI 总线与 AMBA AHB 总线桥接工作, AMBA AHB 作为本地高性能总线, 挂接 AHB-DMA 控制器、AHB-SPI 控制器等高速外设, 这类子模块的验证工作需要平台包含 AMBA AHB 总线驱动; 通过 AHB2APB桥控制器, 完成 AMBA AHB 总线与 AMBA APB 总线桥接工作, AMBA APB 总线作为低速外设总线, 挂接低速外设 APB-UART、APB-WDT 等模块, 这类子模块的验证工作需要平台包含 AMBA APB 总线驱动.

本验证平台采用 Agent 以及 TLM 通信机制设计, 实现平台可扩展性, 支持外设接口类型为 AMBA APB、AMBA AHB、AMBA AXI 单一接口或者组合接口多样性的验证工作, 具备通用性, 可操作类型:

1) AMBA APB 接口: 支持外设地址位宽配置、支持 32 位数据读写操作, 符合 AMBA APB 3.0 协议规范.

2) AMBA AHB 接口: 支持外设地址位宽配置, 支持 8 位、16 位、32 位、64 位 SINGLE 读写操作, 支持 BURST INCR 类型读写操作 (len 覆盖 1~16), 符合 AMBA AHB 协议规范.

3) AMBA AXI 接口: 支持外设地址位宽可配置, 支持 8 位、16 位、32 位、64 位 SINGLE 读写操作, 支持 BURST INCR、BURST WRAP、BURST FIXED 类型读写操作 (len 覆盖 1~16), 符合 AMBA AXI 协议规范.

因此, 验证平台能够满足系统内集成的所有子模

块的验证工作, 所包含的总线驱动, 覆盖目标芯片验证所需的全部 AMBA 接口驱动类型, 具备通用性, 能够通过平台结构扩展性满足子模块验证需要的接口类型以及数目的多样性, 例如 AHB-DMA 控制器, 通过 AMBA AXI 接口进行数据传输、AMBA AHB 接口进行控制器寄存器配置, 需要在同一环境集成 AXI 总线驱动以及 AHB 总线驱动才能完成验证工作, 验证平台具备的可扩展性能够满足验证需求.

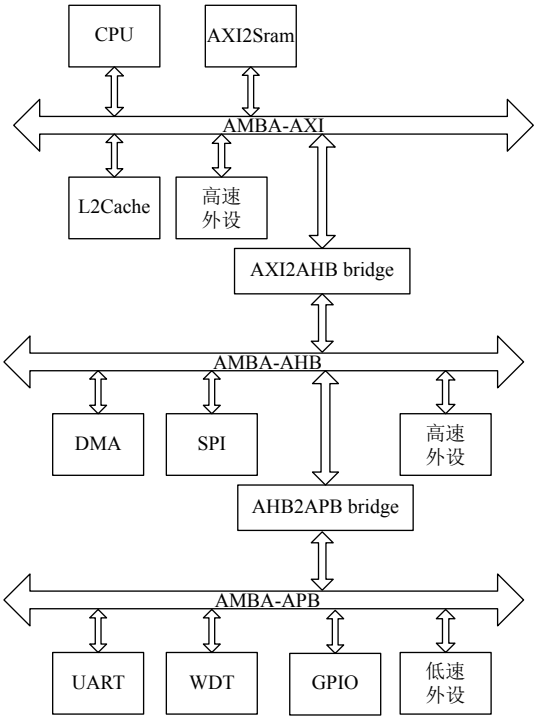


图 1 目标芯片系统结构图

1.1 AMBA APB 总线接口驱动实现

APB 是高级外围总线, 定义了 APB 总线协议, 本平台设计的 APB 总线接口, 支持待测模块地址位宽可配置, 数据接口位宽可配置, 驱动设计符合 AMBA APB 3.0 协议规范.

根据 AMBA APB 协议, 定义 sequence item\monitor transfer, 如表 1 和表 2 所示, 根据`define APB_ADDR_WIDTH 配置支持地址位宽 16 位或者 32 位接口类型, 根据`define APB_DATA_WIDTH 配置支持数据位宽 16 位或者 32 位接口类型; APB 接口驱动设计流程如图 2 所示, 在 VIF.clk 上升沿将激励值赋给对应接口, 在时钟的下一个上升沿, 将 VIF.penable 拉高, 之后, 在每个时钟的上升沿判断 VIF.pready 信号是否拉高, 本节拍等待 VIF.pready = 1'b1 成功将 VIF.prdata 值赋值给 item.pdata, 下节拍上升沿, 将控制信号以及接口地址信号清零, APB 写驱动效果图, 如图 3 所示. APB 接口监控设计, 在 VIF.clk 下降沿, 判断读数据是否有效 (psel == 1'b1 & penable == 1'b1 & pready == 1'b1), 满足条件当拍将 VIF.prdata 赋值给 mtr.prdata, 流程如图 2 所示. APB 读驱动效果图, 如图 4 所示.

表 1 APB sequence item

元素	说明
bit [APB_ADDR_WIDTH – 1 : 0] paddr	操作地址
bit psel	选择信号
bit pwrite	写控制信号(1写0读)
bit [APB_DATA_WIDTH – 1 : 0] pdata	写入\读出数据

表 2 APB monitor transfer

元素	说明
bit [APB_DATA_WIDTH – 1 : 0] prdata	读出数据

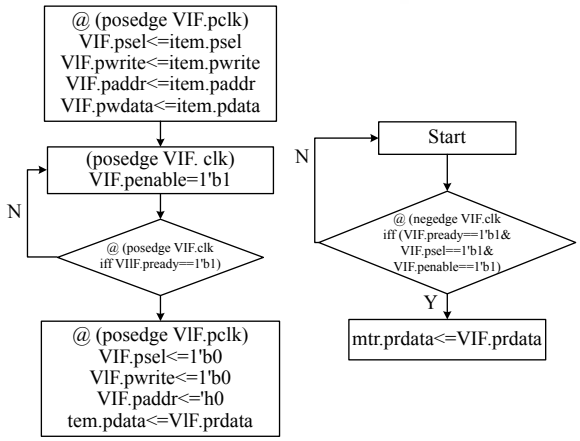


图 2 APB Driver 及 Monitor 设计流程图

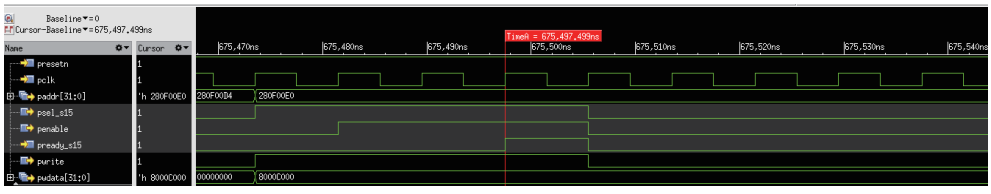


图 3 APB 写效果图

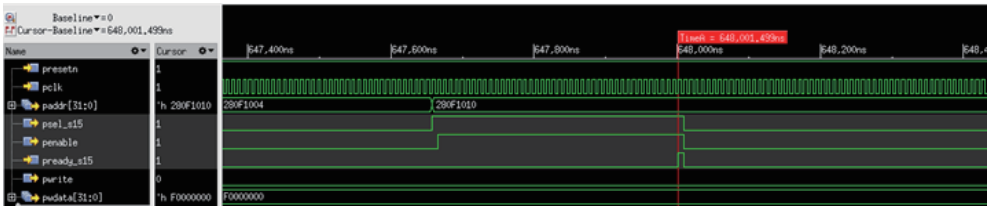


图 4 APB 读效果图

1.2 AMBA AHB 总线接口驱动实现

AHB 是高级高性能总线, 定义了 AHB 总线协议, 本平台设计的 AHB总线接口, 支持待测模块地址位宽 8 位、16 位或者 32 位接口类型, 支持数据 8 位、16 位、32 位、64 位 SINGLE 以及 BURST INCR 访问, 符合 AMBA AHB协议规范.

