

基于 UVM 的浮点功能部件验证

吴升光^{1,2}, 羊箭锋¹, 冯春阳²

(1 苏州大学 电子信息学院, 江苏 苏州 215000; 2 中晟宏芯有限公司, 江苏 苏州 215000)

摘 要: 为了对复杂浮点运算单元进行功能验证, 设计并实现了一种基于 UVM(Universal Verification Methodology)方法的验证平台. 该平台集成了一套高效的浮点数产生机制, 将浮点用例的求解转化为连分式的求解, 拓宽了传统浮点用例的边界定义, 同时也调用了基于 C 语言的参考模型来自检计算结果, 并结合 FCC(Fast Coverage Convergence)技术, 加快了覆盖率收敛. 应用结果表明, 此验证平台能够对浮点运算单元各功能进行高效验证, 极大地减少验证时间, 且平台内嵌的浮点数产生器也能够移植到其他浮点功能验证平台.

关键词: 浮点运算单元; UVM 验证平台; 中间结果约束; 浮点数生成

中图分类号: TN4

文献标识码: A

文章编号: 1000—7180(2017)04—0121—05

Verification of Floating Point Facilities Based on UVM

WU Sheng-guang^{1,2}, YANG Jian-feng¹, FENG Chun-yang²

(1 School of Electronic and Information Engineering, Soochow University, Suzhou 215000, China;

2 SPT Co. Ltd. Suzhou 215000, China)

Abstract: A test-bench was designed and implemented on the basis of UVM to verify all facilities of the floating-point unit. The test-bench integrates an efficient mechanism to generate floating-point operands and expands the corner of traditional cases with the method of convert floating point numbers to continued fractions. It calls a C model to achieve automatic results comparison, in addition to FCC technique, the coverage was converged quickly. Experimental results show that the test-bench can effectively verify all facilities of the floating-point unit and greatly reduce the time, moreover, the floating-point generator can be used for other floating-point verification.

Key words: floating-point unit; UVM test-bench; intermediate result constraint; floating-point operand generation

1 引言

随着高性能处理器复杂度及性能的不不断提升, 电路规模也越来越大, 对应各个复杂模块的验证难度也日益提升. 在研发过程中, 验证所占用的资源和时间已经达到了 70% 以上, 并在进一步地增加. 符合 IEEE 754 标准^[1] 的二进制浮点运算单元是决定处理器性能的重要部件, 其验证面临的挑战一方面是由于其浮点测试用例空间巨大, 并存在许多难以覆盖的边界点; 另一方面, 浮点操作的实现方法、结构及舍入模式等相当复杂^[2]. 本文基于 UVM 方法, 构建了一个 128 bit 浮点运算单元功能部件的验证平台, 并集成一套浮点数生成机制, 能够使覆盖率在

极短的时间内得以收敛.

2 UVM 方法学及 FPU

2.1 UVM

Accellera 组织于 2010 年推出了 UVM 验证方法学^[3]. 它基于 System Verilog 语言, 吸收了 VMM (Verification Methodology Manual)、OVM (Open Verification Methodology) 等主流验证方法学的优点, 采用了最合适的验证框架实现覆盖率驱动, 结合随机约束生成、平台自测等特点, 是一套成熟、开源的验证方法学.

通常基于 UVM 搭建的验证平台由一系列 UVC (Universal Verification Component) 组成. 各

组件均为 UVM 树的节点,构建与连接都是其父节点的相应操作,实现了各个部分的分工协作. UVC 还具有可以阶段性执行的自动化 phase 机制;支持重载和注册的 factory 机制;TLM (Transaction Level Modeling)级建模等各种特性,可配置、封装,重用性强^[4].

2.2 FPU

FPU(Floating Point Unit)是现代微处理器中的重要单元,其性能高低决定了微处理器进行数据处理与运算的速度,是微处理器的核心部件.

本文实现的验证平台面向一款服务器 CPU 中 128 bit 的 FPU 功能部件,该 FPU 支持四百多条条定/浮点标量和向量指令,有多种不同周期数的执行流水线. 按照指令功能,FPU 可实现精度转换类、乘加类、除法/方根类、加密类等功能,其内部结构如图 1 所示.

