第 34 卷 第 4 期 计 算 机 工 程 2008 年 2 月
Vol.34 No.4 Computer Engineering February 2008
·软件技术与数据库· 文章编号：1000—3428(2008)04—0098—03 文献标识码：A 中图分类号：TP311.5

基于模拟器的嵌入式软件动态测试技术

许 福，金茂忠，晏海华，刘 辉，赫建营
（北京航空航天大学软件工程研究所，北京 100083）

摘 要：提出一种嵌入式软件动态测试框架，该框架以模拟器为运行核心，准确模拟嵌入式硬件的运行状态，给出语句覆盖率、分支覆盖率、函数及模块的执行频度、程序执行时间等测试信息，有效地支持了嵌入式软件的动态测试和单元测试。实验证明了该方法的有效性。

关键词：模拟器；嵌入式软件测试；动态测试；单元测试

Dynamic Test Framework for Embedded Software Systems Based on Simulator

XU Fu, JIN Mao-zhong, YAN Hai-hua, LIU Hui, HE Jian-ying
(Software Engineering Institute, Beijing University of Aeronautics and Astronautics, Beijing 100083)

【Abstract】 A dynamic test method for embedded software systems is presented. This method is based on software simulator and can simulate the embedded hardware accurately. It improves dynamic test and unit test, which reports the coverage of statements, the coverage of branches, the calling frequency of functions/modules and the execution clocks. Experimental results show that the technology is effective.

【Key words】 simulator; embedded software test; dynamic test; unit test

1 概述

在嵌入式软件测试中，对实时、嵌入的要求增加了软件测试的复杂性[1]。嵌入式软件测试具有以下特征：

(1)软件的运行环境和开发环境不一致。

(2)软件具有很强的专用性，只能在特定系统下工作。

(3)软件与硬件联系紧密，在不具备硬件的条件下，难以进行系统级的综合测试。

(4)受空间和时间的约束，传统软件测试中的插装技术不能满足灵敏性、实时性及内存空间等方面的要求。

目前采用的嵌入式软件动态测试方法可分为：

(1)在线模拟器。通过硬件完成对处理器的仿真，无需更改被测程序，且测量的执行时间相当准确，但价格昂贵，且不是所有处理器都具备在线模拟器。

(2)指令模拟器。通过软件完成对处理器的仿真，可以得到所执行程序的精确执行时间和运行轨迹，且不需要更改被测程序。

(3)插装工具。在程序中插入探针来记录执行时间和运行轨迹。需更改被测程序，测得的执行时间不准确，甚至会影响程序的执行结果。

20 世纪 70 年代，国外就开始了对嵌入式软件的测试技术研究。1980 年R L Class发表了著名的：The “Lost World” of Software Debugging and Testing[2]，指出了嵌入式软件测试落后于通用软件测试的现状。此后 20 多年间，国外在嵌入式软件测试领域进行了大量的研究工作，并推出了一些嵌入式软件测试工具，具有代表性的有：Lauterbach公司的Trace32-ICE(属第 1 类)，AMC公司的CodeTest，VERILOG公司的LogiScope，IPL公司的Cantata和ATTOL公司的Coverage等(属第 3 类)[3-7]。国内对嵌入式软件测试技术的研究开始于 20 世纪 90 年代前后，具有代表性的有 204 所和总装备部软件测试中心针对汇编语言的嵌入式软件测试工具(属第 2 类)，631 所等单位也研制过一些嵌入式软件测试工具，北航软件工程研究所研制的QESuite/QESat系列软件也可以用于嵌入式软件的测试(属第 3 类)[8-11]。

嵌入式系统广泛采用软硬件协同开发方法，在软件开发过程中，目标机环境很有限，甚至不可用，无法满足软件测试的需要。此外，现有的测试工具还缺乏通用性，往往只能对特定软件系统进行测试，大多采用插装方法，导致测试结果不准确，甚至无法进行测试。由于目标机系统的专用性，目标机环境和相关工具往往非常昂贵，因此无论从经济性还是开发效率方面考虑，应把测试工作尽可能多地放在宿主机上进行。

目标机和宿主机在体系结构上往往有很大差异，目标机软件一般不能在宿主机上直接运行，要在宿主机平台上进行测试需要建立目标机仿真环境。利用软件仿真进行测试具有如下优点：

