Journal of Computer Applications
计算机应用, 2020, 40(5): 1369 - 1373
ISSN 1001-9081
CODEN JYIIDU
2020-05-10
http://www.joca.cn

文章编号:1001-9081(2020)05-1369-05
DOI:10.11772/j.issn.1001-9081.2019091674

Lite寄存器模型的设计与实现

潘国腾*,欧国东,晁张虎,李梦君
(国防科技大学 计算机学院,长沙 410073)
(*通信作者电子邮箱gtpan@nudt.edu.cn)

摘　要:针对集成电路规模扩大、片内寄存器数量激增,导致验证难度加大的问题,提出一种轻量级寄存器模型。首先,设计精简的底层结构,配合参数化设置减少寄存器模型在运行时的内存消耗;然后,分析模块级、系统级等不同层次的寄存器验证需求,使用SystemVerilog语言实现验证所需的各项功能;最后,开发内建测试用例和寄存器模型自动生成工具,缩短寄存器模型所处验证环境的建立时间。实验结果表明,在运行时内存消耗方面,该寄存器模型为通用验证方法学(UVM)寄存器模型的21.65%;在功能方面,可应用于传统的UVM验证环境和非UVM验证环境,对25类寄存器的读写属性、复位值、后门访问路径等功能进行检查。该轻量级寄存器模型在工程实践中拥有良好的通用性和灵活性,满足寄存器验证需求,能有效提高寄存器验证的效率。

关键词:寄存器模型;验证;Python;测试用例;性能分析
中图分类号:TP368　　**文献标志码**:A

Design and implementation of Lite register model

PAN Guoteng*, OU Guodong, CHAO Zhanghu, LI Mengjun
(*College of Computer Science and Technology, National University of Defense Technology, Changsha Hunan* 410073, *China*)

Abstract: Aiming at the problem that the scale of integrated circuits and the number of on-chip registers are increasing, which makes the verification more difficult, a lightweight register model was proposed. Firstly, a concise underlying structure was designed, and parameterized settings were combined to reduce the memory consumption of the register model at runtime. Then the register verification requirements at different levels such as module level and system level were analyzed, and SystemVerilog language was used to implement various functions required for verification. Finally, the built-in test cases and register model automatic generation tools were developed to reduce the setup time of the verification environment in which the register model was located. The experimental results show that the proposed register model is 21.65% of the Universal Verification Methodology (UVM) register model in term of memory consumption at runtime; in term of function, the proposed register model can be applied to traditional UVM verification environments and non-UVM verification environments, and the functions such as read-write property, reset value and backdoor access path of 25 types of registers are checked. This lightweight register model has good universality and flexibility in engineering practice, meets the needs of register verification, and can effectively improve the efficiency of register verification.

Key words: register model; verification; Python; test case; performance analysis

0　引言

在摩尔定律驱动下,单颗芯片的逻辑门数增长迅速,更多的功能模块集成到片上,同时引入更多寄存器,通用片上系统(System-on-Chip, SoC)芯片面临的情况尤为突出[1]。SoC芯片拥有数量庞大的各类寄存器,寄存器验证已成为SoC芯片研发过程中的重要验证项。根据SoC芯片分层验证理念[2-3],模块级的设计规模小、功能简单、激励构造相对容易,但模块数量较多[4];子系统级的设计规模适中、功能相对独立、接口清晰,而验证的完备性却是最大的挑战[5];系统级的设计规模庞大,功能复杂,对仿真资源的需求比较大[6]。无论在模块级、子系统级或系统级验证中,寄存器都是验证的重心之一。如何更高效、更完备地完成芯片的寄存器验证,成为芯片验证工程师必须解决的问题。

目前,业界在芯片的寄存器验证中更多采用C语言或者汇编语言开发测试激励,芯片寄存器初始化配置过程经过取值、译码、执行,然后逐层传输到寄存器所在模块。单个寄存器的配置过程通常需要几百个时钟周期,芯片的初始化过程通常需要配置成千上万个寄存器,配合软件模拟缓慢的仿真速度,使得整个寄存器配置过程需要很长时间。裸机环境下的寄存器通路测试遇到的问题更加严重。寄存器通路验证通常需要覆盖整个芯片的所有寄存器,其数量是初始过程需要配置寄存器数量的数倍,急需一种更加高效的验证方法。