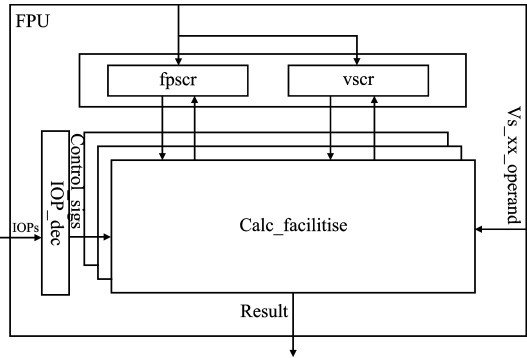


图 1 FPU 功能结构

FPU 接收到指令经过上级部件译码过的中间操作码 (Intermediate Operation Code, IOP), 经过 IOP_dec 模块的控制逻辑, 在同一时刻, FPSCR (Floating Point Status Control Register) 和 VSCR (Vector Status Control Register) 两个特殊寄存器由于上一条指令所产生的状态位 (例如操作异常记录、舍入模式选择、写回抑制等) 会作为特定输入, 影响当前指令的操作、结果的寄存器写回情况, 并更新自身状态进而影响下一条指令.

3 基于 UVM 的 FPU 验证环境

3.1 FPU 验证策略

对于该 FPU 功能部件基于 UVM 的验证, 基本思路是随机约束激励的产生、结果检测与自动比对以及覆盖率收集^[5].

指令对应的控制信号、特殊状态位通过受约束的随机产生, 并分别被打包成 uvm_sequence_item

事务类, 其是能够对功能点进行建模的 uvm_sequence 基类. 然后基于 UVM sequence 机制的 virtual_sequence 在顶层实现和控制多种 sequence 事务类的测试向量同步, 其可以根据应用场景的需要包含多种格式的 sequence. 平台包含一个 C 语言实现的参考模型. 该模型接受指令编号、源操作数及特殊状态信息, 进行相应的操作后将计算结果和特殊状态位信息发送给 uvm_scoreboard, scoreboard 会进行自动比对、记录仿真信息记录到 log 文件并自动收集覆盖率.

3.2 UVM 浮点验证平台结构

该 FPU 共有 7 个功能部件, 本文采用 UVM 仅需搭建一个验证平台便可验证所有的功能部件. 平台的具体实现通过构建一系列 UVC, 并在每个 UVC 中创建涵盖所有基本功能的 base_env, 然后建立顶层的 tb 将这些 UVC 集成起来. 各功能模块 tb 继承 base_env 类并在其上例化自己的接口驱动任务, 然后在顶层通过 UVM 的 override 功能将 base_env 及其各组件重载. 整体层次结构如图 2 所示. 平台对各功能部件的选择通过 SystemVerilog 的预编译宏实现.

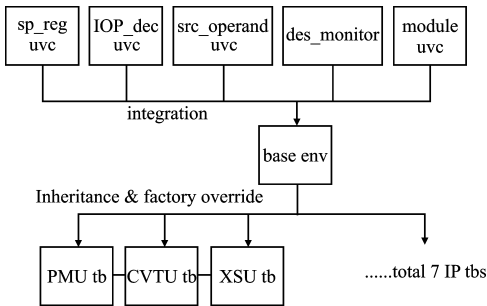


图 2 平台层次结构

基于 UVM 结合浮点运算单元模型搭建的功能部件验证平台如图 3 所示. VS_IPs 是 7 个待验证的功能部件, 每个部件通过 src_op、IOP_dec、sp_reg 三个 UVC 的 driver 和 monitor 实现信息的采样和驱动. src_op UVC 是最主要的激励生成模块, 它具备了一套完整的浮点生成机制. 由其产生的浮点用例能有效覆盖到各种边界, 并且高效可靠, 缩短了验证周期. 下文会对该算法进行介绍. IOP_dec UVC 基于 FPU 中指令译码及控制逻辑简化并建模, 具备模拟指令产生和 IOP 控制信号转换以及对 DUT 接口时序控制、信号采样的完整功能. sp_reg UVC 模拟了两个特殊状态寄存器的功能, 鉴于验证时需要遍历所有情况, 在各功能部件的 uvm_sequence_i-

