https://doi.org/10.1051/jnwpu/20234151024

Chisel 构建处理器模型的功能验证方法研究

吴乐宁[1], 王淼[2], 陈福[1]

(1.航空工业西安航空计算技术研究所, 陕西 西安 710076; 2.西北工业大学 计算机学院, 陕西 西安 710072)

摘 要:随着航空硬件设计复杂度的提高,芯片验证技术已经成为了芯片设计的难点。为了有效缩短设计流程的总体工作时间,有必要在占据设计大量时间的验证中,研究出快速寻找设计错误的方法。被测设计是兼容 ARM V4 指令集架构(instruction set architecture,ISA)的处理器模型 ARMChisel,该处理器模型采用新型的硬件语言 Chisel 构建,是一个具有高复杂性的硬件设计。基于这一嵌入式处理器模型:①设计了支持 ARM V4 ISA 架构全部指令的随机指令生成器,提高了生成测试激励的速度;②根据新型构建语言 Chisel 的特点,针对被测处理器模型设计了 Chisel 层面初级验证、覆盖率快速验证、直接测试验证和复杂应用程序验证策略,确保达到预期的覆盖率;③在 Chisel 环境和 Verilog 环境中搭建了基于嵌入式处理器模型的测试平台,测试平台收集覆盖率同时能快速准确地发现错误并定位错误,提高了验证速度。采用 FPGA(field programmable gute array)方法加速大型应用程序的验证,缩短了验证周期。

关 键 词:Chisel;处理器模型验证;ARM 架构;指令生成器;测试激励

中图分类号:TP368.1 **文献标志码**:A **文章编号**:1000-2758(2023)05-1024-09

近几十年以来,航空计算机技术飞速发展,嵌入式系统由于体积小、可靠性强以及功耗低等优点,被广泛应用于航空电子设备中。然而随着航空嵌入式领域应用程序需求的不断提高以及半导体工艺的快速发展,功能验证已经成为嵌入式系统设计周期中最主要的挑战。为了缩短整个处理器设计完成的总体时间,有必要研究出高速寻找并定位设计错误的验证方法,提高整个验证的速度,这对航空嵌入式处理器设计以及验证具有重大意义。

Chisel[1]是 Berkeley 大学提出的一种嵌入在 Scala[2]中(一种高级汇编语言)的硬件构造语言,使得设计者可以用 Scala 这种函数式编程语言快速设计硬件,并能转化成 Verilog 代码用于进一步实现。Chisel 语言相比较传统硬件描述语言 Verilog 的优点主要有:①抽象程度高、高度参数化;②前端设计周期较短;③除了可以生成 Verilog,还可以生成 C 或者 CPP;④代码可综合。因此,目前已有一些基于 Chisel 语言构建的嵌入式处理器[3]。本文采用以 ARM9TDMI 为目标构建的航空处理器模型 ARMChisel 进行研究,采用经典的 5 级流水线设计,支持 7 种处理器模式及 7 种中断类型。由于航空处理器模型对于精确控制的高要求使得该处理器模型的数据通路和控制通路非常庞大与复杂。此外,基于 Chisel 语言的模型验证框架不像基于 Verilog 和 SystemVerilog 的验证那样具有系统性,这使得基于 Chisel 语言构建的处理器模型验证难度和验证周期大大增加。

目前,主要使用基于仿真的验证方法验证嵌入式系统的功能。这种验证方法[4]借助仿真工具通过模拟实际电路的工作环境对处理器系统进行验证,通常要仿真大量的测试得到期望的覆盖率。目前已有一些工作针对嵌入式处理器进行了功能验证。文献[5]提出了一种自动化的覆盖率驱动验证和优化方法,验证了算术逻辑单元和 RISC 处理器

收稿日期:2022-12-01

作者简介:吴乐宁(1994—),航空工业西安航空计算技术研究所助理工程师,主要从事机载嵌入式软件研究。

通信作者:陈福(1984—),航空工业西安航空计算技术研究所高级工程师,主要从事机载嵌入式软件研究。

e-mail:cfu12435@163.com

的性能。文献[6]设计了一个混合模型作为参考模型,通过比对两者的输出从而定位待测设计的错误以验证多核处理器。文献[7]基于龙芯 2K100B 移动智能终端制定了相关的测试方案。文献[8]针对 X86 处理器阐述了通用的验证环境,并提出了基于寄存器和多级缓存的验证原则。文献[9]基于覆盖率引导的变异模糊测试提出了一种采用最少的设置并利用 FPGA 加速仿真测试的新方法来解决测试覆盖率问题。然而这些研究都没有面向 Chisel 语言进行研究,不能为现有基于 Chisel 语言构建的航空处理器提供功能验证指导。文献[10]基于嵌入式 Scala 的 Chisel 语言开发了名为 ChiselVerify 的高级验证库,为 Chisel 生态系统带来了功能覆盖率等功能。然而仅仅针对 Chisel 设计增加了验证库,并没有为整个 Chisel 设计提供验证方法。