通用验证方法学(Universal Verification Methodology, UVM)的高可重用性、带约束的随机激励生成、完善的UVM库以及覆盖率驱动验证(Coverage Driven Verification, CDV)理念非常贴合模块级和子系统级的验证需求[7-9],逐渐成为业界的主流。UVM里的RAL(Register Abstraction Layer)方案[10]是

收稿日期:2019-10-08;**修回日期**:2019-12-28;**录用日期**:2019-12-30。
基金项目:核高基国家科技重大专项(2017ZX01028-103-002);国家自然科学基金面上项目(61672525)。
作者简介:潘国腾(1977—),男,山东曹县人,副研究员,博士,CCF会员,主要研究方向:微处理器设计与验证; 欧国东(1977—),男,湖南武冈人,助理研究员,博士,主要研究方向:微处理器设计与验证; 晁张虎(1989—),男,河南濮阳人,工程师,硕士,主要研究方向:微处理器设计与验证; 李梦君(1976—),男,湖北云梦人,副教授,博士,主要研究方向:微处理器设计与验证。

一套完整的寄存器验证解决方案，在模块级和子系统级的UVM验证环境中发挥着重要作用，但在系统级验证中却极少被使用，原因是UVM库代码繁复，仿真运行时消耗内存较大，对验证的仿真资源有非常高的要求；同时，UVM的RAL方案要求寄存器模型和RTL代码均符合电子知识产权（IP-XACT）标准[11-12]，UVM内建的寄存器测试用例不对“只读（Read Only，RO）”“只写（Write Only，WO）”等属性的寄存器进行Access检查，导致UVM寄存器模型在系统级验证中难以得到推广。

针对以上问题，本文提出一种Lite寄存器模型，实现更小的运行时内存消耗，可以更广泛地应用于各种验证场景。Lite寄存器模型保留UVM寄存器模型前/后门访问等功能，采用前/后门访问结合的方法进行寄存器通路验证，不仅可以提高仿真验证的速度，同时实现对地址映射的验证。Lite寄存器模型配合内建测试用例，实现对所有访问属性（包括“RO”和“WO”属性）寄存器的Access检查，使功能验证更加高效完备。该模型可以在UVM或非UVM的模块级、子系统级或系统级仿真验证环境中使用，并针对系统级寄存器通路验证进行加强；同时，开发寄存器模型自动生成工具，提供Lite寄存器模型内建测试用例，形成完整的寄存器模型解决方案。

1 Lite寄存器模型

Lite寄存器模型使用SystemVerilog语言实现，支持对多种读写属性的寄存器进行建模，提供各类方法满足不同的寄存器访问需求。

1.1 Lite寄存器模型结构

Lite寄存器模型分3层：Field层、Register层、Regmodel层；外加用于寄存器前门访问的Adaptor，构成寄存器模型的主体。

1）Field层：负责描述寄存器内各域段的基本属性（期望值、映射值、复位值、读写属性、位宽等），是寄存器模型的最小单元。

2）Register层：负责处理寄存器各Fields遍历/拼接、寄存器读写操作（前门/后门），记录寄存器的后门访问路径、地址等信息，是寄存器模型的基本单元。

3）Regmodel层：寄存器模型的顶层，配置模型包含的各寄存器基地址、顶层后门访问路径等属性，设置寄存器模型的Adaptor，建立集成管理方法，为内建测试激励提供接口。

4）Adaptor：处理寄存器模型与待测设计（Design Under Test，DUT）之间的接口协议适配，使用SystemVerilog的Callback机制实现灵活重构。

该寄存器模型的主要功能如下：

1）支持寄存器前门访问和后门访问：后门访问通过SystemVerilog提供的直接编程接口（Direct Programming Interface，DPI）[13]接口实现；前门访问使用接口适配器Adaptor完成事务级与信号级的相互转换，实现寄存器模型与设计接口的解耦，接口协议变化后只需重新实现Adaptor，已实现的寄存器模型无需改动，增强寄存模型的可重用性。