tem 中列出了影响其执行的状态位,以纯随机来置位. VS_testlib 中包含了各种任务用例(uvm_test),这些 test 调用各 UVC 中的 sequence 对 uvm_sequence_item 进行约束随机以得到符合条件的激励,并根据待测需求调用. 当前 testlib 中为每个功能部件编写了 5 个任务用例,如表 1 所示,后续按照需要可不断扩充. 仿真时 Makefile:

```
$(SIM)+UVM_TESTNAME=$(TC_NAME)
指向了任务所调用的 sequence,在任务用例的 build_phase 中通过 config_db 机制:
uvm_config_db#(uvm_object_wrapper)::set(this, "vs_
xx_tb, xx_vsqr, run_phase",
"default_sequence",xx_sparse_rdm_seq::
type_id::get());
完成了 test 与 sequence 的绑定.
```

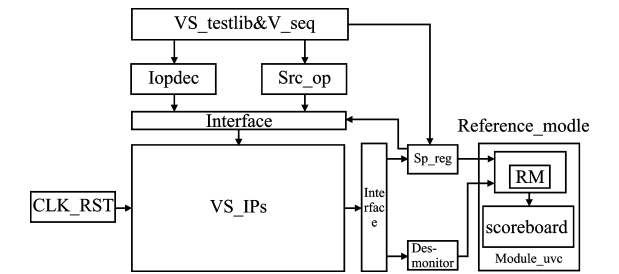


图 3 验证平台框图

表 1 功能部件任务用例列表

任务用例名称	描述
xx_all_type_random_tc	以纯随机的方式验证所有指令和操作数类型的交叉组合
xx_compact_random_tc	验证流水化情况下随机发送指令执行的场景
xx_compact_fixed_tc	验证流水化情况下某一条指令或一类指令的场景
xx_sparse_random_tc	验证间隔发送随机指令的场景
xx_sparse_fixed_tc	验证间隔发送某一条指令或一类指令的场景

3.3 仿真环境配置及启动

仿真过程中,平台有三种可配置方案:一种是指定某一条指令,随机指令需要的源操作数;第二种是指定一源操作数类型,随机符合该操作数类型的一类指令;第三种则属于纯随机,每次随机指令或操作数类型均可. 三种方案都有其特定的应用场景,第一种用于 debug 时错误的迅速定位;第二种会针对某一类数据快速撞出期望的 cornercase(边界情况);第三种则在 regression 跑大量随机 case 时使用. 以第一种情况为例,仿真时在命令行中添加选项:

```
./run, csh -debug -iop_name=xxx ...
virtual sequence 的 new_phase() 中调用如下语句接收 IOP:
$value$ plusargs("iop_name=%d",iop_name);
```

平台收到指令对应 IOP 后,将其值赋给 iop sequence 中的对象. 该对象经过译码逻辑初始化 uvm_sequence_item,同时得到本条指令对应的操作数类型. 数据类型通过一指针被传递到 src_op UVC,经过算法产生的浮点用例被打包成 item 等待 sequence 的发送.

整个平台的启动和停止均由 sequence 控制. v_seq 控制每次发送间隔并同步地手动启动 3 种 UVC 的 sequence.

各 UVC 中的驱动部件 driver 在平台启动后会向 sequencer 请求数据,其通过 seq_item_port 的 get_next_item() 功能向 sequencer 接收 sequence 中 item 指向 req 类的句柄. 在实例化各功能部件的 I/O agent 时,iop,data,reg_state 三种 item 作为参数类重载 req,并提取出控制信号、源操作数以及特殊状态位送到 virtual interface. Driver 的具体驱动行为取决于 sequence 的参数类格式.

功能部件信息的采样分别是对输入和输出信息的采集. 采集时三个 UVC 的 monitor 以及 des_monitor 依据时钟和功能部件的 finish 信号工作. 由于 TLM 通信中 port 是非阻塞性质,若调用写入函数时,module UVC 的 blocking_get_port 被占用无法接收数据,就需要一个暂存的位置. 因此采集到的信息会打包成 item 写入对应的 FIFO. 输入信息传入 module UVC 后驱动功能部件的 C 语言模型,平台调用 DPI 接口,使用如下代码通过指针传递的导入声明将 C 模型封装成 uvm_object 类并载入验证环境:

```
import "DPI -C" function void fadds(input bit [11:0]
inst_iop, input bit [127:0] fra, input bit [127:0] frb, input
bit [31:0] fpscr)
```

输出信息和 C 模型的结果会传到 UVC 的 scoreboard. Scoreboard 中对每条指令建立了比较函数判断两组数据的正确性,若完全一致则打印 match,否则打印 mismatch 相应指令信息并记录 mismatch number,同时统计覆盖率.