根据 AMBA AHB 协议规范, 定义 sequence item, 如表 3 所示. 根据 define AHB_ADDR_WIDTH 配置地址位宽 16, 32, 64 位接口类型; AHB 接口驱动设计, 根据协议规范, 在时钟上升沿等待 hready 拉高, 通过 trans_len 判断是否为 SINGLE 操作, 设置 VIF.Hburst 参数值 (AHB_BURST_SINGLE\AHB_BURST_INCR), 并将 sequence item 赋值给对应的接口信号; 通过 item.hwrite 判断当前操作类型, 若值为 WR, 如图 5 所示, 写操作, 在下一节拍上升沿且 hready 为高电平, 当前操作为 SINGLE, VIF.htrans 赋值 AHB_TRANS_IDLE, 若当前操作为 BURST 操作, VIF.htrans 赋值 AHB_TRANS_SEQ; 根据数据单元大小 8 bit\16 bit\32 bit\64 bit 操作, 对地址在 item.trans_len 个 clk 节拍上升沿, 分别进行 VIF.haddr 加 1 或者 item.hsize*2 操作, 同时, 根据 hsize 类型, 进行 hwdata 与 VIF.hwdata 赋值操作, 例如, 若 AHB 发起字节操作, hwdata 低 8 位值赋给 VIF.hwdata[31:0], 且 hwdata 右移 8 位, 等待下个节拍操作, 直到 trans_len 设定的数据传输完成, 在最后一个节拍 VIF.htrans = AHB_TRANS_IDLE, 写操作完成. AHB 写驱动效果图, 如图 6 所示.

若值为 RD, 如图 5 所示, 读操作, 通过 trans_len 判断当前操作, 若是 SINGLE 操作, 在时钟上升沿, 将 VIF.hrdata 值赋值给 item.hrdata, VIF.htrans 赋值 AHB_TRANS_IDLE, 若 BURST 操作, 在时钟上升沿, VIF.htrans 赋值 AHB_TRANS_SEQ, 地址按照数据单元值进行叠加 VIF.haddr = item.haddr + trans_len* item.hsize*2, 根据 trans_len 的值, 进行循环操作, 在最后节拍, VIF.trans = AHB_TRANS_IDLE. AHB 读驱动效果图, 如图 7 所示.

表 3 AHB sequence item

元素	说明
bit [AHB_ADDR_WIDTH – 1:0] haddr	操作地址
bit [3:0] trans_len	数据单元长度
bit [1023:0] hwdata	写数据
bit [1023:0] hrdata	读数据
ahb_transfer_size_e hsize	数据单元大小, 有效值Bit 8、Bit 16、Bit 32、Bit 64
ahb_transfer_direction_e hwrite	操作类型, 1写0读WR:1, RD:0

1.3 AMBA AXI 接口驱动实现

AXI 是高级可扩展接口, 定义了 AXI 总线协议, 具有 5 个独立的传输通路 (读地址通道、读数据通道、写地址通道、写数据通道、写应答通道), 支持乱序传输, 以点对点的方式进行握手交互, 传输具有高效性, 一般应用于处理器与高速设备之间的连接. 本平台设计的 AXI 接口驱动, 支持待测模块地址为位宽 32 位或者 64 位接口了类型, 支持数据 8 位、16 位、32 位、64 位 single 操作, 支持 BURST 类型 FIXED、INCR、WRAP 三种类型、支持 Burst len 覆盖 0~16, 支持读写乱序发送, 符合 AMBA AXI 协议规范.

根据 AMBA AXI 协议规范, 定义 sequence item, 如表 4 所示. 根据 define AXI_ADDR_WIDTH 配置地址位宽 32, 64 位接口类型; AXI 接口驱动设计, 按照操作类型分成 AXI_WR 及 AXI_RD 操作.

AXI_WR 写操作, 如图 8 所示, 首先, 等待 VIF.awready 拉高, 完成写地址通道握手操作; 在下一拍时钟上升拉高 VIF.wvalid 及 VIF.bready, 通过 trans_len != 0, 判断当前操作是否 SINGLE 操作, 若为 SINGLE, 在下一节拍 VIF.wvalid 拉低; 若为 BURST 操作, VIF.wvalid 值不变, 根据 burst 类型 (AXI_BURST_FIXED、

AXI_BURST_INCR、AXI_BURST_WRAP), 选择进入各自逻辑, 根据 trans_len 设置循环操作, 通过 tran_len == item.trans_len–1, 判断是否为最后一次传输, 满足条件, 当前节拍拉高 VIF.wlast, 在下节拍对控制信号进行清零, 等待 slave 应答 VIF.bvalid 拉高, 满足条件, 当前节拍 VIF.bready = 1’b0, 完成写操作, AXI 写驱动效果图, 如图 9 所示. 其中, AXI 涉及的 3 种驱动逻辑.