2）每个寄存器模型包含3个值：复位值、期望值、映射值。复位过程中，有复位值的寄存器，其模型内的期望值被初始化为复位值；期望值是寄存器模型对设计中寄存器值的预期值；映射值是寄存器模型发起同步操作时，从设计中得到的寄存器的真实值。期望值在前、后门写访问成功后更新，映射值在前、后门读访问成功后更新。读访问成功后会触发模型内部期望值和映射值的比对，检查寄存器值的变化是否正确。

3）支持READ操作、WRITE操作、读等待（Wait READ，WREAD）操作和强制写（Force WRITE，FWRITE）操作四种寄存器访问操作：READ和WRITE操作分别有前门访问和后门访问两种模式；WREAD操作能够检查“WO”类属性寄存器的前门写操作是否正确；FWRITE操作能够检查“RO”类属性寄存器的前门读访问是否正确。WREAD操作和FWRITE操作仅支持后门访问。

4）支持对25种读写属性的寄存器建模：只读、读写、读清零、读置位、可写读复位、可写读清零、写清零、写置位、写置位读清零、写清零读置位、写1清零、写1置位、写1反转、写0清零、写0置位、写0反转、写1置位读清零、写1清零读置位、写0置位读清零、写0清零读置位、只写、只写清零、只写置位、一次写、一次只写。

5）该模型不依赖UVM库（DPI函数库除外），但可应用于UVM环境和非UVM环境中。

6）内建寄存器通路验证测试用例，包括寄存器复位值检查、寄存器读写功能检查和寄存器后门访问路径检查。

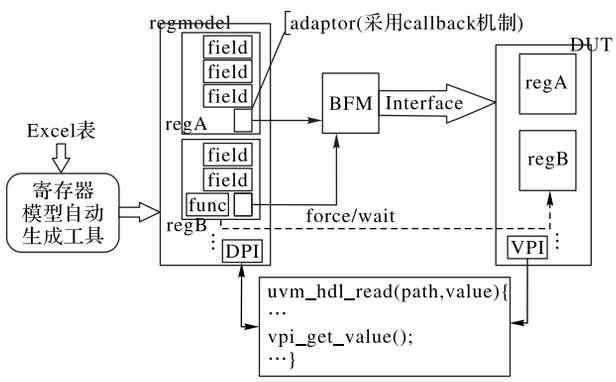


图1 Lite寄存器模型的总体结构

Fig. 1 Overall structure of Lite register model

1.2 Lite寄存器模型库单元

UVM寄存器模型底层库历经OVM（Open Verification Methodology）、VMM（Verification Methodology Manual）、UVM的不断演进，历史包袱沉重，源码结构复杂，多有冗余。业界各主要电子设计自动化（Electronics Design Automation，EDA）工具编译出的Database格式各异，为便于调试均往UVM源码库中添加大量代码，使得UVM源码库更加繁杂。Lite寄存器模型重新设计底层源码库，结构清晰简约，有效减少对内存的损耗。

Lite寄存器模型库由4部分内容组成：

1）宏定义模块。集中配置模型内部的数据结构，针对具体目标调整，可有效减小运行时内存消耗。

2）接口协议适配器。处理模型与DUT之间事务级与信号级的相互转换，同时兼具接口协议BFM功能；

3）寄存器模型基类。描述寄存器的各类属性，建立相关的函数和方法。

4）内建测试激励。用于寄存器通路验证。

Lite寄存器模型库单元完整Package包含如下文件：

```
package lite_regmodel;
  `include "lite_reg_macros.svh"
  `include "dpi/uvm_hdl.svh"
  `include "dpi/uvm_svcmd_dpi.svh"
  `include "dpi/uvm_regex.svh"
  `include "lite_adaptor.sv"
  `include "lite_field.sv"
  `include "lite_reg.sv"
  `include "lite_regmodel.sv"
  `include "lite_reset_check_tc.sv"
  `include "lite_hdl_path_check_tc.sv"
  `include "lite_access_check_tc.sv"
endpackage
```