目前对基于新型硬件语言 Chisel 构建的处理器模型的验证策略研究还比较少。因此,本文提出基于 ARMChisel 嵌入式处理器模型研究的功能验证加速方法。提出了一个覆盖率感知且支持中断测试的随机指令生成器,可以生成大量高质量测试激励,提升随机验证效率;给出了基于 Chisel 设计的分阶段验证策略,加快了验证周期,为这一方向的研究提供指导;基于 Chisel 和 Verilog 设计搭建测试平台,并成功启动操作系统。

1　覆盖率感知的指令生成器

测试激励的质量高低是影响基于仿真验证的一个重要因素。在讨论随机测试激励的生成之前首先分析被测处理器模型的指令集架构。

1.1　指令集架构分析

如图 1 所示,ARM 指令按指令特点可以分为以下几种[10]:数据处理类指令、存储器访问类指令、乘法类指令、跳转指令、杂项指令、协处理器指令。该嵌入式处理器模型,支持数据处理类指令 57 条、存储器访问类指令 28 条、乘法指令、跳转指令和杂项指令共 15 条以及一些系统级的指令。

ARM V4 指令集支持的指令表示的基本格式为〈opcode〉{s}〈c〉〈Rd〉,〈Rn〉{,〈operand2〉}。

其中,{ }中的项目可以省略,而〈 〉中的项目不可省略,opcode 是每条指令的指令助记符,不可省略;s 是每条指令是否要根据指令执行结果改写状态寄存器的标志;c 是每条指令的指令执行条件;Rd

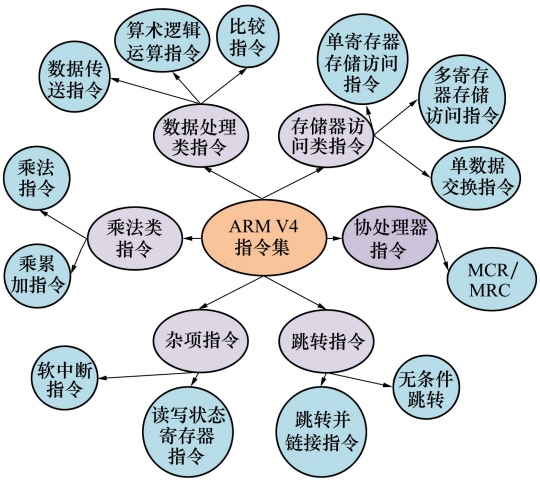

图 1　ARM V4 指令集指令分类情况

一般是目的寄存器,用来存放指令计算得到的值;Rn 一般是第一操作数的寄存器或者基址寄存器;operand2 一般是第二个操作数,第二个操作数的生成有 3 种方式:①8 位立即数循环移位偶数位;②Rm 寄存器的方式;③Rm,shift 类的寄存器移位。

随机指令生成器要根据指令集的分类以及编码规则建立它的生成机制。

1.2　随机指令生成器

基于分析得到的生成机制,设计了一个受约束的随机指令生成器,该生成器具备 5 个功能:①生成的指令符合被测处理器模型的指令集规范,是合法的指令;②支持 ARM V4 指令集所有的指令;③该生成器对存储结构可控,能控制指令的寻址地址以及跳转地址,以防跳转到非法的存储区域,造成仿真错误;④支持 7 种处理器运行模式;⑤支持 7 种中断处理和中断返回。

图 2 为随机指令生成器结构图。

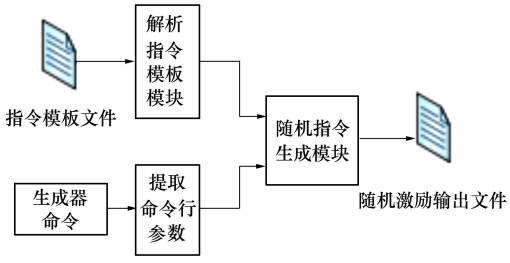

图 2　随机指令生成器结构

该随机指令生成器主要由四部分组成:指令模板文件、解析指令模板模块、提取命令行参数模块、随机指令生成模块。解析指令模板模块用于从根据

指令集规范提前编辑好的指令模板文件中逐行读取指令信息,按规则提取出各个域信息并将信息存进指令池中。提取命令行参数模块用于从输入的命令中提取出随机指令生成器的约束,其中约束包括:要生成的指令、要生成的指令数量和生成无条件执行指令的概率。然后将约束和指令池的信息作为参数传送给随机指令生成模块,最后随机指令生成模块根据得到的约束生成随机指令序列并写入二进制输出文件中。