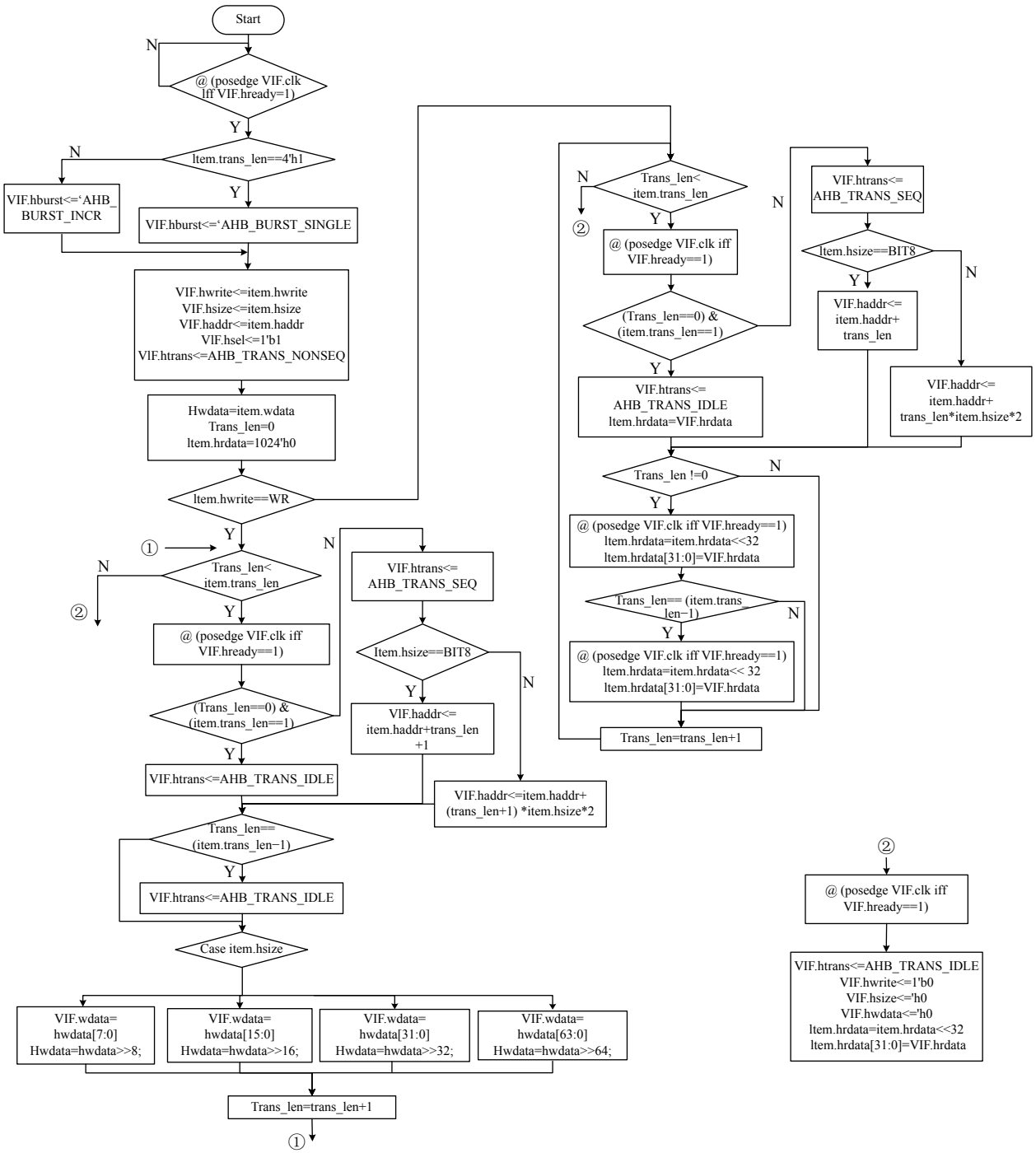


图 5 AHB 驱动设计流程图

1) AXI_BURST_FIXED 类型: 操作访问地址不变, 驱动设计关键在数据选通信号设计, 根据 item.trans_size(8 bit\16 bit\32 bit\64 bit), 若为 8 bit, VIF.wstrb = (8’h1 << item.axi_addr[2:0]); 若为 16 bit,

VIF.wstrb= (8’h3 << item.axi_addr[2:1]*2); 若为 32 bit, VIF.wstrb=(8’hf << item.axi_addr[2]*4); 若为 64 bit, VIF.wstrb = 8’hff, 如图 10 所示.

2) AXI_BURST_INCR 类型: 操作地址以数据单元大小进行累加, 在 fixed 类型基础上增加地址设计逻辑, 若为 8 bit: item.axi_addr= item.axi_addr + 32’h1; 若为 16 bit, item.axi_addr = item.axi_addr + 32’h2; 若为 32 bit, item.axi_addr = item.axi_addr + 32’h4; 若为 64 bit, item.axi_addr = item.axi_addr + 32’h8, 如图 11 所示.

3) AXI_BURST_WRAP 类型, 操作地址具有回卷特点, 以 item.size*item.len 为 wrap boundary, 地址循环操作; 以 item.trans_size = 8 bit 为例, 首先确认回卷地址访问以及起始地址 wrap_axi_addr[log2(item.trans_len)–1:0] = item.axi_addr[log2(item.trans_len)–1:0], 设计数据选通信号 VIF.wstrb=(8’h1 << item.axi_addr[2:0]), 判断回卷是否发送 wrap_axi_addr[log2 (item.trans_len)–1:0] == 1’b1, 若成立, 高位回 0, 再继续回卷操作, 如图 12 所示.

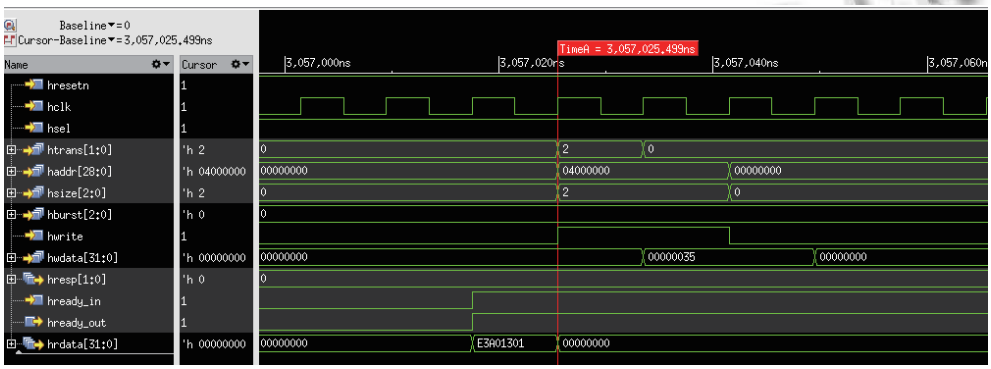


图 6　AHB 写效果图

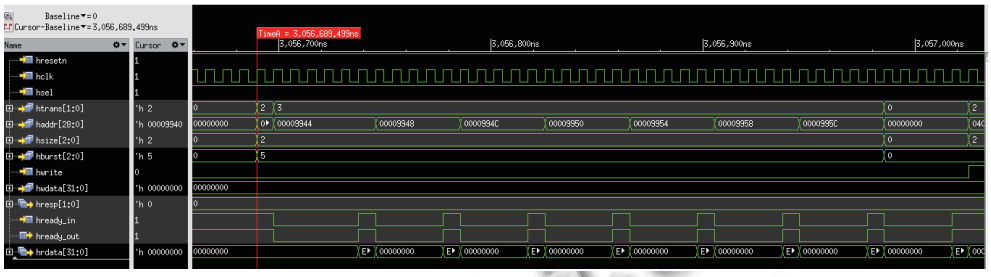


图 7　AHB 读效果图

表 4　AXI sequence item

元素	说明
bit [AXI_WIDTH_ADDR –1:0] axi_addr	操作地址
bit [1023:0] wdata	写操作数据
bit [1023:0] rdata	读操作数据
bit [4:0] trans_len	操作数据长度
bit [1:0]] trans_burst	操作burst类型: AXI_BURST_FIXED\ AXI_BURST_INCR \ AXI_BURST_WARP
bit [1:0] trans_id	操作标识
axi_transfer_direction_e trans_mode	操作传输方向, AXI_WR\AXI_RD
axi_transfer_type_e trans_size	操作数据大小: AXI_BIT8\AXI_BIT16\ AXI_BIT32\ AXI_BIT64

AXI 读驱动效果图, 如图 13 所示. AXI_RD 读操作, 配置读地址通道, VIF.arvalid=1, 配置传输地址, 等待 valid 与 ready handshake, 在时钟上升沿等待 VIF.arready = 1, 若成立, 拉低 VIF.arvalid, 读地址通道配置完成, 开始读数据通道配置操作, 拉高 VIF.rready, 给入传输 id 配置, 通过 VIF.rdata 读回数据, 如图 14 所示.

1.4　验证平台实现

本验证平台采用 UVM 验证方法[10] 学集成了上述前 3 种接口类型. 经典的验证平台结构包含, 如图 15 所示, 需要实现激励功能的 driver 组件, 预判 DUT 输出的 reference model 组件, DUT 输出信息收集的 monitor 组件, 以及给出验证结果的 scoreboard 组件.