库单元中主要类模块的层次结构如图2所示,其中Lite_adaptor为virtual类,Lite_regmodel和Lite_reg中均有其句柄,用户创建Lite_adaptor的子类后注册到Lite_reg中(SystemVerilog的callback机制),实现每个Lite_reg与adaptor的连接,打开寄存器的前门访问通道。

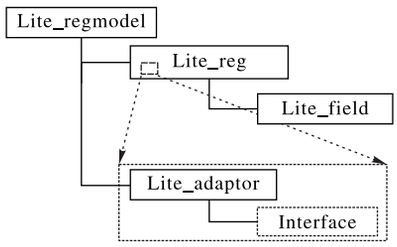


图2　寄存器模型库单元类模块层次结构

Fig. 2　Hierarchical structure of register model library cell class module

Lite_regmodel类中new函数和build函数为virtual函数,由用户在子类中实现具体内容。add_reg函数将模型中所有的Lite_reg句柄存入寄存器池内,以便进行遍历操作。

Lite_reg类中new函数和configure函数为virtual函数,其余函数封装在类内。hdl_check函数、access_check函数和reset_check函数负责寄存器通路检查,fwrite函数、wread函数、read函数和write函数处理前/后门读写访问,add_field函数将所有的field存入fields池内。

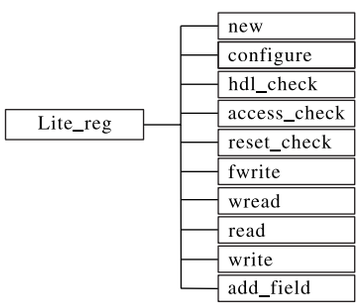


图3　Lite_reg类构成

Fig. 3　Composition of Lite_reg

Lite_adaptor类是virtual类,bus2reg任务和reg2bus任务根据具体接口协议在其子类中实现,wread任务解决了"RO"类寄存器的通路检查问题。

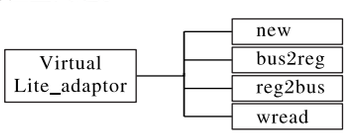


图4　Lite_adaptor类构成

Fig. 4　Composition of Lite_adaptor

Lite_field类中new函数和configure函数为virtual函数,其余函数封装在类内。被封装的函数分别负责寄存器field的mirror值和desire值的维护、为通路验证提供底层支持、前/后门读写访问等。force_wr函数和release_wr函数解决了"WO"类寄存器的通路检查问题。

1.3　内建测试用例

Lite寄存器模型内建寄存器后门访问路径检查、寄存器复位值检查和寄存器读写功能检查3个测试用例。内建测试用例使用foreach遍历Lite_regmodel中的寄存器池进行寄存器复位值检查和后门访问路径检查,根据读写属性采用不同的方式进行寄存器读写功能检查。

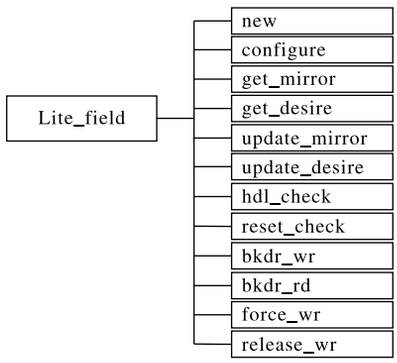


图5　Lite_field类构成

Fig. 5　Composition of Lite_field

寄存器读写功能检查测试用例对每个寄存器进行两种模式的检查:前门写+后门读检查、后门写+前门读检查。前门写+后门读检查将寄存器模型中的映射值按位取反后从前门写入DUT,同时更新寄存器模型中的期望值。随后通过后门读配合模型中的期望值,检查前门写访问是否正确。后门写+前门读检查采用相反的操作,检查前门读访问是否正确。

