

基于高效约束解决算法的浮点数生成器设计

胡广垠, 王子欧, 吴升光, 王申卓
(苏州大学 电子信息学院, 江苏 苏州 215000)

摘 要: 为了对微处理器中浮点运算单元 FPU(floating-point unit)进行高效的功能验证,对浮点运算的边界情况进行了研究,引入了对中间结果(intermediate result)的约束解决算法(constraint solved arithmetic). 与传统的对浮点运算单元的功能验证相比,基于该约束算法的浮点数生成器,拓宽了浮点边界情况的可选范围,有效提高了验证效率. 实验结果表明,集成该浮点数生成器的 UVM 验证平台,能够在 12 小时的测试时间内,对一个浮点运算子模块(floating-point subunit)达到超过 99%的覆盖率.

关键词: 浮点运算单元;中间结果(IR);约束解决算法(CSA)

中图分类号: TP301.6

文献标识码: A

文章编号: 1000—7180(2016)11—0078—05

Design of Floating Point Number Generator Based on High-Efficiency Constraint Solving Algorithm

HU Guang-yin, WANG Zi-ou, WU Sheng-guang, WANG Shen-zhuo

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

Abstract: In order to verify the function of the floating-point unit in the microprocessor high-efficiently, numerous corner cases of floating-point arithmetic has been studied. Several arithmetics are introduced to solve the constraints of the intermediate results. Compared with the traditional function verification of floating-point unit, the floating-point number generator based on the constraint solved algorithms has widened the scope of the optional floating-point corner cases, which effectively improves the verification efficiency. Experimental results show that the UVM verification platform which integrated the generator can test one floating-point subunit efficiently, which achieves high coverage beyond 98%, within 12 hours of testing time.

Key words: floating-point unit; intermediate result(IR); constraint solved algorithm(CSA)

1 引言

浮点运算单元(FPU)是现代微处理器中的重要模块,其性能高低决定了微处理器进行数据处理与运算的速度. 伴随着芯片复杂度的不断上升,设计一种高效的 FPU 并对其进行有效的验证已经成为了一项复杂而充满挑战性的任务^[1]. 对于遵循 IEEE754 标准^[2]的浮点运算操作,巨大而繁复的测试空间(test space)中包含了大量的难以被定位的边界情况.

主流的 FPU 验证方法有两种:形式化验证(Formal verification)与基于仿真(Simulation)的验

证. 形式化验证技术能够有效地证明浮点运算指令的正确性^[3],但这种方法需要十分冗长的体力劳动时间以及相当多的资金投入. 因此在对现有 FPU 的验证中,我们采用了仿真与形式化验证相结合的技术:在仿真达到一个相对稳定的程度时,再采用形式化验证技术来进行形式化的证明.

本文面向 Power 架构微处理器的浮点运算单元的仿真实验,设计了基于高效约束解决算法(constraint solved arithmetic)的浮点数生成器,并在 FPU 的仿真实验流程中作为主要的激励生成工具投入了使用.

2 浮点操作数中间结果

在 IEEE754 标准中,针对浮点数运算,除了输入操作数和结果操作数,还定义了中间结果操作数 (intermediate result). IEEE 标准中规定:每个按照该标准所执行的浮点运算,在输出最终数值结果 (numeric result) 之前,会被表示成一个无限精度、无穷范围的值,该值就是中间结果操作数.对中间结果操作数进行舍入 (rounding) 得到最终结果操作数^[3].

2.1 不可见位的概念

在传统的针对浮点运算单元的基于约束生成方法的仿真测试中,测试用例 (test cases) 仅仅依据对输入操作数与结果操作数的约束生成,没有引入对中间结果以及不可见位的约束方法.

不可见位 (Invisible bits) 是指浮点运算过程中,在对中间结果操作数进行舍入得到最终结果操作数的过程中,中间结果被舍去的部分位,如图 1 所示.假设最终结果的精度为 p ,显然中间结果的精度会大于 p .



图 1 IEEE 规定中间结果的格式 (64 位双精度浮点数)

- S 位:符号位.
- C 位:进位位,用来放置有效数字部分产生的进位输出.
- L 位:隐藏位,作为有效数字部分的起始位.
- FRACTION 位:52 位宽,作为最终结果的有效数字部分.
- Invisible 位:不可见位.
- X 位:粘连位 (sticky bit),作为不可见位的扩展位,由不可见位的低位所有位进行 OR 运算取得.当 X 位为 0,表示该中间结果与实际精确结果相同.

