嵌入式软件测试策略研究

李　伟　程朝辉
(北京市 7223 信箱 10 分箱　100072)

摘　要: 嵌入式实时系统的应用越来越广泛,其可靠性更加依赖于嵌入式软件的质量。对嵌入式软件进行测试是提高其质量的重要手段之一。本文主要研究了嵌入式软件的测试策略,分别讨论了基于主机的仿真环境下和基于目标机平台下如何测试嵌入式软件。

关键词: 嵌入式软件; 测试策略; 覆盖率测试; 性能测试
中图分类号: TP301.6

引　言

在信息技术迅速发展的今天,嵌入式系统的应用日趋复杂,开发技术日新月异,硬件发展的日益稳定,而软件故障却日益突出,其质量引起人们的重视,对嵌入式系统的测试研究显得尤为重要。

嵌入式软件测试由于其自身的特点使得测试较为困难。由于嵌入式系统的自身特点,如实时性(Real-timing),内存不丰富,I/O 通道少,开发工具昂贵,并且与硬件紧密相关,CPU 种类繁多,其缺陷不像 PC 软件的缺陷容易修补等等[1]。对嵌入式软件测试与一般软件的测试策略有了很大的不同,可以说对嵌入式软件进行测试比对普通软件测试来说要难,而对嵌入式软件的性能测试是嵌入式软件测试中比较难的一部分。性能测试则是控制系统性能的有效手段,在软件的能力验证、能力规划、性能优化、缺陷修复等方面都发挥着重要作用。

在嵌入式软件测试中,采用正确的测试策略,可以提高嵌入式软件性能测试效率,避免目标系统的瓶颈。自从出现高级语言,开发环境与最终运行环境通常都是存在差异的,嵌入式系统更是如此。开发环境被认为是主机平台,软件运行环境为目标平台。相应的测试为 host-target 测试和 cross-testing[2]。本文根据自己的测试经验和对嵌入式软件测试的认识,就嵌入式软件测试在主机系统和目标系统环境下的测试策略加以讨论和研究。

收稿日期: 2007-04-30
第一作者: 男,1967 年生,高级工程师
E-mail: Leewei1030@163.com

1　主机仿真系统下嵌入式软件测试

在主机系统下,通过仿真器模拟器,可以对嵌入式软件进行单元级测试,可对测试对象进行覆盖率测试、变量跟踪、寄存器跟踪、内存资源使用等测试[3]。主机系统下嵌入式软件的测试流程如图 1。

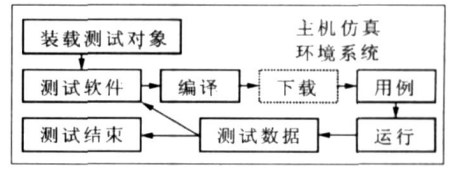

图 1　主机系统下嵌入式软件的测试流图

Fig. 1　Flowchart of embedded software testing on host system

以下以 Wave6000 为例,说明测试与 MCS51 相兼容的嵌入式软件的测试。

1.1　静态分析

导入被测对象工程,可以利用 Wave6000 观察窗口如图 2 所示对嵌入式软件进行静态分析。

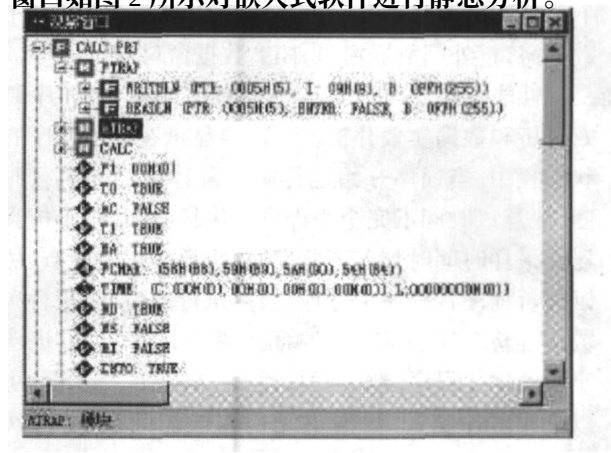

图 2　被测对象观察窗

Fig. 2　Observation window of tested object

可以观察到软件中定义了那些函数模块,函数接口包含几个参数,也可以看到模块内定义的局部变量,在观察窗中节点为 P 表示目前打开的工程,M 表示模块,F 表示函数,√表示简单变量,A 表示数组变量,P 表示指针,L 表示标号。