(1)可以在没有目标机硬件的情况下，支持黑盒测试，此外它还支持白盒测试，使测试活动更加充分。

(2)可以精确记录程序的运行轨迹，可以准确计算出程序运行时间等信息。

(3)支持软硬件的协同开发，在硬件开发的同时，可以进行软件的开发和测试工作，可有效缩短系统开发周期。

(4)可使程序错误易于修正，并可以分清软硬件各自的错误，避免因实际写入目标机 ROM 造成调试和测试困难，降

基金项目：国家自然科学基金资助项目(60573084)

作者简介：许 福(1979－)，男，博士研究生，主研方向：编译技术，软件测试；金茂忠，教授、博士生导师；晏海华，副教授；刘 辉、赫建营，博士研究生

收稿日期：2007-04-05 **E-mail：**xufu@buaa.edu.cn

低开发成本。

(5)测试人员可灵活控制程序的运行状态，并人为地改变运行条件和控制程序的启停等。

2 计算机模拟的常用技术

常用的计算机模拟技术主要有 4 种：

(1)解释模拟(interpreter)。解释型模拟器的工作方式类似于被模拟处理器的取指-译码-执行循环。解释型模拟器的执行流程见图 1。

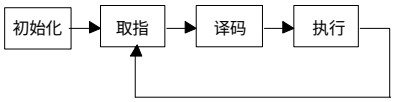

图 1 解释型模拟器的执行流程

(2)穿线代码(threaded code)。该方法产生一个列表，通过遍历该列表中用于完成每条指令模拟的子程序完成指令模拟。穿线代码模拟器的执行流程见图 2。

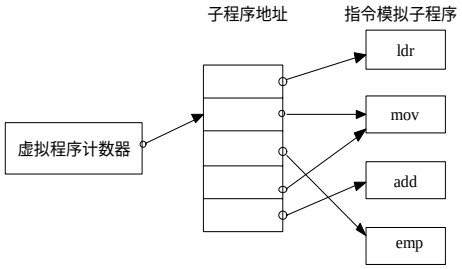

图 2 穿线代码模拟器的执行流程

(3)动态重编译(dynamic recompiler)。动态重编译模拟器每次执行时检查当前指令块是否已被模拟过，如果未被模拟过，则先生成本地代码，然后执行；否则，直接执行本地代码。动态重编译模拟器的执行流程见图 3。

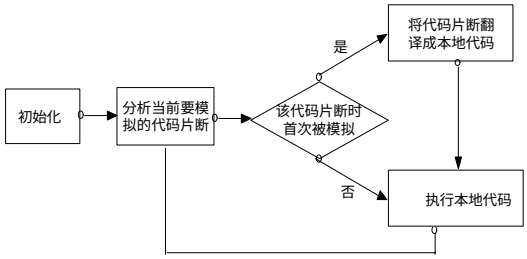

图 3 动态重编译模拟器的执行流程

(4)静态重编译(static recompiler)。静态重编译器事先生成所有代码。

表 1 给出了 4 种模拟方式的对比，从中可看出，采用解释模拟可较好地支持动态测试，尽管模拟速度比较慢，但实现相对简单，且模拟速度也基本能满足测试需求，因此，本文提出的动态测试框架 PowSim 采用解释模拟的方式执行。

表 1 4 种模拟方式的对比

执行速度	内存开销	可移植性		自修改代码
解释模拟	最慢	最小	好	支持
穿线代码	较慢	较小	好	困难
动态重编译	较快	较大	差	困难
静态重编译	最快	最大	差	不支持

3 PowSim 动态测试框架原理

图 4 给出了 PowSim 动态测试框架的处理流程，从中可看出，该框架包括以下几部分：

(1)编译/汇编源文件生成目标文件。要指定适当的编译参数，生成行号信息/段表信息。

(2)生成可执行文件(内存映像)。利用链接器将上述目标文件链接成可执行程序(内存映像)，链接器的选择有 2 种：

1)采用厂商或第三方链接器。实现简单，但被测模块(系统)必须完全通过链接才能执行，不能有效支持单元测试。

2)采用自己编写的链接器。支持对代码片断的测试，支持单元测试，但编码量较大。

(3)构建源文件代码行与可执行文件(内存映像)中地址的映射表。步骤包括：

1)对步骤(1)生成的目标文件进行分析，获得目标文件中的行号信息/段表信息等。

2)获得目标文件中各个段在可执行文件(内存映像)中的布局信息。如果采用第三方链接器，需要从生成的 Map 文件中获得该信息，如果采用自己编写的链接器可以自动获取该信息。