UVM源码库内建的寄存器读写功能检查测试用例不对寄存器模型中具有"RO"属性和"WO"属性的寄存器进行读写功能检查:"RO"属性寄存器值的变化仅由设计内部逻辑决定,验证环境对其不可预期;"WO"属性寄存器在进行前门写后,写入的值可能在很短的时间被设计内部逻辑清除,后门读操作的时机无法控制,但是在子系统级、系统级对这两类寄存器进行读写通路验证同样很有必要。

为解决这两个难题,Lite寄存器模型内建读写功能测试用例,处理"RO"属性寄存器时,调用Lite寄存器模型的fwrite函数,采用force操作完成后门写,前门读访问完成后再release;处理"WO"属性寄存器时,调用Lite寄存器模型的wread方法,在前门写访问完成后逐拍进行后门读访问,直到读出期望的值。在寄存器模型中设置寄存器访问最大延迟周期,当后门访问持续拍数大于该最大周期数后报访问超时错。

2　自动生成脚本

芯片中的寄存器数量庞大,通过手工编码实现对所有寄存器的建模耗时长且容易出错。Lite寄存器模型自动生成工具,可以极大地减少编码的工作量且避免人为错误的引入。

2.1　寄存器描述文档

寄存器描述文档作为《详细设计文档》的重要组成部分,在整个芯片研发周期中几乎贯穿始终。

图6给出工程实践中寄存器描述文档的规范格式,表头包括项目名称、寄存器所在模块名、寄存器通路位宽、寄存器最大访问周期数和文档版信息。正文描述每个寄存器的详细属性。

表头中的"protocal"决定adaptor的实现,"hdl_path"与寄存器中每个field的hdl_path拼接构成寄存器的后门访问路径。每个寄存器中各field可能拥有不同的读写属性,前门访问使用寄存器地址,将同一个寄存器内各fields作为整体进行读写,后门访问以field为单位调用DPI接口实现。

2.2　寄存器模型生成脚本

本文使用Python语言[14]编写寄存器模型生成脚本,从寄存器描述文档中提取信息,对寄存器进行分层建模。脚本采用模块化开发方式,主要包括以下模块:Excel信息扫描模块、Fields生成模块、Register生成模块、Regmodel生成模块。模块化开发增加脚本的可扩展性和可重用性,例如寄存器文档格式变化后

只需要修改Excel信息扫描模块,其他模块可完全重用。

project	test				
module	test				
version	1.0				
max_access_delay	20				
protocal	apb				
hdl_path	top_tb.dut.rg_ctl				
regname	offset	fields	bits	reset	access
ctl	0x00	lock	31	0x1	RW
		mode_sel	30:0	0xa	RW
status	0x04	err_type	31:16	0x0	RO
		max_err_num	15:0	0xffff	RW
clr	0x08	clr	31:0	0x33333333	WO

图6 寄存器描述文档规范

Fig. 6 Register description document specification

不同寄存器建立的模型之间存在大量的“共性”,比如都包含new函数、configure函数、read/write task等;同时不同寄存器之间也存在着明显的“差异”,如图6中描述的地址、读写属性、复位值、寄存器名等。因此,脚本使用gen_reg函数生成寄存器模型的“共性”部分,同时提取图6中的“差异”信息生成“个性”部分,建立完整的寄存器模型。

最后,脚本根据寄存器文档表头信息生成寄存器模型的顶层文件,将已完成建模的所有寄存器逐一在顶层文件中例化。

3 实验结果

首先选取工程实践中具有代表意义的寄存器,使用硬件描述语言Verilog[15]做寄存器传输级(Register Transaction Level, RTL)编码,作为实验的DUT。然后运行寄存器模型生成工具对DUT中实现的寄存器进行Lite寄存器模型建模。最后采用业界主流模拟仿真工具进行仿真,评估寄存器模型的功能和性能。

3.1 自动生成寄存器模型

执行寄存器模型生成脚本,生成test_regmodel. sv文件,test_regmodel. sv中单个寄存器模型:

```
class clr extends lite_reg;
  rand lite_field clr;
  function void build();
    clr=new("clr");
    clr.configure(32,0,"WO",0,'h33333333,1,0,0);
    add_field(clr);
  endfunction
endclass
```