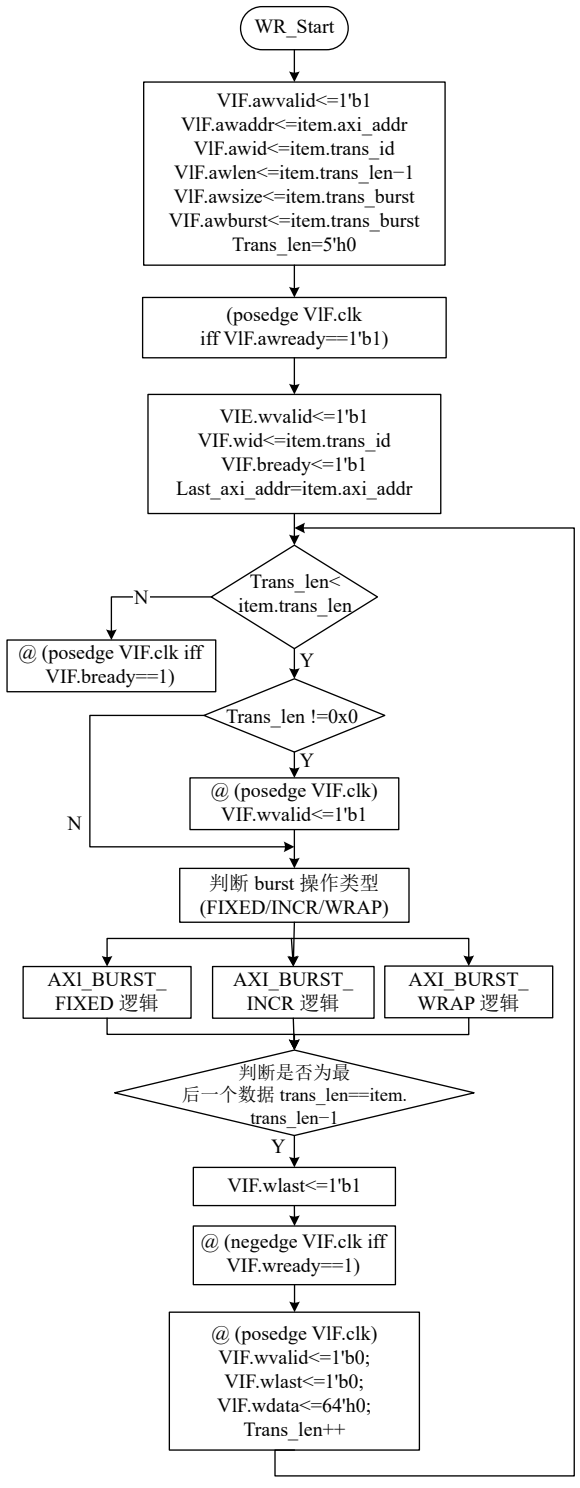


图 8 AXI_WR Driver 流程图

为了满足目标芯片的验证工作, 减少验证的前期准备工作, 通常待测模块需要的接口数量是不定, 例如测试 AXI[11] 转 AHB 接口的桥模块, 需要驱动多路 AXI 总线接口以及观测多路 AHB 总线接口, 基于可扩展性考虑, 验证平台定义一个通用的 Agent 模板.

Agent[12] 是容器化概念, 本平台设计的通用 Agent 模板内部集成了 monitor、driver、sequencer、以及 driver 与 reference model 通信队列传输通道、DUT 与 scoreboard 通信队列传输通道、reference model 与 scoreboard 通信队列传输通道, Agent 定义了各个组件应该存放的位置以及之间如何进行通信[13], 如图 16 所示. 对于不同总线驱动设计, 整体 Agent[14] 框架不变, 新创建文件夹, 改变关键字以及 driver_item_dut 内容既可. 对于相同总线接口的扩展, 例如原 AMBA_UVM[15] 平台内部, 只定义了一路 AXI_MASTER, 验证 PCIE 需要两路 AXI_MASTER, 此时只需要对 uvm/axi_uvc 进行复制, uvm/axi1_uvc, 更改 axi1_uvc 关键字即可, 这样能便于灵活扩展, 实现验证环境集成两路 AXI_MASTER 接口驱动可以同时使用, 满足验证需求.

通信队列传输通道利用 TLM 事务级端口设计, 将组件之间的通信细节 (信息交换细节) 与通信架构的细节分离开来, 事务请求在调用通信队列时发生. 本平台利用通用的 uvm_tlm_analysis_fifo 组件, 明确了数据传输方向, 数据通过 get_ap 获取, 通过 put_ap 传送出去, 实现各个组件之间的通信. 在顶层 tb 例化 uvm_tlm_analysis_fifo 事务, 与通用 agent 模板内预先设计通路配合使用, 实现各个模块之间的通信, 如图 17 所示.

uvm_tlm_analysis_fifo(xx_monitor_sequencer_item) dut2scb_mtr_fifo;

Uvm_tlm_analysis_fifo(xx_monitor_sequencer_item) rm2scb_mtr_fifo;

Uvm_tlm_analysis_fifo(xx_sequence_item) drv2rm_item_fifo;

平台可扩展性实现还需要定制激励发送形式, 这个过程由 sequencer 实现. 可以将 sequencer 看作是个瞄准器, sequence item 就是要发射的物件. 为了便于扩展, 本验证平台建立一个通用的 dut_virtual_ sequencer, 内部对各个接口 sequencer 进行例化. 可以实现通过一个通道发送不同的 sequence item 目的, 如图 16 所示. 当平台接口数目需要扩展时, 可以通过在 dut_virtual_ seuqncer 进行例化.

class dut_virtual_sequencer extends uvm_sequencer;

`uvm_component_utils(dut_virtual_sequencer)

```
dut_apb_sequencer apb_seqr;
dut_ahb_sequencer ahb_seqr;
dut_axi_sequencer axi_seqr;
…
endclass
class dut_tb extends uvm_env;
`uvm_component_utils(dut_tb)
dut_virtual_sequencer vsqr;
…
function void connect_phase(uvm_phase phase);
$display("connect_phase\n");
vsqr.apb_seqr = apb.common_agent.sequencer;
vsqr.ahb_seqr = ahb.common_agent.sequencer;
vsqr.axi_seqr = axi.common_agent.sequencer;
…
Endclass
```

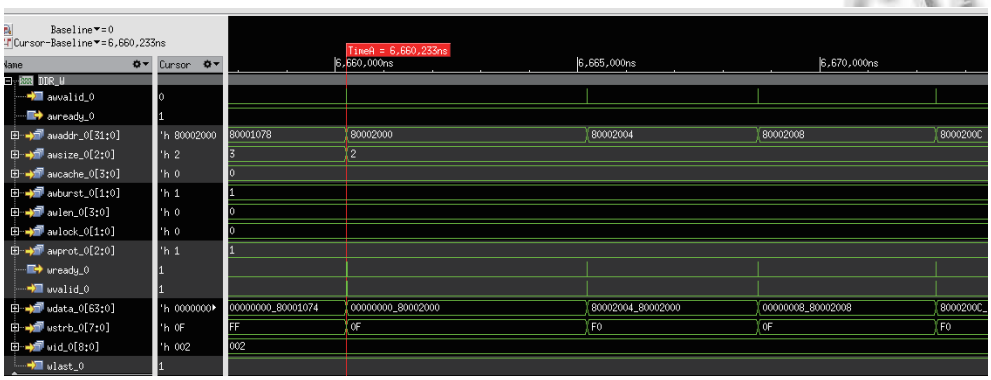


图 9　AXI_WR 效果图

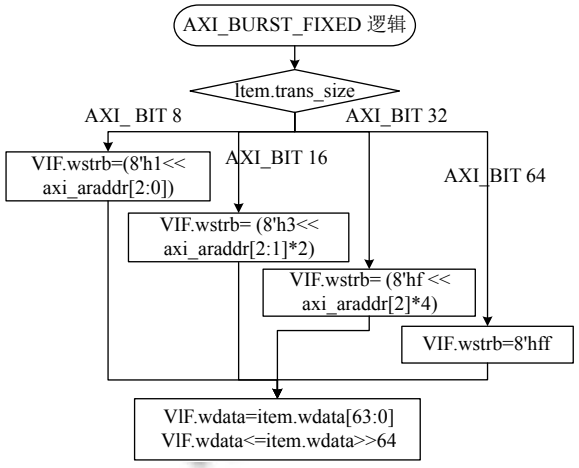


图 10　AXI_BURST_FIXED 流程图

本平台是基于 UVM 验证方法学[16–19] 结合 Agnet 以及 TLM 通信机制进行设计的验证平台, 平台的架构本身具备稳定性, 能够确保验证工作的展开; 而本平台在 driver 以及 monitor 设计中预留了 probe 用于覆盖率数据收集, 能够在验证工作完成之后, 通过 EDA 工具对数据进行分析, 形成代码、功能、断言覆盖率, 用于总结子模块的验证工作是否完备, 若未达到验证需求, 根据覆盖率能够快速准确地找到未覆盖的点, 创造场景激励, 实现全面覆盖, 最终保障验证的完备性.