1.2.1 指令模板文件

指令模板文件用于提供给随机指令生成模块关于 ARM V4 指令集的指令编码相关信息,使得指令生成模块可以根据这个文件生成有效合法的指令序列。

指令模板各域的意义为:①insnname 表示一个指令类的名称。②cond:4 表示 ARM V4 指令的高 4 位(条件码),属于可随机生成的域。③fixedvalue 表示指令中固定值的部分,属于不可随机生成的域。④randvar:randwidth 表示一个宽度为 randwidth 名为 randvar 的可随机生成域。⑤constraints 表示该指令类的约束内容。比如,一部分指令的目的寄存器不可以为 15 号寄存器(PC)。⑥memory 表示该类指令的寻址模式。

其中可随机生成域和不可随机生成域可根据指令实际情况增删。约束域和存储域若没有可以省略。

1.2.2 随机指令生成模块

随机指令生成模块用于生成随机指令并写入二进制输出文件中。首先读入要生成指令的信息和约束,然后设置好存储器的大小,接着写入中断处理指令序列,最后按照随机数随机到指定数量的指令并写入一个结束指令(一条特定的未定义指令)即可。

针对约束访存地址和异常中断处理功能描述如下:约束访存地址:在随机生成指令之前先分配好存储器的大小以及地址,后面生成访存指令时根据设置的数据存储器大小和地址对随机到的访存地址加以约束,使得访存地址都是有意义的地址。异常中断处理:随机指令生成器生成的指令序列希望发生中断时,能不影响程序正常执行,因此对中断处理进行了简化。生成器在生成指令之前要写入二进制输出文件的简化中断处理程序来支持中断返回。

支持中断测试的处理机制:

0x00000000: BL INST_ADDR;
0x00000004: LDR PC,[PC,#20];
0x00000008: LDR PC,[PC,#16];
0x0000000C: LDR PC,[PC,#16];
0x00000010: LDR PC,[PC,#12];
0x00000014: MOV R0,R0;
0x00000018: LDR PC,[PC,#4];
0x0000001C: LDR PC,[PC,#0];
0x00000020:ANDEQ R0,R0,R8,LSR
0x00000024: ANDEQ R0,R0,IP,LSR
0x00000028: SUBS PC,LR,#0;
0x0000002C: SUBS PC,LR,#4;

其中 INST_ADDR 为随机指令序列的第一条指令的地址,即发生复位时跳转到第一条指令的地址重新执行。最终会开始执行中断返回指令 SUBS PC,LR,#0,该指令将 PC 值改写为发生中断时的指令的下一条指令地址。其他中断情况下和软中断和未定义指令中断情况类似,唯一不同的是中断处理指令为 SUBS PC,LR,#4,该指令将 PC 值改写为发生中断时的指令地址。

该指令生成器是一个可以根据验证需求配置的高效伪随机指令生成器。使用随机指令生成器可以提高生成测试激励的速度,加速整个验证过程。

2 嵌入式处理器模型的阶段验证

根据被测设计(一个新型构建语言 Chisel 构建的兼容 ARM V4 的嵌入式处理器模型 ARMChisel)的特点,制定了如图 3 所示的分阶段验证策略。

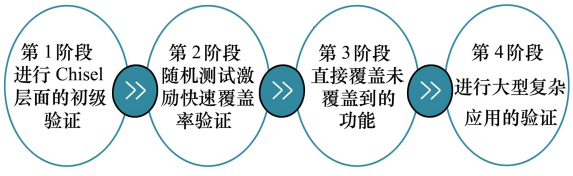

图 3 4 个验证阶段

2.1 第 1 阶段:Chisel 层面初级验证

在整个验证过程中,由于整个处理器模型的设计均使用 Chisel 语言构建,因此第 1 阶段的验证是基于 Chisel 语言下的验证。之所以设置了 Chisel 层面的初级验证,而不直接转为 Verilog 进行验证,是因为由 Chisel 设计转化为的 Verilog 设计可读性较差,验证时定位错误后难以修改代码,因此必须在 Chisel 层面进行初级验证。

Chisel 提供的验证库有 3 种可用的 tester[11],分别为 PeekPokeTester, SteppedHWIOTTester, OrderedDecouoledHWIOTester。其中 PeekPokeTester 是最灵活的一个 tester,它提供了如表 1 所示的几个操作用于测试验证,并结合目前的开源 Chisel 验证库[11]进行第 1 阶段的功能点验证。