3)结合 A, B，就可以构建出源文件中行号与可执行文件(内存映像)中地址的映射关系表。

(4)模拟器加载步骤(2)生成的可执行程序(内存映像)及测试用例执行，得到程序的执行信息。

(5)生成动态测试信息报表。由于步骤(4)记录了程序的运行轨迹，因此可以统计出每个地址的执行次数，结合步骤(3)中构建的源文件行号与地址的映射表，统计出源文件中代码行的执行情况，并计算语句覆盖率等信息。

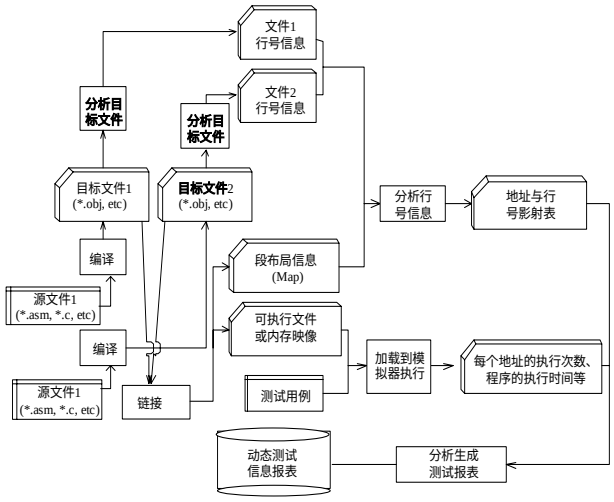

图 4 PowSim 动态测试框架的执行流程

4 实例研究

结合本文研制的两款动态测试工具，阐述 PowSim 动态测试框架的处理过程。

第 1 款工具是 1 套 Intel 8086 系列芯片的动态测试环境，采用第三方链接器。使用该工具执行动态测试的步骤如下：

(1)编译/汇编源文件生成目标文件。本工具支持两种语言：汇编语言和C语言，输出的目标文件格式为OMF(Object Module Format)[12]。

(2)调用OMF分析器，分析各个目标文件中的段定义(SEGDEF)和行号定义(LINNUM)。SEGDEF和LINNUM是TIS OMF Specification[12]中的两种标准记录格式，定义如表 2 和表 3 所示。

表 2 SEGDEF 标准记录格式

1	2	<var>	2/4	1/2	1/2	1/2	1
98/99	Len	Attr	SegLen	Index	Class	Overlay	ChkSum

表 3　LINNUM 标准记录格式

1	2	1/2	1/2	<var>	1/2
94/95	Len	BaseGrp	BaseSeg	DebugInfo	ChkSum

(3)通过第三方链接器 JLoc 对步骤(1)生成的各个目标文件进行链接，生成可执行文件(内存映像)，同时导出 Map 文件。Map 文件列出了各个目标文件中的各个段在可执行文件(内存映像)中的布局位置。表 4 给出了一个 Map 文件示例。通过分析 Map 文件，可获得各个目标文件中的各个段在可执行文件(内存映像)中的布局信息。

表 4　一个 Map 文件示例

Base	Start	Len	Seg	Group	Class	Module
F0000	F0000	21	_TEXT	null		startup.asm startup.obj
F0000	F0128	1B	_TEXT	null	CODE	fun1.c fun1.obj

(4)综合步骤(2)、步骤(3)，构建出源文件中的代码行与可执行文件(内存映像)中地址的映射表。

(5)模拟器加载步骤(3)生成的可执行文件(内存映像)和测试用例执行，记录程序的运行信息，如每个地址的执行次数，程序运行时间等。

(6)利用步骤(5)记录的每个地址执行次数信息，查询步骤(4)构建的行号与地址映射表，统计出语句覆盖率、分支覆盖率、函数和模块的执行频度等信息，生成测试信息报表。