2.2 中间结果的约束

在对浮点运算单元进行功能验证的流程中存在大量复杂的边界情况 (corner cases). 以 rounding 运算为例,对舍入运算的验证^[4]包含了许多重要的测试点,相对应的测试用例如果仅考虑对输入操作数与最终结果的约束,那么会遗漏许多重要的边界情况^[11]. 例如,当一个待舍入的无限精度的精确值与某个舍入特殊值 (可以是两个浮点数的中间值) 非常接近但又不完全相等时,在这种情况下,逻辑设计的

准确性要求 FPU 能够在两个可能的浮点值中准确地选择那个正确值. 一旦选择错误,会导致舍入运算的错误以及最终结果的错误.

对中间结果的约束能够提供满足上述测试点要求的测试用例,只需要将不可见位的约束条件设置为 1111...111 并将粘连位 X 设置为 1,即可产生与某个舍入特殊值非常接近但又不完全相等的浮点数.

从这个例子可以看出增加对中间结果的约束能够拓宽测试空间的范围,并增加了大量可定义的边界情况.

3 FloatGen

为了能够对中间结果进行约束,产生能够覆盖到中间结果边界情况的测试用例,我们设计了一个基于高效约束解决算法的随机浮点数生成器 FloatGen. 该软件使用 c 语言编写,在 Linux-x86_64 下使用 GCC 编译. Linux 版本为 RedHat Enterprise Linux Server release 6. 6 (Santiago). FloatGen 程序框架如图 2 所示.

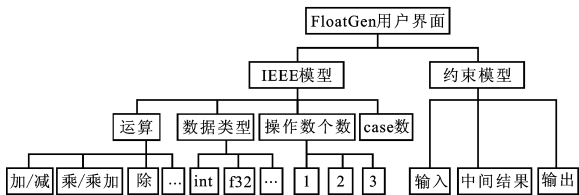


图 2 FloatGen 框架结构

FloatGen 包含了两类算法模型, IEEE 模型和约束模型^[8,9,10], IEEE 模型包括 (运算类型, 数据类型, 操作数个数, case 数), 约束模型包括 (输入, 中间结果, 输出结果).

当需要生成用于 f32 格式浮点数加法操作的 100 000 条测试用例时,在图 3 所示的用户界面下输入指令:

```
./FloatGen-add-f32-3-n 100000 - E
```

程序会调用如下代码:

```
if ( ! strcmp( argPtr, "-n" ) ) {
    if ( argc < 2 ) goto optionError;
    genLoops_forever = true;
    genLoops_givenCount = true;
    i = strtol( argv[1], (char * *) &argPtr, 10 );
    if ( * argPtr ) goto optionError;
    genLoops_count = i;
}
if ( ! strcmp( argPtr, "-add-f32" ) || ! strcmp( argPtr,
"-add-float32" ) ) {
```

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
FloatGen [<option>... ] [<operation>] [<type>] [<operands num>] [<cases num>] [<input>] [<intermediate>]
<option>: [-r is default]
-help
-prefix <text> --Write <text> as a line of output before any test cases.
-forever --Generate test cases indefinitely.
-rnear even --Round to nearest/even.
-rminMag --Round to minimum magnitude (toward zero).
-rmin --Round to minimum (down).
-rmax --Round to maximum (up).
-rnear maxMag --Round to nearest/maximum magnitude (nearest/away).
-tininessbefore --Detect underflow tininess before rounding.
-tininessafter --Detect underflow tininess after rounding.
-notexact --Rounding to integer is not exact (no inexact exceptions).
*****IEEE Models*****:
<operation>: <type> <operands num>
-add <int> <int> <operands num>
-sub <u132> --Unsigned 32-bit integer. <operands num>
-multiply-add <u164> --Unsigned 64-bit integer. <operands num>
-conv <u128> --Unsigned 128-bit integer. <operands num>
-divide <int> --Signed 32-bit integer. <operands num>
-cvtFP2Int <int> --Signed 64-bit integer. <operands num>
-cvtFP2FP <int> --Signed 128-bit integer. <operands num>
-cvtInt2FP <float> <operands num>
-RFI <float> --Binary 32-bit floating-point (single-precision).
-f32 --Binary 64-bit floating-point (double-precision).
-f128 --Binary 128-bit floating-point (quadruple-precision).
*****Constraint models*****:
<input>:
-A --Almost Equal Significands
-B --Basic Input Classes
-C --Division by Zero
-D --Special Input Significands
<intermediate>:
-E --Basic Output Classes
-F --Output Corner Case Neighborhoods
-G --Near Overflow
-H --Near Underflow
-I --Rounding Corner Cases
suxue@localhost/
```

图 3 FloatGen 用户界面