表 1　Chisel 验证操作

tester	功能
poke(c.io.in,value_in)	设置端口 in 的值为 value_in
peek(c.io.out)	读取端口 out 的值
expect(c.io.out,value_right)	判断端口 out 是否为值 value_right
step(1)	推进一个时钟周期

其中,poke( ) 用于设置被测设计输入端口的值,peek( ) 用于从被测设计输出端口读取输出结果,expect( ) 用于判断端口的输出值和特定值是否相等,step( ) 用来推进被测设计的时钟。

第 1 阶段主要进行简单逻辑以及数据通路、控制通路的验证,充分利用 Chisel 提供的验证支持,排查出一些逻辑错误。Chisel 语言层面进行测试的平台搭建如图 4 所示。

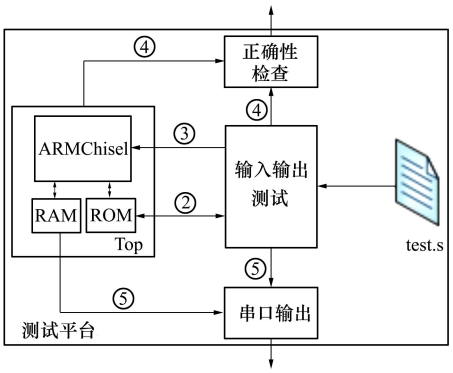

图 4　Chisel 层面测试平台

测试平台由 5 个部分组成:Chisel 测试类、被测设计、测试激励、串口输出、正确性检查。

首先从测试激励文件 test.s 读取指令,并把指令写入指令存储器(ROM)中去,然后开始仿真并进行正确性检查以及串口输出。其中采用测试激励自校验和参考模型比较法进行结果正确性检查。为了可见手写应用程序和标准程序集中的输出结果,在 Chisel 层面的测试平台中模拟串口输出,用于可视化应用程序的验证。

Chisel 并没有提供直接的生成覆盖率信息工具,但是可以在构建 Chisel 的工具 SBT 中构建指令 build.sbt 增加用于生成测试覆盖率信息的插件 sbt-scoverage 获得被测设计的覆盖率。根据需要设置 coverageEnabled, coverageMinimum 和coverageFailOnMinimum 等一系列覆盖率配置参数。调用命令执行即可得到各模块的覆盖率信息。

虽然在 Chisel 语言层面可以进行一些简单功能验证和代码覆盖率的收集,但是缺乏 EDA 工具的支持使得大型设计的验证仍然比较困难。因此考虑将被测的嵌入式处理器模型转化为 Verilog 语言继续进行后续阶段的验证。

2.2　第 2 阶段:覆盖率快速验证

第 2 阶段要进行覆盖率快速验证,使用随机指令生成器生成大量的随机测试激励,尽可能覆盖到被测模型的所有功能点,以便快速达到较高的覆盖率。

在验证之前,首先要基于被测设计提取出功能点。对于被测模型 ARMChisel,共提取出 135 个功能点,其中单条指令功能点 100 条,组合指令功能点 13 条,特殊情况功能点 22 条。

这一阶段需要借助 EDA 工具提供的一些功能,因此搭建基于 Verilog 语言的测试平台,如图 5 所示。测试平台由五部分组成:被测处理器模型、测试激励、正确性检查、覆盖率检查和串口输出。

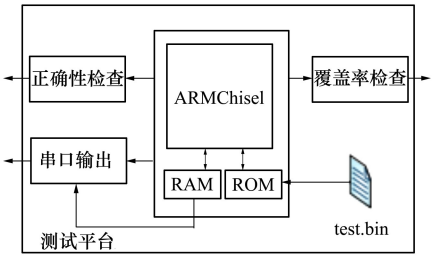

图 5　Verilog 层面测试平台

首先从测试激励文件(test.bin)中读指令到指令存储器(ROM)中,然后开始仿真验证被测设计。在验证过程中,会进行执行结果正确性检查和覆盖率检查,可以快速地发现错误并定位错误。其中采用测试激励自校验法和参考模型比较法来进行结果正确性检查,采用代码和功能双覆盖率来进行覆盖率检查。在 Verilog 层面的测试平台中也增加了串口输出的模块,用于可视化应用程序的验证。

2.3 第 3 阶段:直接测试验证

第 3 个阶段验证的目标是使整个设计的功能覆盖率达到 100%,包括一些边界情况和特殊情况,该阶段的验证步骤为:

1) 在前 2 个阶段验证的前提下,会得到一个较高的功能覆盖率报告,分析该报告得到功能覆盖率的覆盖情况;

2) 将未覆盖功能点全部罗列出来,然后针对这些功能点,生成特定测试激励,一般 1 个功能点 1 个测试激励;

3) 仿真运行这些直接测试激励或者随机测试激励,得到新的覆盖率报告;