1.2 对被测对象进行动态测试

利用 Wave6000 对被测对象进行编译生成目标代码,通过装入目标代码就可以对被测对象进行动态测试。

通过初始化被测对象运行所需的测试用例,运行被测对象,利用 Wave 6000 可以观察程序的运行情况,语句和分支覆盖是否满足测试要求。如图 3 所示,深色为程序运行执行过的语句,浅色为未执行的语句。显示 CPU 执行了哪些函数、谁在调用、参数是什么、何时调用等情况。这种功能可以让测试者知道那些代码、分支没有被执行到。这样有助于提高代码质量并消除无用代码。

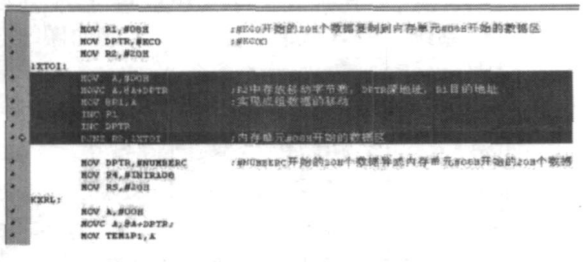

图 3 语句分支执行覆盖图

Fig. 3 Coverage of branch code execution

利用其 REG 和 SFR 窗口可以跟踪程序运行过程中 CPU 中通用寄存器和特殊功能寄存器中的内容变化。

Data 监视器:可以在不停止 CPU 运行的情况下显示指定变量内容,包括 Data 内部数据窗口、序数据窗口、外部数据窗口、BIT 数据窗口等。

性能跟踪、分析功能,主要包括计时器、程序时效分析和数据实效分析。计时器显示在应用程序的状态栏中,当程序开始运行时。累计程序的执行时间,注意该时间不完全为程序的执行时间,由于仿真器在采样时间时加入了监控时间;程序时效分析,可以分析程序各模块中过程、函数执行的时间、执行次数及占整个程序运行的时间的百分比等,可以进一步帮助优化程序,改善程序性能;数据实效分析,可以统计出各变量、数据单元是否被访问过、访问次数、访问频率和访问方式(由 R 和 W 表示访问方式,如 RW 表示先读后写),可以帮助改善成结构,使程序更加稳定、有效。如图 4 为程序执行时效分析窗口。

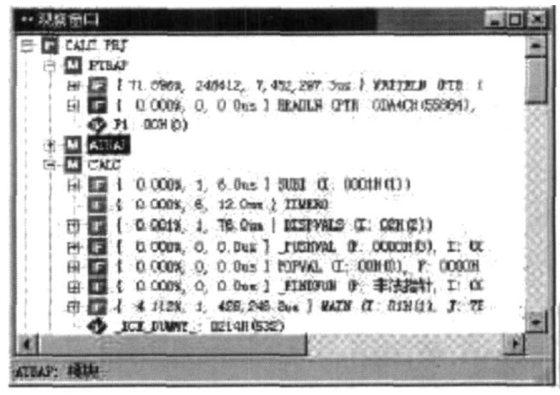

图 4 程序执行时效分析窗口

Fig. 4 Program runtime time performance analysis window

橄榄绿表示此函数已被执行过,红色表示函数中的某变量被改变。{X%,Y,Zus}函数名表示程序时效分析内容,其中 X 表示函数执行时间占程序总执行时间的百分比,Y 表示此函数被执行的次数,Z 表示该函数累计执行的时间,图中{71.896%,248412,7425297.3 us} WRITELN 表示模块中 WRITELN 函数被执行了 248412 次,累计运行时间 7425297.3 us,占整个程序运行时间的 71.896%。

2 目标环境下嵌入式软件测试策略

实时嵌入系统的广泛应用,对嵌入式软件的可靠性及其性能要求越来越高,而在主机环境下通过仿真环境来测试是非常困难的。为了获得客观正确真实的数据,必须在目标环境下对嵌入式软件进行测试,所有的系统测试和确认测试均需在目标环境下执行,包括恢复测试、安全测试、强度测试、性能测试等[3]。