```
functionCode = F32_ADD;
.....
switch ( functionCode ) {
.....
case F32_ADD;
    trueFunction_abz_f32 = f32_add;
    goto gen_abz_f32;
```

程序运行结果会生成 100 000 条符合约束要求的测试用例。如果需要其他约束条件,只需要在指令中继续添加其他可选项。

FloatGen 通过对不同约束进行组合,提高了生成的测试用例的可选择性,使 FPU 验证工程师能够根据不同的子模块中的不同指令,针对性地选取不同的约束组合,如图 4 所示。从而产生高效的浮点测试用例,提高了覆盖率,缩短了验证周期。

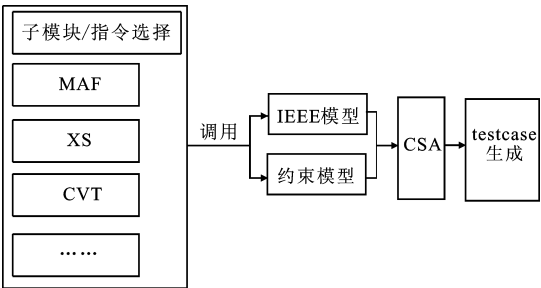


图 4 FloatGen 工作流程

4 对中间结果的约束解决算法 IRCSA(Intermediate result constraint solved algorithm)

对中间结果约束可以看成是对不可见位的约束:给定某类浮点操作运算类型(加/减/乘/除/舍入...),以及所要求解的边界情况约束条件,通过约束解决算法(CSA)得出两个浮点操作数 a 和 b ,

使 a 和 b 经过该类型浮点运算后能够得到满足该约束条件的中间结果。如果不存在符合条件的 a 和 b ,则声明没有解决方法。下文中给出了 FloatGen 中针对加/减法运算的 IRCSA。

4.1 声明和定义

p (精度):表示结果操作数的有效数字部分位宽。

q :表示中间结果操作数的有效数字部分位宽。

精确结果:表示浮点运算的实际结果,具有无穷的精度和无限宽度的位宽。

不可见位:中间结果有效数字部分的低 $q-p$ 位。

约束掩码:一串包含 $\{0,1,x\}$ 的字符,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。

指数偏移:表示加/减运算中两个输入操作数的未偏移指数(unbiased exponent)的差值。为了正确表示该运算,将第二个操作数的指数进行偏移,从而使两个输入操作数的指数相等。

指数逼近:表示加/减运算中,当两个输入操作数的绝对值十分接近时,输出结果的指数位同时小于两个输入操作数的指数位的情况。

4.2 加/减法运算的 IRCSA

4.2.1 确定 q 的取值

为了确保能够正确地进行舍入操作,在硬件设计中加运算或者减运算要求 q 的位宽至少为 $p+2$ 。为了满足更高的精度要求以及遵循简化硬件设计的目的,实际硬件设计中定义的 q 远大于该最小值。因此当一个 FPU 的设计中包含了乘加操作($a \times b + c$)时,乘加操作与加操作能够共用相同的硬件模块^[6]。

在加减法操作类型的 IRCSA 中我们没有将 q 定为固定值,定义 $p+2 \leq q \leq 4p$ 。

4.2.2 算法流程

算法的框架如图 5 所示。主体是基于列出的所有所需边界情况的指数基(base exponent cases)。每个指数基定义了两个输入操作数之间、第一个操作数与中间结果之间指数的差值,以及该运算是加或者减。

算法的步骤如下:

- (1)列出一个表格,表格包含了所有所需的指数基(能够根据需求不断补充新的指数基)。
- (2)在表格中随机选取一个指数基。
- (3)使用该指数基,构建定点数的约束掩码,用来表示浮点约束问题的有效数字部分。
- (4)使用 FloatGen 中的 FXgen 函数,找出符合

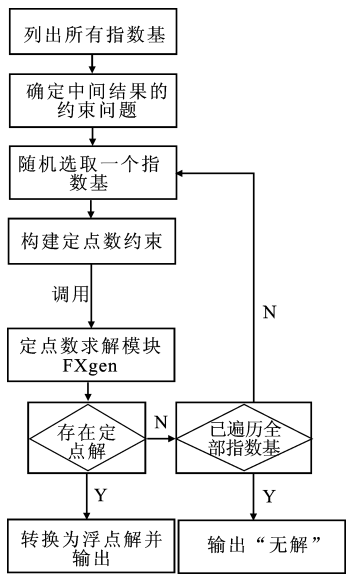


图 5 对加/减法操作的中间结果约束解决算法的流程

所构建的定点数约束掩码的一个随机的解决方法。

(5)根据所选指数基,将该定点解转换为浮点数解。

4.2.3 列出所有所需的指数基

- 每一个指数基包含了一下信息:
- (1) 输入操作数 a 和 b 的符号位.