4) 分析得到的新覆盖率报告,查看是否达到 100%的覆盖率,如果达到了期望覆盖率则本阶段验证结束,进入第 4 个阶段的验证。如果没有达到期望覆盖率则继续执行步骤 1)~4),直到达到期望的覆盖率。

在本阶段,对于一些可使用带约束的随机指令生成器生成的测试激励验证的功能点,可以用随机指令生成器来快速生成测试激励。

2.4 第 4 阶段:复杂应用程序验证

通过前 3 个阶段的验证,行覆盖率以及功能覆盖率均达到了 100%,这意味着验证流程已经接近尾声。为了验证被测设计处理复杂情况的能力,选择一些复杂应用程序来进行最后一个阶段的验证。

表 2 为第 4 阶段选取的一些应用程序,通过仿真运行这些应用程序,可以测试处理器模型在复杂情况和指令组合下的处理能力。

表 2 复杂应用程序

复杂应用程序	功能
Mibench[11-12]	排序,数据运算,字符处理的程序嵌入式标准程序集
Dhrystone benchmark[13]	处理器性能测试程序
uClinux[14]	轻量级嵌入式操作系统

3 实验结果及分析

进行初级验证以便排除低级错误,考虑到 Chisel 的验证能力,在正确运行如表 3 所示的一些手写汇编程序后,即可进入第 2 阶段的验证。

表 3 部分手写汇编程序

汇编程序	功能
dp0.s/dp1.s/dp2.s/	数据处理类指令(无冒险)
dp0_h.s/dp1_h.s/dp2_h.s	数据处理类指令(有冒险)
multest.s	乘法操作类指令(无冒险)
multest_h.s	乘法操作类指令(有冒险)
str.s/strb.s/strh.s	单存储器访问类指令(无冒险)
str_h.s/strb_h.s/strh_h.s	单存储器访问类指令(有冒险)
ldmtest.s	多存储器访问类指令(无冒险)
ldmtest_h.s	多存储器访问类指令(有冒险)
msrtest.s	读写状态寄存器类指令(无冒险)
msrtest_h.s	读写状态寄存器类指令(有冒险)
btest.s/switest.s	跳转和软中断类指令(无冒险)
btest_h.s/swiftest_h.s	跳转和软中断类指令(有冒险)

手写的汇编程序涵盖了指令集中的所有指令,分为有冒险和无冒险的指令,用来测试被测处理器模型数据通路的控制通路。

3.1 第 2 阶段验证结果分析

第 2 阶段是尽快达到较高覆盖率的一个验证阶段。基于被测处理器一共提取功能点 135 个。为了研究随机指令数与覆盖率的关系,进行了多次仿真实验,仿真了第 25~700 条指令,实验所得数据如表 4 所示。

从表中可以看出当指令数达到 575 时,单条指令类的功能点数就可以覆盖到 98。功能点数无法达到 100 是因为在随机时没有随机 2 条更改状态寄存器的指令。如果随意更改状态寄存器的值就会导致处理器处于未知甚至非法的运行模式,从而发生错误。因此这 2 条指令需在第 3 阶段验证。组合指令类的功能点和特殊情况类的功能点则很难达到全覆盖验证。

为了研究分析第 2 阶段的验证何时转入第 3 阶段进行,统计了验证过程中随机指令数的大小和覆盖率的关系,结果如图 6 所示。

表 4　随机测试激励覆盖情况

随机指令个数	单条指令	组合指令	特殊情况	总覆盖情况	覆盖率/%
25	32	11	3	46	34.1
50	51	12	3	66	48.9
75	61	10	4	75	55.6
100	68	11	5	84	62.2
125	74	12	5	91	67.4
150	81	12	5	98	72.6
175	84	12	5	101	74.8
200	88	11	6	105	77.8
225	90	12	6	108	80.0
250	92	12	7	111	81.5
275	92	11	7	110	81.5
300	95	12	8	115	85.2
325	95	12	7	114	84.4
350	94	12	11	117	86.7
375	93	12	9	114	84.4
400	94	12	13	119	88.1
425	93	12	9	114	84.4
450	96	12	10	118	87.4
475	95	12	10	117	86.7
500	97	12	12	121	89.6
525	95	12	13	120	88.9
550	97	12	11	120	88.9
575	98	12	9	119	88.1
600	98	12	9	120	88.9
625	99	12	10	121	89.6
650	98	12	12	122	90.4
675	98	12	11	121	89.6
700	98	12	10	120	88.9