一般的仿真验证中,由于每次仿真种子(SV-SEED)是随机的,可能会导致种子的重复. 为此, Cadence 公司为项目定制了一项 FCC(Fast Coverage Convergence)技术. 将功能部件所要满足的功

能点、需要覆盖到的边界条件、舍入情况等特征均列出,建立覆盖率模型,并将指令执行结果作为覆盖率采集点,在相应类文件中创建和随机,同时对已建立的功能点进行采样,编译时加上 $\sum \text{tc_fcc}$ 选项,运行仿真.在每次仿真结束后,根据当前覆盖率值的反馈,控制下次仿真的随机变量生成,减少测试向量的重复,从而加快覆盖率的收敛.

4 浮点用例生成模块

浮点验证中,边界情况的测试用例是最容易被遗漏的,本文提出了一种拓宽边界情况用例范围的方法,将浮点数的生成问题转化为连分式约束的解决^[5-7].考虑到篇幅,下文给出了浮点生成模块中较为复杂的除法用例生成算法.

在讨论算法之前,给出一些需要的定义:

p (精度)、 q : 最终结果与中间结果有效数字位宽;

中间结果:假设最终结果精度为 p ,其有效数字部分 S 为 $1.b_1b_2\cdots b_{q-1}\sigma$,其中 $q > p$ 且 σ 为粘滞位(σ 为 0 或 1).

不可见位:中间结果有效数字的低 $q-p$ 位;

约束掩码: $\{0, 1, x\}$ 的集合,用来表示一组不确定的二进制数.其中 x 表示不定值 1 或 0.

除法测试用例生成算法如下所示.

除法操作中间结果的计算可能会产生一个无限位宽的数.设定 $q = 2p$ 足够约束无限精度商的正确舍入,且这个设定是正确的,这是因为商的前 $2p$ 位就决定了其剩余尾数位(可能为无限位宽)是否必须为全 0,或能够包含 1.因此,算法设定 $q = 2p$.

该算法的第一阶段是随机选择用于中间结果有效数字位 S 以及粘滞位 σ 的值,这些值和给定的掩码相匹配.然后试着生成满足约束的两个输入操作数的有效数字位,若找不到满足约束的输入,则更改中间结果及粘滞位的值,并不断迭代直到找出满足约束的解或者达到预先设定的迭代次数为止.

在随机选择之前,中间结果的选取范围通过消除已确定无解的用例来缩小.当有效数字第 p 位为 1 且粘滞位 X 为 0 时,显然,此种情况无解.

由于除法操作中多数包含舍入操作,有舍入则粘滞位 σ 必为 1.下文的描述中,测试用例生成场景仅针对粘滞位 σ 为 1 的情况, σ 为 0 时类似.

对于这种情况, a/b 的有效数字位必须是 $S_c + \epsilon$ 的形式,其中 $0 < \epsilon < 2^{1-q}$.如此,又存在两种可能性:(i) $S_c + \epsilon = S_a/S_b$, (ii) $(S_c + \epsilon)/2 = S_a/S_b$. 将其

转化为整数系数和变量的公式:

$$(i) (2^{p-1} S_a) / (2^{p-1} S_b) - (2^{q-1} S_c) / 2^{q-1} = \epsilon$$

$$(ii) (2^{p-1} S_a) / (2^{p-1} S_b) - (2^{q-1} S_c) / 2^q = \epsilon/2$$

式中, $0 < \epsilon < 2^{1-q}$, $(2^{q-1} S_c)$, 2^{q-1} , 2^q 为已知量, $(2^{p-1} S_a)$, $(2^{p-1} S_b)$ 为待生成的未知量.