 - (2) 两个输入操作数指数的约束关系, 定义为 $E_a - E_b = m$, $E_a - E_b \geq m$ 或 $E_a - E_b \leq m$.
 - (3) 第一个操作数和中间结果操作数指数的约束关系, 定义为 $E_a - E_c = n$, $E_a - E_c \geq n$.

以下列举了两类指数基组的例子:

在这两个例子中,我们假定 $p=5, q=12$. 字符串 1AAAA 代表 a 的有效数字部分, 字符串 1BBBB 代表 b 的有效数字部分, 字符串 1CCCCCCTT. . TTT 代表精确结果 c 的有效数字部分. 字符串最后的 TT. . TTT 位表示了精确结果超出中间结果的部分, 中间结果的 sticky bit 是由这些位进行 OR 运算后得出.

- (1)加法||没有进位||存在指数偏移
- 运算操作为加法, $E_a - E_c = 0$, $E_a - E_b$ 在范围 $[1, q-1]$ 中枚举取值. 该类指数基组包含了 $q-1$ 个不同的指数基. 以 $E_a - E_b = 1$ 为例, 有效数字部分结构如下所示.
- Sa : +1AAAA00000000
Sb : +01BBBB00000000
Sc : +1CCCCCCCCCCT
- (2)减法||指数逼近位数 1 bit||存在指数偏移
- 运算操作为减法, $E_a - E_c = 1$, $E_a - E_b \geq q$, 该类

指数基组只包含了一个指数基. 有效数字部分结构如下所示.

Sa : +1AAAA000000000000
Sb : +00000000000001BBBB
Sc : +01CCCCCCCCCCTTTTT

对于加法操作, 其余的各个指数基的区别, 在于是否有进位生成, 以及指数偏移是否为 1 或 0, 或大于 1. 对于减法操作, 不同取决于指数逼近位数为 0, 为 1, 或大于 1 bit, 以及两个操作数之间是否存在指数偏移. (不是所有的上述情况的组合都是合法的有效情况)

4.2.4 构建定点数的约束掩码

在选定一个具体的指数基以后, 相当于构建了一组定点数的约束掩码. 这些约束掩码被作为激励灌入 FXgen 中来得出有效数字部分的定点解.

当操作为加法, 使用 FXgen 并将进位序列的约束掩码定义为 $0xxx \cdots xxx0$. 当操作为减法, 将约束掩码的规则重定义为 $Mx = My + Mz$, 再使用 FXgen 并将进位序列约束掩码定义为 $0xxx \cdots xxx0$.

此时浮点操作数的有效数字部分被表示为了定点数的约束掩码: Mx 表示操作数 a 的有效数字部分, My 表示操作数 b 的有效部分, Mz 表示中间结果的有效部分.

Mx 和 My 被定义为 $1xxx. xxx$ 的形式, 开头为 1, 其余位为 x . Mz 的形式为: 开头为 1, 再接 p 位 x , 再接不可见位的约束掩码, 最末位为粘连位的约束掩码.

以“加法操作, 粘连位为 1, $p=5, q=12$ ”为例. 不可见位的约束掩码表示为 IIIIII. 在指数基 ($E_a - E_c = 0$, $E_a - E_b \geq 8$, 粘连位为 1) 下, 定点数约束掩码如下所示.

- Mx: 1xxxx00000000
My: +000000001xxxx
Mz: 1xxxxIIIIIIx
- 在此基础上, 我们还要考虑某些复杂的特殊情况. 例如, 由于在 FXgen 中, 约束掩码中所有 x 位的组合可能会决定粘连位的值, 因此我们有时无法一次就让 FXgen 产生粘连位为 1 的解决方案. 很明显当 Fxgen 产生的 x 位都为 0 时, 尽管粘连位被约束为 1, 但是在算法中会把结果舍弃并重新进行生成操作直到一个定点解被找出. 在每次重新生成的操作过程中, 我们在可能决定粘连位值的 x 位中选出一位, 将其值设置为 1, 其余 x 值不变. 在每次迭代中, 随机选取一位 x 值进行设置. 在此方法下, 如果有定点解存在, 那么我们一定会在某次迭代中找到.