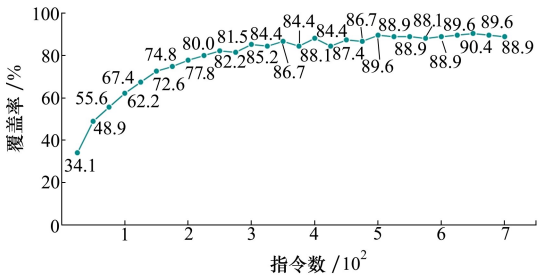

图 6　随机指令数与覆盖率关系图

图中统计了随机指令数从 25~700,所达到的功能覆盖率情况。从图中可以看出当指令数小于等于 250 时,覆盖率持续上升,而当指令数超过 250 之后,覆盖率开始波动性上升,超过了 80%。指令数在 500~700 之间,覆盖率并没有明显提升。可以看到,在指令数为 650 时达到最高覆盖率 90.4%。

指数令超过 250 后会有波动性的上升,是因为设计的随机指令生成器的随机数种子由时间函数产生,因此每次生成的随机测试激励都是具有独立性的随机。当生成的指令数较大时,就会产生这种波动式增长。

在第 2 阶段的验证中,当覆盖率达到 90%左右时,再增加指令数获得的覆盖率增益微乎其微。因此,停止第 2 阶段的验证,进入第 3 阶段的验证。

3.2　第 3 阶段验证结果分析

在完成第 2 阶段的验证之后,行覆盖率和功能覆盖率均处于较高的水平。第 3 个阶段主要通过分析覆盖率报告有目的性地设计测试激励,覆盖未覆盖到的所有功能点。表 5 列举了在结束第 2 阶段的验证之后得到的一些代表性的未覆盖到的功能点。

表 5　未覆盖到功能点列举

类型	未覆盖到的功能点	可能的原因
单条指令	MSR reg	会影响处理器执行,因此未随机生成该指令
组合指令	比较再分支	未生成该组合指令类型
特殊情况	ADD sp plus imm	该指令一操作数为 sp,未生成指定域为 sp 的 ADD 指令
特殊情况	LDM (exception return)	该指令未生成用于中断返回的 LDM 指令,未生成
特殊情况	STM(user registers)	未生成用户模式下的 STM 指令
特殊情况	LDRB literal	该指令基址寄存器为 pc,未生成指定域为 pc 的 LDRB 指令
特殊情况	LDRSH literal	该指令基址寄存器为 pc,未生成指定域为 pc 的 LDRSH 指令
特殊情况	MSR reg sys	由于伪随机 MSR reg 导致的,未生成该指令

从表中可以看出,单条指令类、组合指令类和特殊情况类的功能点均有未覆盖到的,而大多数为特殊情况功能点。

重复性地分析覆盖率报告并增加测试激励,直到代码覆盖率和功能覆盖率达到 100%,即可进入下一阶段的验证。

3.3 第 4 阶段验证结果分析

第 4 个阶段主要是进行一些复杂应用程序的验证。stringsearch_ small 是嵌入式标准程序集 Mibench 的一个字符查找程序。该程序在处理器模型上通过模拟的串口输出 57 行字符,与 Mibench 提供的 stringsearch_small 程序的正确输出完全相同。

在嵌入式处理器中,Dhrystone Benchmark 是一种经常使用的测试标准。图 7 为运行 Dhrystone 2.1 的仿真截图。

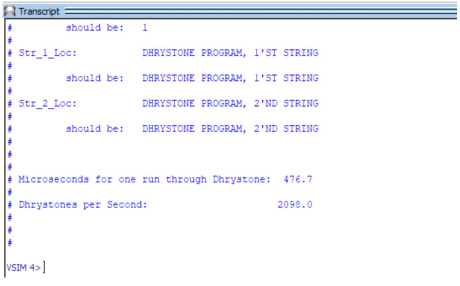

图 7 Dhrystone 仿真图

从图中可以看到每秒钟能执行 Dhrystone Benchmark 程序 2 098 次。

除了上述测试激励外,还使用了嵌入式操作系统 uClinux 进行验证。uClinux 的启动分为 2 个阶段[15],第 1 个阶段是 bootloader 的启动阶段。第 2 阶段是对内核进行初始化和启动的阶段。

仿真之前,首先要设置好启动过程中需要访问的寄存器值。其次使用 initial 语句产生 IRQ 中断并分配好存储器。接着就可以仿真真实的硬件环境,在 Modelsim 中启动操作系统。图 8 为在 Modelsim 中仿真运行操作系统的运行截图。