可以发现,在上述情况中,未知分数的分母 $(2^{p-1} S_b)$ 明显小于已知分数的分母 2^{q-1} 或 2^q .因此,此处存在的问题是要通过一个小分母的分数找到一个具有大分母已知分数的近似值.基于连分数的求解是得到这种类型的近似值的最接近自然表示的方法^[8].

使用连分数生成(浮点)解的方法,如下所述:

记 $\bar{p} = (2^{p-1} S_a)$, $\bar{q} = (2^{p-1} S_b)$ 为待生成的未知整数,写出(i), (ii) 两种形式:

$$(i) (2^{q-1} S_c) / (2^{q-1}) < \frac{\bar{p}}{\bar{q}} < [(2^{q-1} S_c) + 1] / (2^{q-1});$$

$$(ii) (2^{q-1} S_c) / (2^q) < \frac{\bar{p}}{\bar{q}} < [(2^{q-1} S_c) + 1] / (2^q).$$

这些未知整数必须均为 p 位二进制整数,亦即它们必须满足如下关系:

$$2^{p-1} \leq \bar{p} < 2^p, 2^{p-1} \leq \bar{q} < 2^p$$

下文以情况(i)为例进行算法的说明;情况(ii)与之类似,不再赘述.首先写出不等式(i)中左右两侧给定分数的连分数扩展.记 $\bar{p}'/\bar{q}' = (2^{q-1} S_c) / (2^{q-1})$, $\bar{p}''/\bar{q}'' = [(2^{q-1} S_c) + 1] / (2^{q-1})$.由于这两个分数都十分接近,很可能它们的扩展会在某一点相等,所以有 $\bar{p}'/\bar{q}' = [\bar{p}_0'; \bar{p}_1', \dots]$, $\bar{p}''/\bar{q}'' = [\bar{p}_0''; \bar{p}_1'', \dots]$, 其中 $\bar{p}_0' = \bar{p}_0''$, $\dots$, $\bar{p}_{i-1}' = \bar{p}_{i-1}''$.如果记 $[\bar{p}_0; \bar{p}_1, \dots]$ 为 $\bar{p}/\bar{q}$ 的扩展,则其前 i 项必定与 $\bar{p}'/\bar{q}'$ 和 $\bar{p}''/\bar{q}''$ 一致.如此,三个分数的前 i 项渐进式就可以得出(相等)并记为 P_m/Q_m , $m = 0, 1, 2, \dots, i-1$.

记三个分数的末尾部分为 $\hat{p}'/\hat{q}' = [\hat{p}_i'; \hat{p}_{i+1}', \dots]$, $\hat{p}''/\hat{q}'' = [\hat{p}_i''; \hat{p}_{i+1}'', \dots]$, $\hat{p}/\hat{q} = [\hat{p}_i; \hat{p}_{i+1}, \dots]$.显然,前两者为已知,第三个尾数为待生产的未知量,且 $\hat{p}/\hat{q}$ 位于 $\hat{p}'/\hat{q}'$ 和 $\hat{p}''/\hat{q}''$ 之间.应用连分数的属性(参考文献[8]),可得 $\bar{p} = P_{i-1}\hat{p} + P_{i-2}\hat{q}$, $\bar{q} = Q_{i-1}\hat{p} + Q_{i-2}\hat{q}$,进而可以转化为 $\hat{p}, \hat{q}$ 必须满足的条件.要生成浮点解,首先要解决这些条件的约束问题,下文给出了 i 为偶数情况下待解决的条件(i 为奇数时,保留第一项条件不用):

$$(I) \hat{p}' / \hat{q}' < \hat{p} / \hat{q} < \hat{p}'' / \hat{q}''$$

$$(II) 2^{p-1} \leq P_{i-1} \hat{p} + P_{i-2} \hat{q} < 2^p$$

$$(III) 2^{p-1} \leq Q_{i-1} \hat{p} + Q_{i-2} \hat{q} < 2^p$$

在生成了 $\hat{p}, \hat{q}$ 解的同时,也就生成了 $\bar{p}, \bar{q}$ 的解,方法如下。

第一步,将(I),(II),(III)化成3个新不等式, $F_1(\hat{q}) < \hat{p} < G_1(\hat{q}), F_2(\hat{q}) < \hat{p} < G_2(\hat{q}), F_3(\hat{q}) < \hat{p} < G_3(\hat{q})$,其中 $F_j(\hat{q}), G_j(\hat{q})$ 和 $\hat{q}$ 成线性关系