2.1 测试数据的采集

当被测对象的可执行程序在目标环境下运行时就会产生数据,这些数据就是客观评估被测对象的稳定性、安全性、技战术指标满足用户需求的的重要输入条件,所以要顺利实现嵌入式软件测试,首先需要解决的就是如何把测试数据上载回主机[4]。经过实际摸索发现,可以把数据的上载分为 3 种方式:实际的物理通道;开发工具 IDE 的虚拟 IO 功能;读取内存区数据。

2.1.1 实际的物理通道 实际的物理通道包括如以太网、串口、并口、USB 等方式,在测试的时候,可以直接使用这种通信程序和主机通信,实现测试数

据上载。测试工具的库中 IO 函数使用 fprintf() ,那么在这种方式下,就需要修改测试工具的库,假设以太网的发数据的函数是 Netsend() ,那么需要定制 fprintf ,使它调用 Netsend() ,这样就可以实现数据上传了。也可以采用专用数据采集仪获得目标系统上的数据,如采用 NI 公司提供的数据采集议和 LabView 编程工具可以实时获得被测对象在目标系统运行产生的数据。

2.1.2　开发工具虚拟 IO　开发嵌入式软件一般需要支持交叉开发方式的开发工具,大多数这种开发工具具备编译、下载、调试的功能,测试时也可以借助开发工具来下载测试程序。有些高级的开发工具 IDE,具备虚拟 IO 功能,给测试带来了很大的方便,比如 TI CCS,持 fprintf ,直接在主机上生成测试数据文件。

2.1.3　读取内存数据　若目标系统既没有物理通信方式,开发工具也没有虚拟 IO 功能,可以采用读取内存数据的方式。开辟一块足够大的缓存,修改测试工具的库,把输出的数据写入 buffer 中,在测试过程中或者测试执行后,使用开发工具读取内存的功能把缓存中的数据读取出来,在主机上保存成文件。

2.2　嵌入式软件系统级测试

由于在仿真环境下采集的测试数据存在一定的误差,从而影响正确的评估嵌入式软件的性能、稳定和各项技术指标是否满足需求。只有在目标系统下对被测对象进行测试才可获得比较客观正确的测试数据,为正确的评估嵌入式软件质量提供科学依据。以下讨论如何利用 RTInsight 专业测试工具在目标系统测试嵌入式软件。

RTInsight 是 LDAR 公司的定时硬件数据采集器,通过和被测试系统的物理总线连接,实时监控系统的总线读写情况,配合测试工具 Testbed 及 RTView 软件的使用,实现时间性能分析,变量监控和堆栈监控等功能[5]。

Testbed 的 bitmap 插装技术不同于普通的插桩技术,在历史文件中,用一个位代表一个分支点,0 代表这个点没有执行过,1 代表已经执行了,这样历史文件的最小体积就相当于分支点个数/8 个字节[4]。而且在执行过程中,是在内存中开辟一个数组记录分支点信息,程序结束时才写入文件的,这就大大提高了执行效率。同时,由于采用 RTInsight 高速虚拟端口技术使得代码插装量可控制在每个特征点(即函数入口、出口,程序分支点)一到两条指令或语句,大大减少了插装代码增加对被测系统的影响,从而满足嵌入式系统在实时性方面的要求。

2.2.1　RTInsight 硬件连接　RTInsight 硬件连接如图 5 所示。

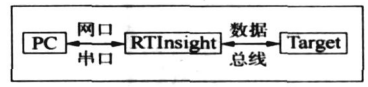

图 5　RTInsight 硬件连接图

Fig. 5　Illustration of RTInisight 's Hardware connection

RTInsight 通过通用探头或专用探头与目标系统 RAM 芯片插座、CPU 插座或专用总线(如 PC104)相连,支持所有 8 位、16 位及 32 位微处理器与微控制器。

通过串口和网口将 PC 机和 RTInsight 的连接,以便 PC 机向目标机下载程序和从 RTInsight 中高速获取实时数据。

2.2.2　性能测试　先通过编译生成的 map 文件查找到要检测的函数的开始地址和结束地址。然后,在 Rtview 中输入这些地址[6]。启动 Rtview 分析功能,运行被测试程序。