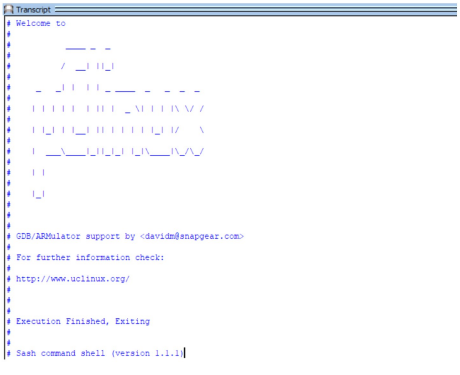

图 8 uClinux 操作系统仿真图

启动过程需要在 Modelsim 中仿真 90 ms,其中时钟周期为 10 ns。启动 uClinux 操作系统仿真了 900 万次时钟,一共执行了 4 063 341 条指令。

3.4 基于 FPGA 的验证结果分析

在进行大型程序仿真时选用基于 FPGA 的验证方法进行仿真验证。硬件环境为:FPGA 型号 Xilinx Artix-35T FPGA(xc7a35ticsg324-1L),处理器 Intel Core i5-3230M CPU。软件环境为:Windows 10 专业版,64 位操作系,Vivado 2016.4,Tera Term 串口软件。

基于 FPGA 搭建的验证系统,如图 9 所示。

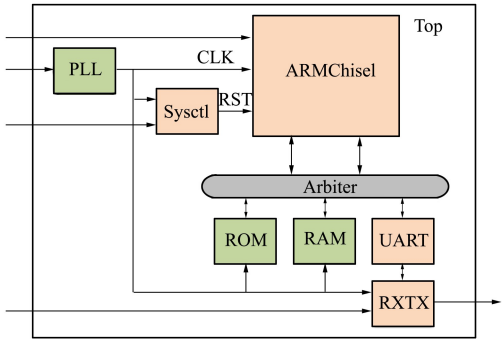

图 9 片上系统结构图

图中 PLL、ROM、RAM 均使用 Xilinx 提供的 IP 核构建。ARMChisel、Sysctl、Arbiter、UART、RXTX 分别为使用硬件语言描述的功能模块。

在前期验证充分的情况下,使用 Vivado 提供的逻辑分析仪(integrated logic analyzer,ILA)核,可以使用网表插入调试探测的方法进行验证,并分析调试结果。

图 10 为 FPGA 加速验证的连接图。Arty 开发板正在加速仿真 Mibench 标准程序集的 basicmath_small 测试程序。

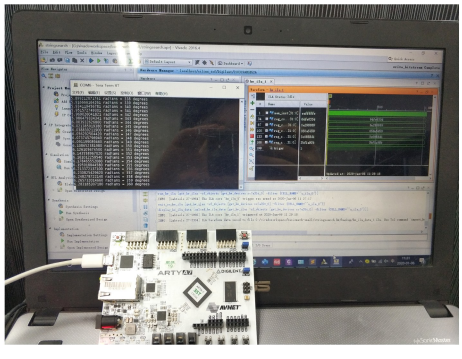

图 10 FPGA 加速验证连接图

从捕获到的波形图上可以得到,ARMChisel 处理器模型运行该程序用了 1 694 387 630 个时钟周期,共执行了 1 007 179 985 条指令。串口软件输出为加速仿真 basicmath_small 测试程序时 Tera Term 串口软件的输出。将串口输出的内容与 Mibench 提供的正确输出结果进行比较,共 19 733 行输出,与正确输出完全相同,ARMChisel 处理器模型均运行正确。

另外,该过程还验证 Mibench 的一系列标准测试程序。串口软件的输出结果均与 Mibench 程序集正确输出结果相同。

使用 FPGA 进行加速验证可以加快程序仿真速度,然而相比于基于仿真的验证,调试信号的可视性较低,定位错误能力低,潜在延长了设计验证时间。因此,基于硬件的加速验证适合后期加速大型程序的验证。通过加速验证这些标准程序集的程序,进一步提高了验证效率。

4　结　论

本文研究了由新型硬件语言 Chisel 构建的兼容 ARM V4 架构的嵌入式处理器模型的验证方法。主要得到 3 个结论:①设计了支持 ARM V4 ISA 全部指令的随机指令生成器。可生成大量的测试激励用于随机测试,加快了生成激励的速度。②被测处理器模型由新型硬件语言构建,针对这一特点设计了 4 个阶段的验证:Chisel 层面初级验证、覆盖率快速验证、直接测试验证和复杂应用程序验证,并分析了各阶段转换的时机,确保了预期的覆盖率要求。③搭建了基于 Chisel 语言层面的测试平台,研究了在 Chisel 层面进行正确性验证和覆盖率收集的方法。由于 Chisel 验证的局限性,搭建了基于 Verilog 语言层面的测试平台进行验证。测试平台收集覆盖率同时能快速准确地发现错误并定位错误,提高了验证速度。

参考文献:

[1] BACHRACH Jonathan, VO Huy, RICHARDS Brian, et al. Chisel: constructing hardware in a scala embedded language[C]// Design Automation Conference, 2012