基于 PowSim 动态测试框架开发的第 2 款工具是一套 ANALOG DEVICE 公司的 AD21060 芯片的测试环境。相比于 Intel 8086 芯片动态测试环境，该工具的特点是采用了自己编写的链接器而不是使用第三方链接器，支持在不具有完整代码的情况下执行测试，可有效地支持单元测试。

该工具的实施步骤与上述 8086 芯片动态测试工具的实施步骤基本一致，不同之处主要有：

(1)采用ANALOG DEVICE公司的编译器对程序进行编译，输出的目标文件格式是ELF(Executable and Linking Format)[13]。

(2)使用自己编写的链接器。由于链接过程完全由自己程序控制，因此可知道每个目标文件中每个段在最终生成的可执行程序(内存映像)中的布局信息，不用显式导出 Map 文件获得该信息。

(3)支持在只具有部分程序的情况下开始测试工作，有效地支持单元测试。单元测试一般都有许多输入/输出条件，并需要编写相应的桩函数，本工具采用灵活方便的脚本语言 Tcl/Tk 来完成这些工作，从而可以简化信息的查询/设置，并可支持测试用例的管理和批量执行。

5　结束语

本文提出了一种基于模拟器的嵌入式软件动态测试框架，该框架具有以下特点：

(1)不需要实际目标机环境即可对其上运行的嵌入式软件进行测试。

(2)代码覆盖率报表生成。支持生成语句覆盖率、分支覆盖率报表。

(3)程序执行频度分析。支持统计条件分支、多出口跳转等的执行频度，支持统计模块被动态调用的次数。

(4)中断的跟踪。可以识别中断函数、跟踪中断 I/O 和中断函数安装。

(5)断言的设定和跟踪。支持在源程序中设置断言，并对设定的断言进行动态跟踪。

(6)实时性能分析测试。支持对指定程序段和函数的执行时间进行计算和统计。

(7)基于 Tcl/Tk 的测试交互过程控制，有效地支持单元测试等。

本文提出的测试方法和测试工具已在航天科工集团四院十七所等多家单位获得了推广与应用。

参考文献

[1] 刘洪勋. 实时嵌入式软件仿真测试环境的设计和实现[D]. 北京: 北京航空航天大学, 2000.

[2] Robert L G. The “Lost World” of Software Debugging and Testing[J]. Communications of the ACM, 1980, 23(5): 264-271.

[3] Host/Target Testing[Z]. (2006-09-02). http://www.iplbath.com.

[4] nATTOL Coverage[Z]. (2006-09-02). http://www.attol.com.

[5] Isaksen U, Bowen J P. System and Software Safety in Critical Systems[EB/OL]. (1996-10-02). http://www.jpbowen.com/pub/scs-survey-tr97.pdf.

[6] Lauterbach-TRACE32 Microprocessor Development Tools, Emulators, Debuggers, Simulators[Z]. (2006-08-02). http://www.lauterbach.com.

[7] Microsystems Corporation. Code Test Embedded Software Verification Tools[EB/OL]. (2004-10-02). http://www.amc.com.

[8] 贺红卫. Intel8086 软件仿真器的设计与实现[J]. 系统仿真学报, 1996, 8(3): 50-55.

[9] 航天工业总公司. 嵌入式汇编语言(8086/8031)测试工具[Z]. 北京: [出版者不详], 1997.

[10] 王　璞, 张臻鉴. 基于覆盖的软件测试技术在实时嵌入式软件中的应用研究[J]. 计算机工程与设计, 1998, 19(6): 45-54.

[11] 北航软件工程研究所. QESuite/QESat 用户手册[Z]. 北京: [出版者不详], 2005.

[12] TIS Committee. Tool Interface Standard Relocatable Object Module Format(OMF) Specification[Z]. 1995.

[13] TIS Committee. Tool Interface Standard Executable and Linkable Format(ELF) Specification[Z]. 1999-06.

(上接第 97 页)

[3] The Open Service Gateway Initiative. OSGi Service Platform Specification Release 4[Z]. 2005.

[4] Chen G, Kotz D. A Survey of Context-aware Mobile Computing Research[R]. Dartmouth College, Department of Computer Science, Technical Report: 2000-381, 2000.

[5] Forgy C L. Rete: A Fast Algorithm for Many Pattern/many Object Pattern Matched Problem[J]. Artificial Intelligence, 1982, 19(1): 17-37.