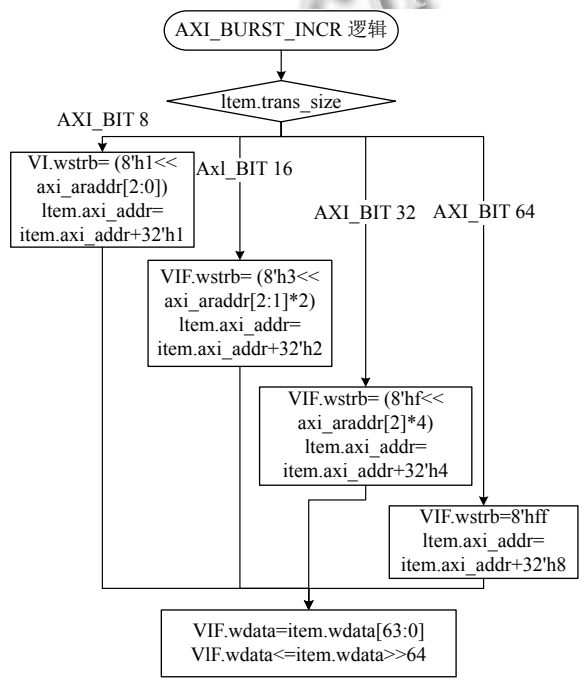


图 11　AXI_BURST_INCR 流程图

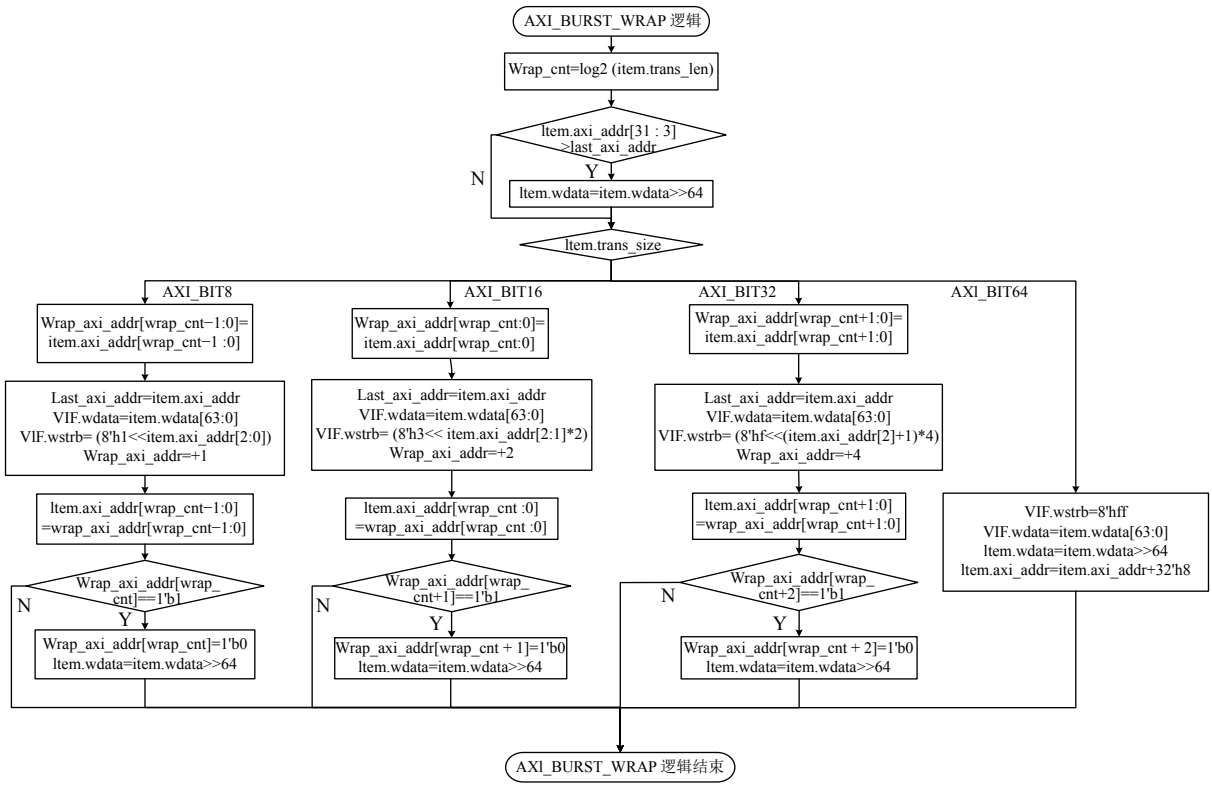


图 12 AXI_BURST_WRAP 流程图

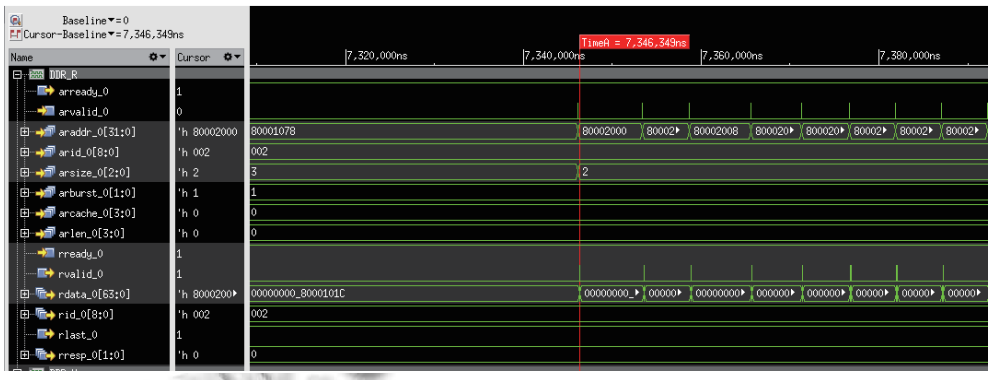


图 13 AXI_RD 效果图

2 AXI-SRAM 控制器验证

目标芯片系统内部按照需求集成了 1 MB 的片上存储设备, 作为 boot ram 使用. 以自研模块 AXI2SRAM 控制器为例进行验证平台使用说明.

对待测子模块接口进行分析, 有两种接口类型, 一路 AXI 接口[20] 用于与系统内部 CPU 或者其它 master 设备进行数据传输, 一路 slave sram 接口用于挂接存储设备. 因此, 根据待测子模块的功能特性将挂接 local_mem 的 axi2sram 控制器整体作为被测试目标, 验证平台搭建如图 18 所示.