[2] CHIUSANO P, BJARNASON R. Functional programming in Scala[M]. Greenmch: Greenmch Manning Press, 2014

[3] CELIO C, CHIU P F, ASANOVIC K, et al. BROOM: an open-source out-of-order processor with resilient low-voltage operation in 28 nm CMOS[J]. IEEE Micro, 2019, 99(1): 1

[4] ALIMI N, LAHBIB Y, MACHHOUT M, et al. Simulation-based verification of large-integer arithmetic circuits[C]//2016 1st IEEE International Verification and Security Workshop, 2016

[5] MARCELA Imková, ZDENEK Kotásek. Automation and optimization of coverage-driven verification[C]// Euromicro Conference on Digital System Design, 2015

[6] 王树朋. 基于仿真的多核处理器功能验证技术研究[D]. 杭州:浙江大学, 2017
WANG Shupeng. Research on simulation-based multi-core processor function verification technology[D]. Hangzhou: Zhejiang University, 2017 (in Chinese)

[7] 赵康. 面向龙芯 2K1000B 处理器的系统级功耗优化与验证[D]. 南京:东南大学, 2020
ZHAO Kang. System-level power optimization and verification for Godson 2K1000B processor[D]. Nanjing: Southeast University, 2020 (in Chinese)

[8] 陈国华. 乱序超标量多核处理器的验证方法[J]. 集成电路应用, 2019, 36(9): 3
CHEN Guohua. Verification method for super scale multi-core processor[J]. Applications of IC, 2019 (in Chinese)

[9] LAEUFER K, KOENIG J, KIM D, et al. RFUZZ: coverage-directed fuzz testing of RTL on FPGAs[C]//IEEE/ACM International Conference on Compater Aided Design, 2018

[10] DOBIS A. Towards functional coverage-driven fuzzing for Chisel designs[C]//Workshop on Open-Source EDA Technology, 2021

[11] Suereth, Joshua D. Sbt in action: the simple Scala build tool[M]. Greenwick: Greenwick Manning Publications Co, 2016

[12] GUTHAUS M R, RINGENBERG J S, ERNST D, et al. MiBench: a free, commercially representative embedded benchmark suite[C]//IEEE International Workshop on Workload Characterization, 2001

[13] WEICKER R P. Dhrystone: a synthetic systems programming benchmark[J]. Communications of the ACM, 1984, 27(10):

1013-1030

[14] 周建胜, 谷勇, 刘文权. 一种开放源代码的嵌入式操作系统-uCLinux[J]. 中国科技信息, 2005(19): 82
ZHOU Jiansheng, GU Yong, LIU Wenquan. An open source embedded operating system-uCLinux[J]. China Science and Technology Information, 2005(19): 82 (in Chinese)

[15] CHEN J, FAN Y, RAN L, et al. uClinux transplantion based on ARM7TDMI[J]. Journal of Electron Devices, 2003(3): 12-13

Research on functional verification method processor model built by Chisel

WU Lening[1], WANG Miao[2], CHEN Fu[1]

(1.AVIC Xi'an Aeronautics Computing Technique Research Institute, Xi'an 710076, China;
2.School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: With the increasing complexity of hardware design, verification has become the difficulty of chip design. In order to effectively shorten the overall working time of the design process, it is necessary to work out a method to quickly find design errors in the verification that takes up a lot of time in the design. The design under test is an ARM Chisel compatible with the ARM V4 instruction set architecture (ISA) processor model. The processor model is built with a new hardware language Chisel and is a highly complex hardware design. Based on this embedded processor model, ①a random instruction generator supporting all instructions of the ARM V4 ISA architecture is designed to increase the speed of generating test stimuli; ②based on the characteristics of the new construction language Chisel, designed for the processor model under test four verification stages: primary verification at the Chisel level, rapid verification of coverage, direct test verification and verification of complex applications, to ensure that the expected coverage is achieved; ③built in the Chisel environment and Verilog environment based on the embedded processor model Test platform. The test platform can quickly and accurately find errors and locate errors while collecting coverage, which improves the verification speed. Finally, the FPGA acceleration method is used to accelerate the verification of large-scale application programs and shorten the verification cycle.

Keywords: Chisel; processor model validation; ARM architecture; instruction generator; test stimulus

引用格式: *吴乐宁, 王淼, 陈福. Chisel 构建处理器模型的功能验证方法研究[J]. 西北工业大学学报, 2023, 41(5): 1024-1032*

WU Lening, WANG Miao, CHEN Fu. Research on functional verification method processor model built by Chisel[J]. Journal of Northwestern Polytechnical University, 2023, 41(5): 1024-1032 (in Chinese)