RTIsight 提供的代码插装方式适用于带指令预取与指令 CACHE 的嵌入式系统,可准确地分析系统执行时间与性能。能同时分析 4 G 个任务下的 128 K 个函数,最大的循环调用深度可达 1 K。提供系统总体执行时间;每个子程序最大、最小、累加执行时间与执行次数;中断响应时间。

提供变量监控分析功能,可实时监控 8 个系统变量和 2 个数组。

使用 RTInsight 堆栈监控功能可实时监控被测系统堆栈使用情况,一旦系统堆栈溢出,RTInsight 可将引起堆栈溢出现场实时跟踪记录下来供测试人员分析(如图 6,7 所示)。

能实时分析在有嵌入式操作系统各个任务的执行情况,例如是退出还是在执中;并且分析各个任务的时间性能,例如累计执行时间等等。

提供 128 K 跟踪分析功能,可实时跟踪记录系统执行状态,并可设置灵活的触发条件与记录条件。

3　结束语

嵌入式系统的应用在国防、航天、金融、通信及其他重要领域的应用愈来愈广泛,对其可靠性、安全性和性能指标的要求越来越高,只有加强对其测试

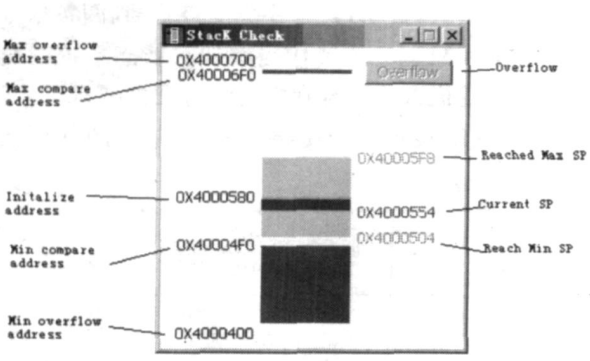

图 6 RTInsight 堆栈实时监控图

Fig. 6 Demonstration of RTInisight 's real-time stack monitoring

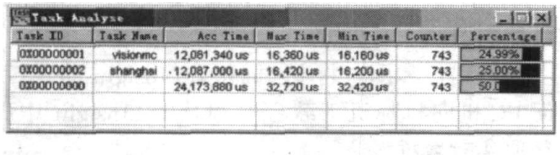

图 7 RTInsight 任务时间性能分析图

Fig. 7 View for RTInsight 's task time performance analysis

的力度,采用正确的测试策略和工具,才能保证对其测试的充分性,获得客观真实的测试数据,有助于对其做出客观的评价。

参考文献:

[1] BROEKMAN B, NOTENBOON E. Testing Embedded Software [CP/OL]. http: //edu. qiji. cn/eprint/abs/2756. html.

[2] TEHRANIPOUR M H, NOURANI M, FAKHRAISE S M, et al. Navabi Embedded Test for Processor and Memory Cores in System-on-Chips [CP/OL]. http //: www. citeseer. ist. psu. edu/696922. html.

[3] MYERS G J. **软件测试艺术**[M]. **王峰,陈杰,译. 北京: 机械工业出版社**, 2006.

[4] SCHMITT W. Automated Unit Testing of Embedded ARM Application [J]. Information Quarterly, 2004, 3 (4): 9.

[5] **上海创景计算机系统有限公司**. LDAR Testbed RTInsight **使用指南**[M]. 2005.

[6] **杜延, 刘从越. 嵌入式实时系统软件测试实践**[J]. **微计算机信息**, 2006, 22(26): 31 - 33.

The research on testing strategy of embedded software

LI Wei CHENG Zhao Hui

(Beijing P. O Box 7223 10th 100072)

Abstract: As application of embedded real-time systems increases, their reliability becomes more and more dependent upon the quality of embedded software. Testing of embedded software is an important way to guarantee its quality. In this paper, the testing strategy for embedded software is discussed, and issues concerning how to test embedded software on emulated host environment as well as target platform are addressed.

Key words: embedded software; testing strategy; covering rate testing; performance testing