寄存器模型顶层模块:

```
class test_regmodel extends lite_regmodel;
  function new (string name= "test_regmodel", virtual interface
  apb_if avif);
    adaptor=new("adaptor",20);
    this.avif=avif;
  endfunction
  rand clr clr;
  …
  function void build();
    adaptor.avif=avif; ctl=new("ctl");
    ctl.build();
    ctl.configure("top_tb.dut.rg_ctl",'h00);
    ctl.adaptor.push_back(adaptor); add_reg(ctl);
    …
  endfunction
  …
endclass
```

3.2 功能模拟

Lite寄存器模型设计实现底层库单元,支持前门访问和后门访问,写访问时更新寄存器模型的值,读访问时进行返回值的正确性检查。该模型兼容UVM模拟仿真环境和非UVM模拟仿真环境。

为验证Lite寄存器模型读写访问功能是否正确,分别在非UVM模拟环境顶层top_tb的Initial块和UVM模拟环境的sequence中添加基于Lite寄存器模型的内建测试激励和自建测试激励,模拟实际工程项目验证项对DUT进行寄存器验证。实验中分别使用Lite寄存器模型的前门/后门访问方式结合READ、WRITE、FWRITE和WREAD操作方法,对DUT中寄存器进行访问。内建测试激励包括寄存器后门路径检查、寄存器复位值检查和寄存器读写功能检查,内建测试用例;自建测试激励覆盖Lite寄存器模型的所有寄存器访问方式。仿真结果如图7、8所示。

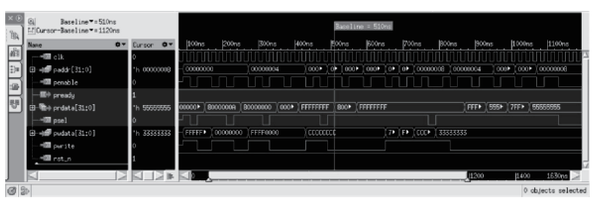


图7 仿真运行波形

Fig. 7 Waveform of simulation

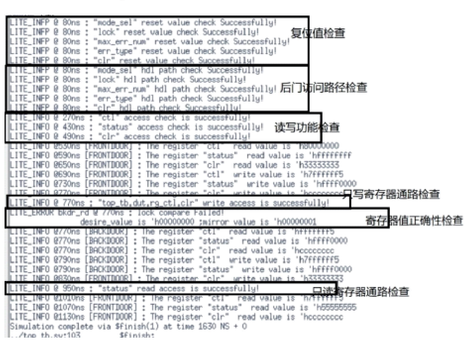


图8 仿真运行终端打印结果

Fig. 8 Output result of simulation terminal

```
//create regmodel
regmodel=new("regmodel",apb_if);
regmodel.build();
//create reset value check testcase
reset_check=new("reset_check");
reset_check.rgm=regmodel;
reset_check.start_test();
//create hdl path check testcase
hdl_check=new("hdl_check");
hdl_check.rgm=regmodel;
hdl_check.start_test();
//create access check testcase
access_check=new("access_check");
access_check.rgm=regmodel;
access_check.start_test();
```

为验证Lite寄存器模型对错误的检测功能是否正常,在自建测试激励中通过force操作注入错误。仿真过程中该模型准确检查出注入的错误并打印错误报告。

自建测试用例:

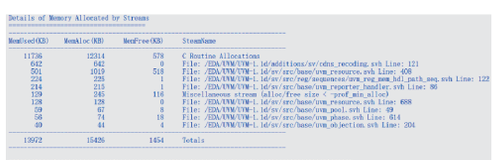
```
//add testcase
regmodel.ctl.read(FRONTDOOR,value);
regmodel.status.read(FRONTDOOR,value);
regmodel.clr.read(FRONTDOOR,value);
```

```
regmodel.ctl.write(FRONTDOOR,'h7fffffff5);
regmodel.status.write(FRONTDOOR,'hffff0000);
regmodel.clr.write(FRONTDOOR,'hcccccccc);
regmodel.clr.wread('hcccccccc);
//inject error force dut.rg_ctl.lock=1'b1;
regmodel.ctl.read(BACKDOOR,value);
regmodel.status.read(BACKDOOR,value);
regmodel.clr.read(BACKDOOR,value);
release dut.rg_ctl.lock;
@(posedge clk);
regmodel.ctl.write(BACKDOOR,'h7fffffff5);
regmodel.status.write(BACKDOOR,'hffff0000);
regmodel.clr.write(FRONTDOOR,'h33333333);
regmodel.status.fwrite('h55555555);
regmodel.ctl.read(FRONTDOOR,value);
regmodel.status.read(FRONTDOOR,value);
regmodel.clr.read(FRONTDOOR,value);
```