4.2.5 构建浮点解

通过确定符号位,指数部分,有效数字部分,能够构建两个浮点操作数 a 和 b ,以及符合不可见位约束掩码的中间结果操作数 $a+b$.

符号位在之前由指数基确定,有效数字部分在 Fxgen 生成的定点解中提取.

在选取指数部分的过程中,除了指数基中所给定的约束,我们还需要考虑到浮点数格式的定义(IEEE754).我们先在一个给定范围 $\max(E_{\min}, E_{\min} + m) \leq E_a \leq \min(E_{\max}, E_{\max} + m)$ 内选取一个 E_a 值.然后根据 E_a 与 E_b 的关系,确定 E_b 的约束范围,再随机选取一个 E_b 的值.

此算法通过给 FXgen 随机输入某一个指数基来产生所需结果.如果 FXgen 找到了一个定点解,算法会生成符合约束的浮点操作数.如果 FXgen 发现没有对应的定点解存在,那么会随机选取另一个指数基,直到发现任意一个定点解或者列表中所有的指数基都已经被覆盖到时.此时表明对该问题不存在任何解决办法.

上述算法通过将浮点数的加法问题转换成定点数问题,从而解决了在浮点数加减法操作中的中间结果约束问题,并且该算法也可以用在乘加操作中.

5 实验结果

应用 FloatGen 对项目组设计的 Power 架构的浮点运算单元进行验证.在 Cadence 的 Incisive(版本号:15.10-S006)软件仿真环境下,自带的 imc 软件能够有效地统计功能覆盖率.在针对 Power 架构 FPU 的一个运算子模块 PM 模块(浮点数置换单元)^[7]的 UVM 验证中,仅使用 FloatGen 的 IEEE 模型生成的十万条符合 IEEE754 标准的 case,测试的功能覆盖率为 83.19%.同时使用 FloatGen 的 IEEE 模型与约束模型生成十万条 case,最终的测试功能覆盖率为 95.13%,并且能够覆盖绝大多数中间结果的边界情况,如图 6 所示.

最终在 12 小时的测试时间内,UVM 验证平台完成对 PM 浮点运算子模块的 99.47% 的覆盖率.与之相比,当验证平台未集成该 FloatGen 而使用简单约束的随机数作为激励时,针对 PM 模块的验证流程中,达到相同的覆盖率所花费的时间超过 60 小时,如图 7 所示.

测试结果表明,FloatGen 中的 IEEE 模型能够生成符合 IEEE 所规定的各种类型的浮点操作数.约束模型则能够在 IEEE 模型生成数的基础上,通

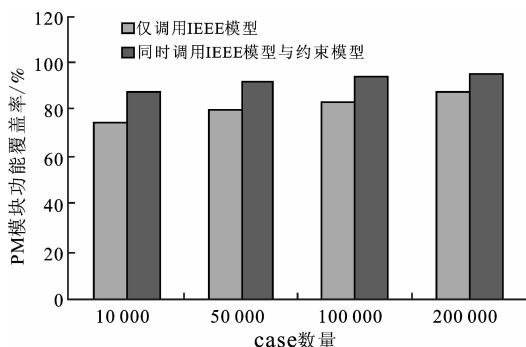


图6 FloatGen 测试功能覆盖率

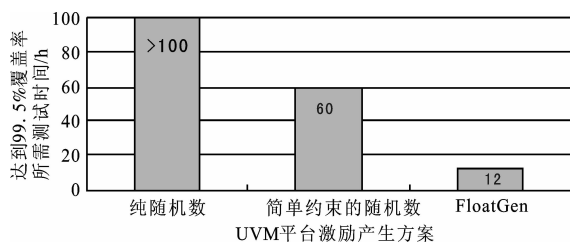


图7 不同激励解决方案下达到规定覆盖率要求所需测试时间
过调用不同的约束算法,生成满足各种不同约束条件的操作数.

6 结束语

FloatGen 在实际的验证工作中生成的高覆盖率,高针对性的测试集,极大地提高了验证效率,缩短了验证周期,并且面向 FPU 中所有的浮点运算子模块,拥有良好的重用性.针对不同功能的浮点模块,项目组验证工程师能够根据不同的模块运算特性,选择 FloatGen 中提供的不同约束进行组合,从而产生满足不同要求的测试集.

在下一步研究中,我们将会为 FloatGen 添加更多的可选择的约束条件.并且针对十进制浮点运算单元 DFU(decimal floating-point unit),加入十进制浮点数的生成算法.并且将 FloatGen 软件充分集成到项目组开发的 UVM 验证平台中,最终实现验证流程的自动化与高效化.

参考文献:

- [1] 于伽. 随机测试程序发生器的设计与实现[J]. 微电子学与计算机, 2012, 29(7): 103-106.
- [2] M. Aharoni, Solving Constraints on the Intermediate Result of Decimal Floating-Point Operations[C]//Proc 18th IEEE Computer Arithmetic Symp. Santiago: IEEE, 2007: 38-45.