本平台具备验证所需的 AXI 接口驱动, 在平台 tb 中例化连接 axi_if 接口驱动和 axi2sram 控制器, 控制器外接 local_mem, 如图 18 所示, 完成验证环境构造. 根据 dut 特点, 编写 reference model, 验证覆盖, axi 8 bit、16 bit、32 bit、64 bit single 操作, axi burst fixed、incr、wrap 操作, burst_len 覆盖 1~16; 基于 UVM 平台特点[21,22], 产生大量的带约束的随机操作, 例如, 以 axi_test_64bit_wrap 测试场景为例, 如图 19 所示为激励构造思路, 激励 axi_bit64_wrap_randomize 能够实现数据随机、传输长度随机、地址随机, 交叉组合; 平台定义的通用 dut_

virtual_sequencer 机制利用 p_sequencer.axi_seqr 能够准确的将测试激励赋值给指定 axi_if 接口, 最终完成随机测试.

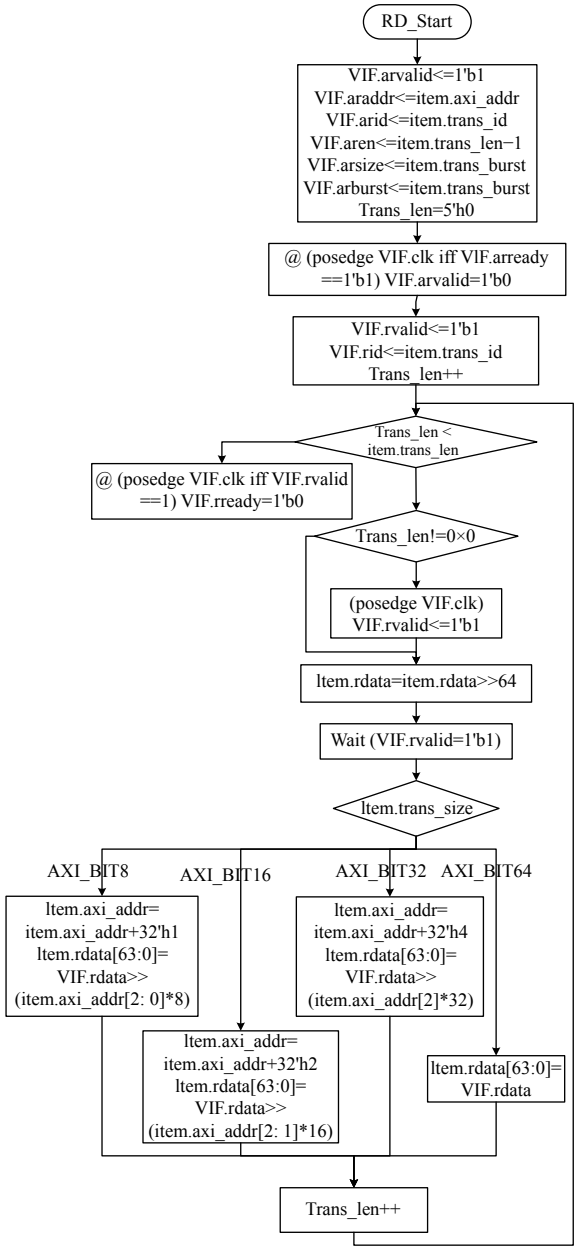


图 14 AXI_RD DRIVER 流程图

实验结果, 如图 20 所示, 本验证平台定义了一个通用的记分板, 用于收集 DUT 接口以及 reference model 分别发送回 Scoreboard 的数据, 并对数据进行分析, 最终打印出比较结果. 基于待测模块结构特点, 平台仅应用 AMBA AXI[23] 驱动, 因此, 记分板对比结果仅 axi 接口具有有效数据, dut_mtr 为待测模传输给 scb 的反馈结果, rm_mtr 为参考模型[24] 传输给 scb 的期望结果, mch_mtr 为成功对比的数目, nmch_mtr 为不符合预期结果的数目; 通过结果可分析, 产生随机次数为 150 028 次, 其中待测目标发送给记分板的数据数目是 150 028 次, 参考模型发送给记分板的数据数目是 150 028 次, 其中, 按照数值产生的先后数据进行对比, 对比失败的数据数目为 0, 因此, 认为本次随机场景测试结果有效. 运行波形如图 21.

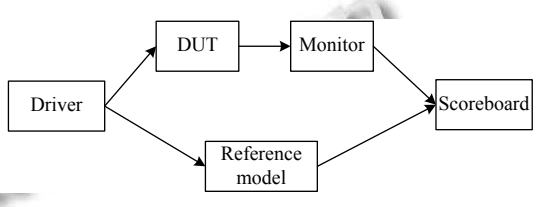


图 15 经典验证平台结构

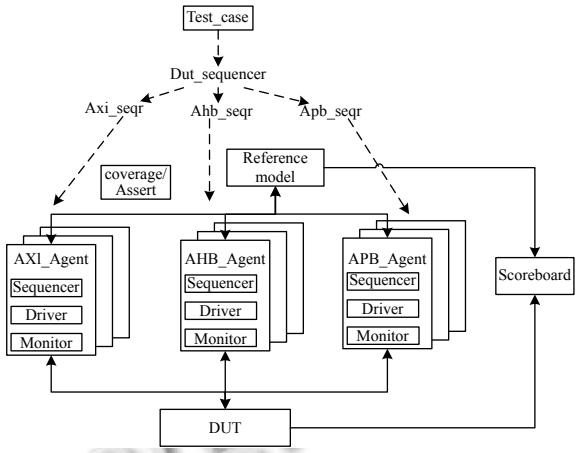


图 16 平台整体结构图

根据验证大纲, 对所列验证点进行递归运行, 在运行脚本中加入 define COV_WORK, 开启验证平台的 probe 功能, 运行脚本 cov.sh 收集覆盖率数据; 运行 merge.sh 最终通过 Cadence 工具得到覆盖率报告, 如图 22 所示, 代码覆盖率 100%, 如图 23 所示, 功能覆盖率 100%; 能够得到结论, AXI2SRAM 控制器通过验证过后硬件代码具备功能完备且可靠, 达到验证目标, 完成验证工作.

```
//cov.sh 脚本
select_coverage -betf -module dut_tb_top…
set_assign_scoring
set_branch_scoring
set_statement_scoring
set_expr_scoring -all
```

```
set_fsm_scoring -hold_transition
set_fsm_arc_scoring
set_glitch_strobe 1 ns
set_toggle_strobe 1 ns
set_covergroup -per_instance_default_one
select_functional
```

```
//merge.sh 脚本
load_test cov_work/scope/*
test_order -b -e > ./coverage_test_order.log
union merge_code
report_summary -instance -besaftd dut_tb_top… > ./cov_summary.log
```