仿真结果证明,Lite寄存器模型通过前门/后门对寄存器发起READ、WRITE、FWRITE和WREAD操作,能够进行数据正确性比对,实现预期功能;Lite寄存器模型的内建测试用例能够对包括“RO”属性和“WO”属性等特殊寄存器在内的所有寄存器进行寄存器通路验证,解决了UVM寄存器模型在此方面的问题。

3.3　性能分析

为分析Lite寄存器模型仿真时对内存的消耗,实验中分别使用Lite寄存器模型和UVM寄存器模型对同一个DUT做寄存器建模,如图9所示。在仿真过程中加入-memxprof选项(仿真工具中收集仿真性能的选项)收集内存消耗,得到结果如图9所示。



Details of Memory Allocated by Streams

MemUsed(KB)	MemAloc(KB)	MemFree(KB)	SteamName
11736	12314	578	C Routine Allocations
642	642	0	File: /EDA/VMM/VCM-L.1d/additions/sv/cdns_recoding.svh Line: 121
501	1019	518	File: /EDA/VMM/VCM-L.1d/sv/src/base/uvm_resource.svh Line: 698
224	225	1	File: /EDA/VMM/VCM-L.1d/sv/src/reg/sequences/uvm_reg_mem_hdl_path_seq.svh Line: 122
214	215	1	File: /EDA/VMM/VCM-L.1d/sv/src/base/uvm_reporter_handler.svh Line: 86
129	245	116	Miscellaneous streams (alloc/free size < -prof_min_alloc)
128	128	0	File: /EDA/VMM/VCM-L.1d/sv/src/base/uvm_resource.svh Line: 688
59	67	8	File: /EDA/VMM/VCM-L.1d/sv/src/base/uvm_pool.svh Line: 69
56	74	18	File: /EDA/VMM/VCM-L.1d/sv/src/base/uvm_phase.svh Line: 614
49	44	4	File: /EDA/VMM/VCM-L.1d/sv/src/base/uvm_objection.svh Line: 204
13972	15426	1454	Totals

(a) UVM寄存器

Details of Memory Allocated by Streams

MemUsed(KB)	MemAloc(KB)	MemFree(KB)	SteamName
2786	2852	66	C routine Allocations
225	225	0	File: ~/regmodel/lite_regmodel/lite_field.sv Line: 102
9	14	5	File: ~/regmodel/lite_regmodel/lite_field.sv Line: 130
5	5	0	File: ~/regmodel/apb/apb_if.sv Line: 66
1	1	0	File: ~/regmodel/lite_regmodel/lite_field.sv Line: 90
3026	3105	79	Totals

(b) Lite寄存器

图9　模型内存消耗分析报告

Fig. 9　Analysis report of model memory consumption

通过分析报告可知,得益于精简的底层库实现及配置化处理,Lite寄存器模型的运行时内存消耗是UVM寄存器模型运行时内存消耗的21.65%,极大地减小了仿真所需的内存资源消耗,在实际工程应用,尤其在系统级寄存器通路验证中意义较大。

4　结语

本文针对实际工程应用中UVM寄存器模型的不足,设计实现Lite寄存器模型,并配套开发内建测试激励和Lite寄存器模型自动生成工具。实验结果表明,Lite寄存器模型能够替代UVM寄存器模型完成寄存器验证,且对运行时内存的消耗更小;同时,可兼容UVM和非UVM的模拟验证环境,配合自动生成工具和内建测试用例形成完整的集成电路寄存器验证方案,在实际工程项目中发挥重要作用。

参考文献(References)

[1] HENNESSY J L, PATTERSON D A. Computer Architecture: a Quantitative Approach [M]. 6th ed. San Francisco: Morgan Kaufmann, 2017:12-25.