第二步,上述3个不等式左右两边变量两两组合,构成仅与 $\hat{q}$ 相关的不等式,即 $F_j(\hat{q}) < G_k(\hat{q})$ 。其中某些对所有 $\hat{q}$ 均满足的不等式可以不作考虑(例如, $F_j(\hat{q}) < G_j(\hat{q})$)。

第三步,通过求解每个与 $\hat{q}$ 相关的不等式来确定 q 的上下边界。

第四步,若上下边界交集为空,则无解。若上下边界存在交集,则在交集中随机选取一正整数作为 $\hat{q}$ 的值。

第五步,第一步中的关系式即成为对 $\hat{p}$ 的约束,这三个区间的交集中若存在正整数,则随机选取一数作为 $\hat{p}$ 的值,若不存在,则返回第四步。

第六步,返回 $\bar{p} = P_{i-1} \hat{p} + P_{i-2} \hat{q}, \bar{q} = Q_{i-1} \hat{p} + Q_{i-2} \hat{q}$ 并停止迭代。

对于情况(i)的 $(S_c, S_c + 2^{1-q})$ 以及情况(ii)的 $(S_c/2, S_c/2 + 2^{-q})$,可以通过细化初始区间来提高算法的效率,因为两个 p 位整数比率的二进制表示不可能包含大于 p 位连续的0或者 p 位连续的1,用 $(S_c + 2^{-q-p}, S_c + 2^{1-q} - 2^{1-q-p})$ 替代情况(i)中的区间或者用 $(S_c/2 + 2^{-q-p-1}, S_c/2 + 2^{-q} - 2^{-q-p})$ 替代情况(ii)中的区间并不会损失解的数目,在某些情况下,这种细化会提高算法的效率。

5 验证效率与结论

项目前期的验证中,平台的浮点用例通过集成伯克利大学浮点生成器 Testfloat 来产生^[9]。在针对浮点数置换功能部件(PMU)的仿真验证中,使用 Testfloat 生成的一百万条符合 IEEE 754 标准的浮点用例,测试的功能覆盖率为 83.19%。后续采用本文实现的浮点生成模块同样产生一百万条浮点用例,最终的功能覆盖率为 99.99%,并能覆盖到绝大多数中间结果的边界情况,如图4所示。

经与同期展开的“C+内嵌汇编”验证对同一功能部件的54个功能测试点进行覆盖率收敛对比,结果如图5所示。可以看出,由于 UVM 在前期比“C

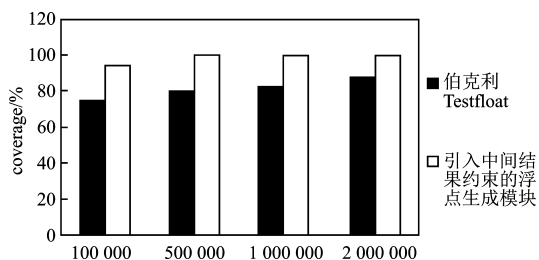


图4 功能覆盖率对比

十内嵌汇编”多出了构建验证事务及环境的时间,初始阶段功能覆盖点的击中速度要慢于“C+汇编”。但由于 UVM 平台的可重用性、定制的 FCC 功能以及高效的浮点生成模块,在 21 个小时内就达到 90% 以上的覆盖率,而“C+汇编”验证达到相同的覆盖率需要超过 100 个小时。最终在第 37 个小时本平台达到覆盖率收敛,远远高 C 验证。