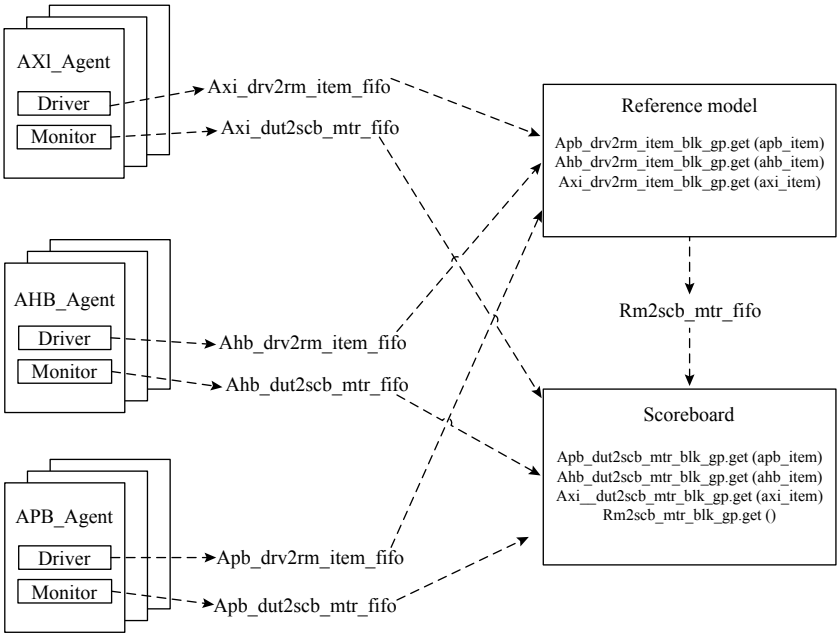


图 17 平台通信队列

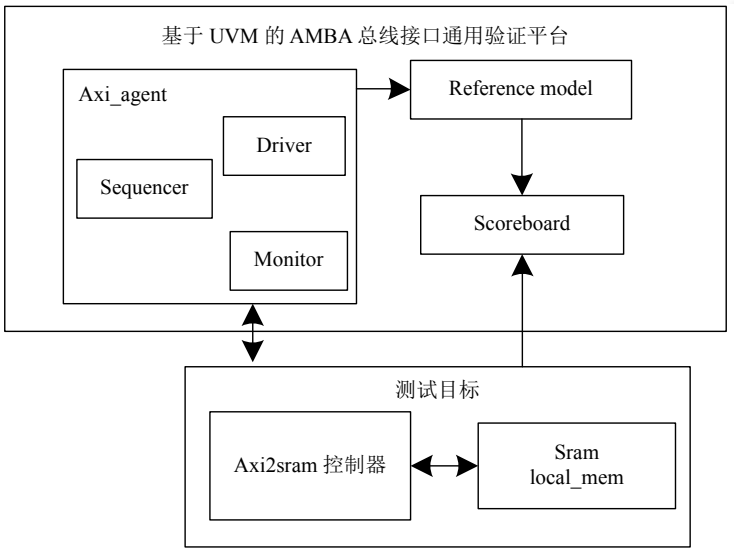


图 18 axi2sram 验证平台结构

3 结论与展望

目前, 本平台能够支持 AMBA 总线接口[25] 的 DUT 测试工作, 在未来, 随着应用以及芯片验证的需要, 基于平台结构的可扩展性, 能够加入其它接口协议, 丰富接口类型, 最终实现接口驱动丰富的面向全芯片模块验证的 UVM 平台.

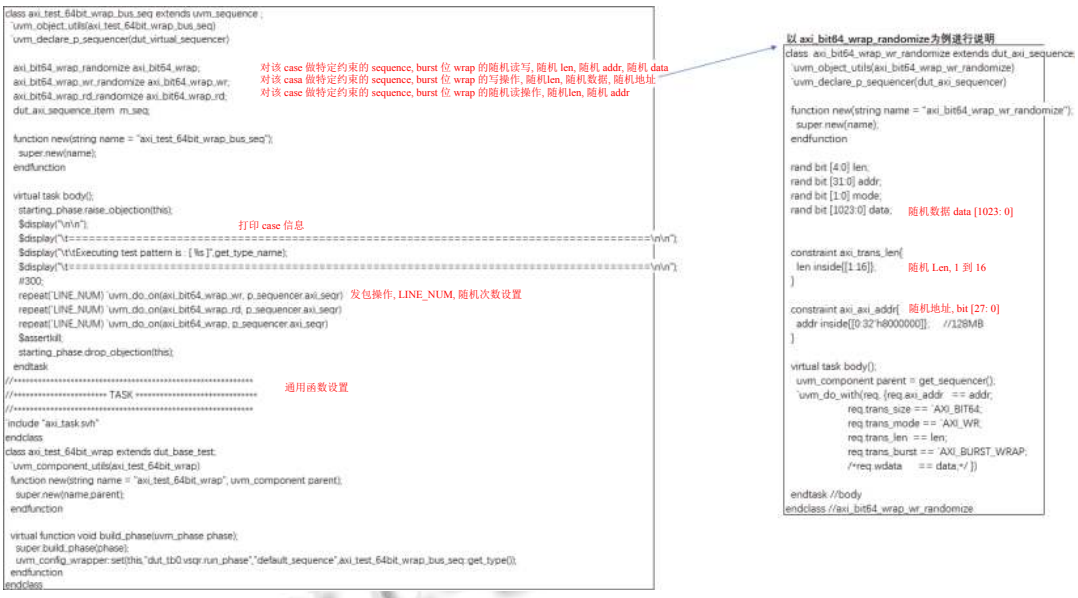
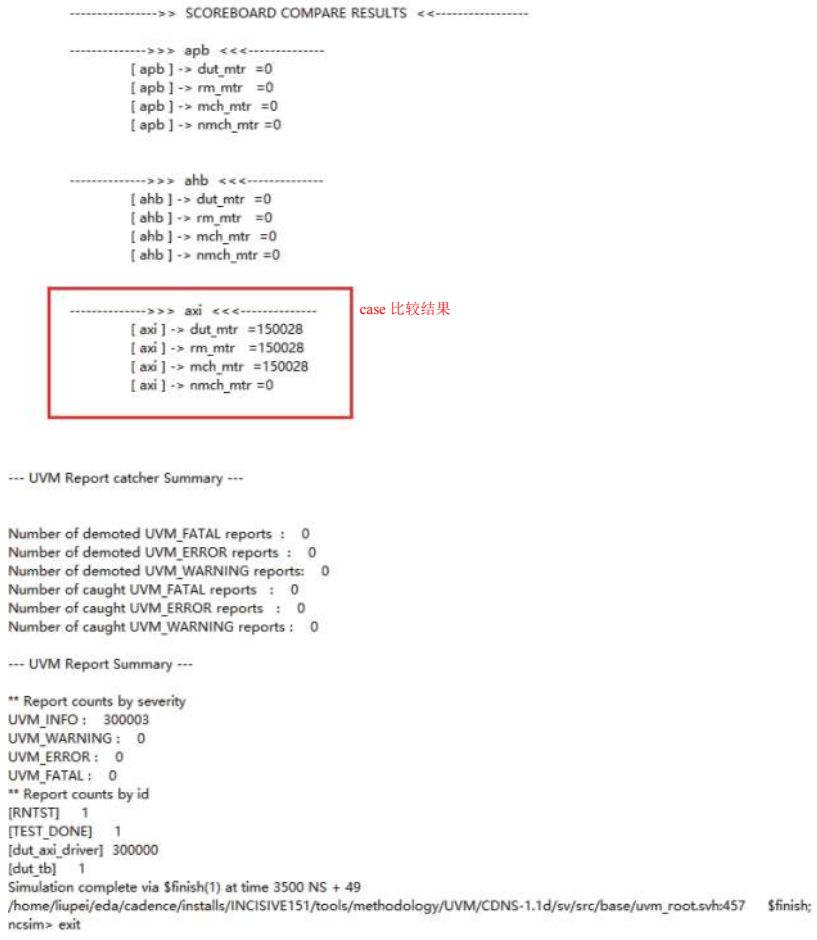


图 19 axi_test_64bit_wrap




```
----------->> SCOREBOARD COMPARE RESULTS <<----------

---------->>> apb <<<----------
        [ apb ] -> dut_mtr  =0
        [ apb ] -> rm_mtr   =0
        [ apb ] -> mch_mtr  =0
        [ apb ] -> nmch_mtr =0


---------->>> ahb <<<----------
        [ ahb ] -> dut_mtr  =0
        [ ahb ] -> rm_mtr   =0
        [ ahb ] -> mch_mtr  =0
        [ ahb ] -> nmch_mtr =0

---------->>> axi <<<----------
        [ axi ] -> dut_mtr  =150028
        [ axi ] -> rm_mtr   =150028
        [ axi ] -> mch_mtr  =150028
        [ axi ] -> nmch_mtr =0


--- 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 :  300003
UVM_WARNING :  0
UVM_ERROR :  0
UVM_FATAL :  0
** Report counts by id
[RNTST]  1
[TEST_DONE]  1
[dut_axi_driver]  300000
[dut_tb]  1
Simulation complete via $finish(1) at time 3500 NS + 49
/home/liupei/eda/cadence/installs/INCISIVE151/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457  $finish;
ncsim> exit
```