[2] SPEAR C, TUMBUSH G. System Verilog for Verification: a Guide to Learning the Testbench Language Features: 3rd Edition [M]. Berlin: Springer, 2012: 385-414.

[3] MINTZ M, EKENDAHL R. A Layered Approach [M]. Boston: Springer, 2007:47-68.

[4] FRANCESCONI J, RODRIGUEZ J A, JULIÁN P M. UVM based testbench architecture for unit verification [C]// Proceedings of the 2014 Argentine Conference on Micro-Nanoelectronics, Technology and Applications. Piscataway: IEEE, 2014:89-94.

[5] MEHTA A B. System Verilog Assertions [M]. New York: Springer, 2014:9-28.

[6] MEHTA A B. Functional Verification: Challenges and Solutions [M]. Cham: Springer, 2018:5-12.

[7] Accellera. Universal Verification Methodology (UVM) 1.2 class reference[EB/OL]. [2019-07-16]. https://workspace. accellera. org/downloads/standards/uvm/UVM_Class_Reference_Manual_1. 2. pdf.

[8] Accellera. Universal Verification Methodology (UVM) 1.2 user's guide [EB/OL]. [2019-07-16]. https://accellera. org/images/downloads/standards/uvm/uvm_users_guide_1. 2. pdf.

[9] 张强. UVM实战[M]. 北京:机械工业出版社, 2014:1-20. (ZHANG Q. UVM Practice [M]. Beijing: China Machine Press, 2014:1-20.)

[10] 林玉新. 基于VMM RAL的寄存器验证方法的研究[D]. 西安:西安电子科技大学, 2011:3-40. (LIN Y X. Research on register verification method based on VMM RAL[D]. Xi'an: Xidian University, 2011:3-40.)

[11] IEEE. IEEE standard for IP-XACT, standard structure for packaging, integrating, and reusing IP within tool flows: IEEE Std 1685-2014[S]. Piscataway: IEEE, 2014-09-12.

[12] KRUIJTZER W, VAN DER WOLF P, DE KOCK E, et al. Industrial IP integration flows based on IP-XACT standards [C]// Proceedings of the 2008 Conference on Design, Automation and Test in Europe. Piscataway: IEEE, 2008:32-37.

[13] SUTHERLAND S. Integrating SystemC models with Verilog and SystemVerilog models using the SystemVerilog Direct Programming Interface (DPI) [EB/OL]. [2019-07-16]. http://www. sutherland-hdl. com/papers/2004-SNUG-Europe-presentation_SystemVerilog_DPI_with_SystemC. pdf.

[14] MATTHES E. Python编程:从入门到实践[M]. 袁国忠,译. 北京:人民邮电出版社, 2016:2-14. (MATTHES E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming [M]. YUAN G Z, translated. Beijing: Posts and Telecom Press, 2016:2-14.)

[15] 夏宇闻,韩彬. Verilog数字系统设计教程[M]. 4版. 北京:北京航空航天大学出版社, 2017: 1-50. (XIA Y W, HAN B. Verilog Digital System Design Tutorial [M]. 4th ed. Beijing: Beihang University Press, 2017:1-50.)

This work is partially supported by the National Science and Technology Major Project of China (2017ZX01028-103-002), the Surface Program of National Natural Science Foundation of China (61672525).

PAN Guoteng, born in 1977, Ph. D., associate research fellow. His research interests include microprocessor design and verification.

OU Guodong, born in 1977, Ph. D., research assistant. His research interests include microprocessor design and verification.

CHAO Zhanghu, born in 1989, M. S., engineer. His research interests include microprocessor design and verification.

LI Mengjun, born in 1976, Ph. D., associate professor. His research interests include microprocessor design and verification.