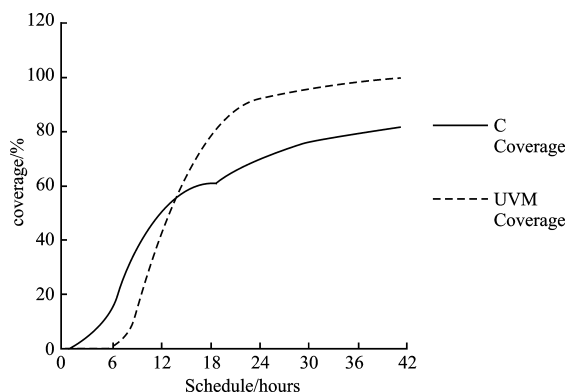


图5 覆盖率收敛速度对比

6 结束语

本文基于 UVM 方法构建了一个 128 bit 浮点计算单元的验证平台,引入对中间结果约束的算法高质量地产生浮点数,能够高效地对 FPU 所有功能部件进行验证,具有良好的可重用性,加速了覆盖率收敛,极大地提高了浮点验证的效率,缩短了验证周期。同时,浮点生成模块能够作为一个浮点生成器移植于到各类浮点验证平台中。

参考文献:

- [1] IEEE. IEEE Std 754-2008, IEEE standard for floating-point arithmetic[S]. IEEE,2008.
- [2] 冯春阳,杨靓,黄士坦. 随机法和 Hensel Lifting 联合的浮点乘测试用例生成方法[J]. 计算机辅助设计与图形学学报,2014,26(9):1509-1520.

(下转第 130 页)

参考文献:

[1] 新浪网. 7 亿人! 四分之一上网只用手机[EB/OL]. [2015-07-16]. <http://news.sina.com.cn/c/2016-08-04/doc-ifyxuszpp2832693.shtml>.

[2] 姜逸文. 移动互联网下的信息安全问题及应对策略[J]. 中国新通信, 2014, 16(10): 123-124.

[3] 张樵苏. 电信运营商积极构建网络安全生态圈[EB/OL]. [2015-07-16]. http://news.xinhuanet.com/2015-07/16/c_1115950293.htm.

[4] 张芳琴. PDCA 在建筑安全管理中的应用[J]. 现代职业安全, 2015, 15(5): 88-90.

[5] 丁建立. 网络安全[M]. 武汉: 武汉大学出版社, 2007.

[6] 伊查克. 企业生命周期[M]. 赵壑, 译. 北京: 华夏出版社, 2004: 8.

[7] 张爱玲, 徐杰. 基于全生周期的运营商安全管理平台

安全策略模型分析[J]. 微型机与应用 2016, 35(8): 1-3.

[8] 中国移动集团. QB-Y-001-2011 11Y001 中国移动业务支撑系统安全管理平台技术规范 V1. 0. 0. doc [s]. 中国移动集团, 2011.

[9] 中国移动集团. QB-Y-049-2013 13Y049 中国移动业务支撑网安全管理平台技术规范 V2. 0. 0. [s]. 中国移动集团, 2013.

作者简介:

张爱玲 女, (1972-), 硕士, 副教授. 研究方向为软件工程、项目管理. E-mail: 724817362@qq.com.

徐杰 女, (1979-), 硕士, 高级工程师. 研究方向为软件工程、电子商务.

(上接第 125 页)

[3] Accellera. Universal verification methodology 1. 1 user's guide[M]. Cadence Design System Inc., Mentor Graphics Cor., Synopsys Inc., 2011.

[4] 熊涛, 蒋见花. 基于 UVM 方法学的纵向可重用研究[J]. 微电子学与计算机, 2016, 33(4): 64-68.

[5] Wazlowski M E, Adiga N R, et al. Verification strategy for the Blue Gene/L chip[J]. IBM Journal of Research and Development, 2005, 49(2/3): 303-318.

[6] Ziv A. Solving range constraints for binary floating-point instructions[C]//Proc. 16th IEEE Symposium on Computer Arithmetic (ARITH16). IEEE, 2003: 158-164.

[7] Aharoni M. Solving constraints on the intermediate result of decimal floating-point operations[C]//Proc.

18th IEEE Computer Arithmetic Symp. (ARITH 18). IEEE, 2007: 38-45.

[8] Khinchin A Y. Continued fractions[M]. New York: Dover, 1997: 105-126.

[9] John R Hauser. Berkeley testFloat[EB/OL]. [2015-10-15]. <http://www.jhauser.us/arithmetic/TestFloat.html>.

作者简介:

吴升光 男, (1991-), 硕士研究生. 研究方向为集成电路设计与验证.

羊箭锋(通讯作者) 男, (1978-), 副研究员, 硕士生导师. 研究方向为信号与信息处理.

E-mail: jfyang@suda.edu.cn.