表 1 功能结构表

总功能	第一级分解	第二级分解	第三级分解	第四级分解
FR ₀ =航天器舱布局 CAD 设计	FR ₁ =简化待布物		FR ₁₁ =选择数学优化模型	
			FR ₁₂ =选择符号模型	
			FR ₁₃ =选择可视化仿真模型	
	FR ₂ =确定航天器舱布局	FR ₂₁ =确定航天器舱初始布局	FR ₂₁₁ =显示布局方案信息	
			FR ₂₁₂ =获取布局方案信息	
			FR ₂₂₁₁ =显示启发式法信息	
			FR ₂₂₁ =显示布局优化算法信息	FR ₂₂₁₂ =显示演化法信息
				FR ₂₂₁₃ =显示遗传法信息
				FR ₂₂₁₄ =显示模拟退火法信息
		FR ₂₂ =优化航天器舱布局	FR ₂₂₂₁ =采用启发式法	
			FR ₂₂₂₂ =采用演化法	
			FR ₂₂₂₃ =采用遗传法	
			FR ₂₂₂₄ =采用模拟退火法	
			FR ₃₁ =依据层次权重进行定性定量分析	
		FR ₃ =评价航天器舱布局	FR ₃₂ =用模糊数学进行总体评价	
			FR ₃₃ =研究信息部分清楚、部分不清楚并带有不确定性现象	
			FR ₃₄ =研究解决不相容问题	

[5] Nam Pyo Suh. 公理设计:发展与应用[M]. 谢友柏, 译. 北京:机械工业出版社,2004.

[6] 厉金宏,樊树海,方叶祥,等. 可重构制造系统设施布置公理化设计研究[J]. 工业工程与管理,2011,16(2): 72-76.

[7] 曾威,史彦军,滕弘飞. 卫星舱布局 CAD 系统:Pro/Engineer 二次开发若干关键技术[J]. 计算机辅助设计与

图形学学报,2005,17(11): 2575-2579.

作者简介:

张 雷 男,(1991-),研究生. 研究方向为公理化设计.
E-mail:ZL859528@163. com

(上接第 82 页)

[3] 陈博文. 浮点乘加部件的自动化形式验证[J]. 计算机研究与发展,2010,47(S1):262-267.

[4] Goni O,Todorovich E. Components for Coverage-Driven Verification of floating-point units[C]// IX Southern Conference on Programmable Logic (SPL). Montpellier,France: IEEE,2014:1-7.

[5] AAR Sayed-Ahmed. Three engines to solve verification constraints of decimal Floating-Point operation[C] // Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers (ASILOMAR). Nontpellier, France: IEEE, 2010: 1153-1157.

[6] Ziv A. Solving the generalized mask constraint for test

generation of binary floating point add operation[J]. Theoretical Computer Science, 2003, 291 (2): 183-201.

作者简介:

胡广垠 男,(1993-),硕士研究生. 研究方向为集成电路设计与验证. E-mail:guangyin_hu@sina. com

王子欧 男,(1968-),副研究员,硕士研究生导师. 研究方向为纳米器件与存储系统.

吴升光 男,(1991-),硕士研究生. 研究方向为集成电路设计与验证.

王申卓 男,(1992-),硕士研究生. 研究方向为集成电路及 SOC 设计.