图 20 axi_test_64bit_wrap 实验结果

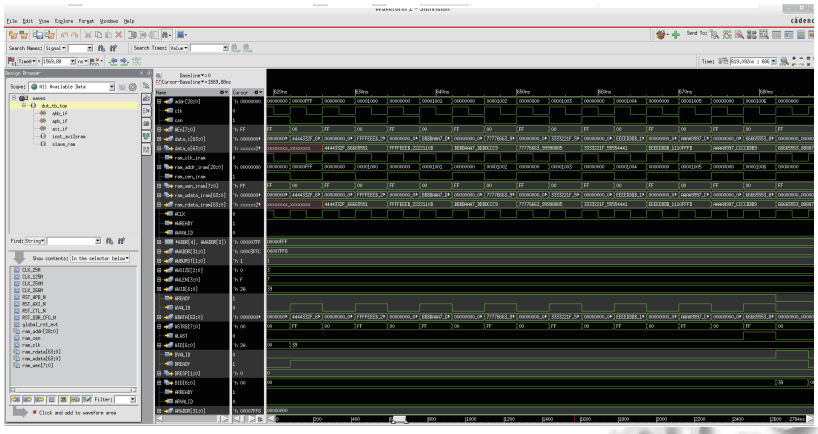


图 21 axi_test_64bit_wrap 运行波形

Coverage summary report, instance-based

Top level summary

Instance name: dut_tb_top
Module/entity name: dut_tb_top

其中, Self 为 TB 平台覆盖率汇总, 数据无关;
inst_axi2sram, 为待测模块, 覆盖率汇总, 重要数据

Total	Block	Branch	Statement	Expression	Toggle	Name
100%	100% (310/310)	100%(223/223)	100%(192/192)	100% (372/372)	100%(1478/1478)	Cumulative
78%	86% (18/21)	No Items	89%(24/27)	50% (4/8)	99% (173/175)	Self

Coverage of immediate sub-instances:

Total	Block	Branch	Statement	Expression	Toggle	Name
No Items	No Items	No Items	No Items	No Items	No Items	apb_if
No Items	No Items	No Items	No Items	No Items	No Items	ahb_if
No Items	No Items	No Items	No Items	No Items	No Items	axi_if
100%	100% (289/289)	100% (223/223)	100% (165/165)	100% (364/364)	100% (1303/1303)	inst_axi2sram
No Items	No Items	No Items	No Items	No Items	No Items	slave_ram

图 22 代码覆盖率

Overall covergroup coverage

Weighted coverage	Uncovered bins	Total bins	Total covergroups
100%	0	184	8

Per covergroup coverage

Weighted coverage	Goal	Weight	Uncovered Bins	Total Bins	Name	Comment
100%	100%	1	0	19	uvm_pkg.i_general_write_BIT8_event	
100%	100%	1	0	19	uvm_pkg.i_general_write_BIT16_event	
100%	100%	1	0	19	uvm_pkg.i_general_write_BIT32_event	
100%	100%	1	0	19	uvm_pkg.i_general_write_BIT64_event	
100%	100%	1	0	27	uvm_pkg.o_general_read_BIT8_event	
100%	100%	1	0	27	uvm_pkg.o_general_read_BIT16_event	
100%	100%	1	0	27	uvm_pkg.o_general_read_BIT32_event	
100%	100%	1	0	27	uvm_pkg.o_general_read_BIT64_event	

图 23 功能覆盖率

参考文献

1 张强. UVM 实战-卷 I. 北京: 机械工业出版社, 2014.
2 刘达, 倪伟, 徐春琳. 基于 UVM 的 AXI 总线验证 IP 设计. 微电子学, 2019, 49(5): 680–685.
3 ARM. AMBA APB protocol specification. 2010.
4 ARM. AMBA 3 AHB-Lite Protocol Specification. 2006.
5 ARM. AMBA AXI protocol specification. 2010.
6 ARM. AMBA ATB protocol specification. 2006.
7 魏吉泰. 基于 AMBA 总线的通用异步接口的设计与验证 [硕士学位论文]. 成都: 电子科技大学, 2019.
8 王一楠. 基于 AMBA2.0 的 AHB Matrix 总线架构设计 [硕士学位论文]. 西安: 西安理工大学, 2018.
9 王一楠, 林涛, 余宁梅. 基于 AMBA 的 AHB 总线矩阵设计. 微电子学与计算机, 2019, 36(2): 73–77.
10 王心弋. 基于 UVM 的自适应验证平台设计与实现 [硕士学位论文]. 成都: 电子科技大学, 2020.
11 贾玲玲. 基于 AMBA 总线的 SMBus 总线控制器的设计与实现 [硕士学位论文]. 成都: 电子科技大学, 2020.
12 Bromley J. If systemverilog is so good, why do we need the UVM? Sharing responsibilities between libraries and the core language. Proceedings of the 2013 Forum on Specification and Design Languages. Paris, France. 2013. 1–7.
13 刘斌. 芯片验证漫游指南. 北京: 电子工业出版社, 2018.
14 克里斯·斯皮尔. SystemVerilog 验证. 张春, 麦宋平, 赵益新, 译. 北京: 科学出版社, 2009.
15 李晨阳, 宋澍申, 王涛, 等. 一种基于 UVM 的高层次化验证平台设计. 微电子学与计算机, 2019, 36(6): 79–83.
16 Huang X, He X, He ZR, *et al*. Using UVM testbench to generate the analog stimuli. Proceedings of 2019 2nd International Conference on Informatics, Control and Automation. Hangzhou, China. 2019. 249–253.
17 Marconi S, Conti E, Christiansen J, *et al*. A UVM simulation environment for the study, optimization and verification of HL-LHC digital pixel readout chips. Journal of Instrumentation, 2018, 13(5): P05018. [doi: 10.1088/1748-0221/13/05/P05018]
18 阎芳, 李哲玮, 田毅, 等. 基于 UVM 的多通路航空总线收发器 IP 验证. 电光与控制, 2018, 25(1): 70–73.
19 Fiergolski A. Simulation environment based on the universal verification methodology. Journal of Instrumentation, 2017, 12: C01001. [doi: 10.1088/1748-0221/12/01/C01001]
20 Akhila P, Prathap Kumar MKC. Design of AMBA 3.0 (AXI) bus based system on chip communication protocol. International Journal of Science and Research, 2013, 2(3): 57–60.
21 倪伟, 王笑天. 基于 UVM 的功能覆盖率驱动 SDIO IP 验证. 微电子学, 2017, 47(3): 392–395, 411.
22 Kiran A, Thrimurthulu V. Verification of AMBA AHB2APB bridge using Universal Verification Methodology (UVM). International Journal in IT and Engineering, 2016, 4(12): 1–10.
23 李哲玮. 基于 AMBA 协议的片上数据总线适航性设计 [硕士学位论文]. 天津: 中国民航大学, 2018.
24 钱一文, 景为平, 蒋斌. 基于 UVM 的 CPU 卡芯片验证平台. 微电子学与计算机, 2016, 33(6): 37–40.
25 胡景华. 基于 AXI 总线的 SoC 架构设计与分析 [硕士学位论文]. 上海: 上海交通大学, 2013.