基于 CUnit 自动化测试框架的设计与实现①

陈炳煌, 邵 明, 林秋果

(福建工程学院 电子信息与电气工程系, 福州 350108)

摘 要: 以 Xunit 为代表的软件自动测试框架已经趋于成熟, 针对嵌入式系统应用特点, 提出了一种基于 CUnit 的自动化测试框架, 并介绍了相关实现技术, 对 CUnit 单元测试框架进行二次开发, 使之适合在凌阳单片机 SPT6608A 上使用, 在实践中证明此框架行之有效, 能全面实现对 SPT6608A 软件模块的自动化测试.

关键词: 测试框架; CUnit; 自动化测试

Design and Implementation of Automation Testing Framework Based on CUnit

CHEN Bing-Huang, SHAO Ming, LIN Qiu-Guo

(Department of Electronic Information and Electrical Engineering, Fujian University of Technology, Fuzhou 350108, China)

Abstract: To XUnit, as a representative of the software to be automatic test framework has tended to mature. Based on embedded system application characteristics, this paper proposed a test automation framework based on the CUnit, and introduced the relevant implementation technology. The CUnit unit test framework was secondary developed, was made suitable for use in SPT6608A sunplus single chip microcomputer. In practice, the framework is proved effective, it can achieve to SPT6608A software modules of test automation.

Key words: testing framework; CUnit; automated testing

近年来随着嵌入式系统应用不断扩展, 嵌入式系统的软硬件规模和系统的复杂性不断提高, 如何更好的保证系统软件的质量就成为一个重要的问题[1]. 要保证嵌入式软件系统的正确性就必须对其软件进行全面的测试, 如果按照传统的手工测试方式, 不仅需要花费大量的人力、财力和物力, 而且受软件上市时间的压力, 传统的手工测试方式已经无法满足实际测试的要求. 因此我们迫切需要一种测试工具能够在软件开发的开始阶段、中间阶段、系统总集成等过程中对软件进行在线的自动测试与分析[2], 以保证最终软件的可靠性和稳定性. 这样一方面可以大大提高研发的效率, 另一方面也保证的软件的质量.

本文从开源测试框架的评估入手, 筛选出开源测试框架 CUnit[3-4], 成功的对该测试框架进行二次开发并移植到嵌入式系统中, 通过实例验证了框架的正确性和可移植性, 最终形成适合在嵌入式系统中使用的单元测试框架. 文中将测试框架移植到凌阳单片机 SPT6608A 上, 完成了某公司的某个软件模块的自动化测试, 并输出相应的测试报告.

1 CUnit框架使用说明

CUnit 是 XUnit 家族中的一员, 它是一种单元测试自动化框架, 负责管理和运行 C 语言下的单元测试工作[5], 以静态库的形式提供给用户使用, 图 1 给出了 CUnit 结构描述.

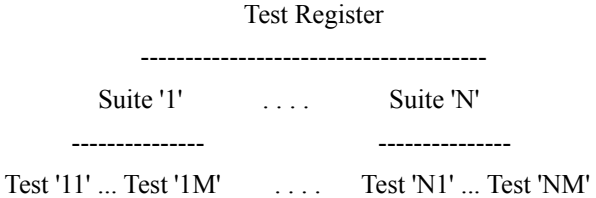

图 1 框架结构

① 基金项目:福建省科技重大专项资助项目(2010HZ0002-1);福建省经贸委 2006 省装备制造业消化吸收再创新项目(闽经贸计[2006]804 号)
收稿时间:2012-07-10;收到修改稿时间:2012-09-22

CUnit 的测试是单线程启动,只能注册一个测试 Test Registry, 一次测试(Test Registry)可以运行多个测试包(Test Suite), 而每个测试包可以包括多个测试用例(Test Case), 每个测试用例又包含一个或者多个断言类的语句. 具体到程序的结构上, 一次测试下辖多个 Test Suite, 它对应于程序中各个独立模块; 一个 Suite 管理多个 Test Case, 它对应于模块内部函数实现. 每个 Suite 可以含有 setup 和 teardown 函数, 分别在执行 suite 的前后调用.

CUnit 提供了一批类 Xunit 的 assert 函数[6]. 这些 assert 函数的作用是, 断言当前条件是否为真. 断言方式由函数功能定义, 如比较两个字符串是否相等, 两个函数是否相等之类. 一旦断言为真, 则继续执行, 当某个 Test 函数成功执行完毕, 则算通过了一个 test. 一旦某个断言为假, 则会执行 Add Failure 函数, 在屏幕上显示失败的断言.

2 CUnit框架二次开发

2.1 CUnit 框架开发移植原理

对自动化测试框架进行开发和设计的最终目的是改进出理想的嵌入式软件测试自动化框架, 完成在嵌入式系统中的白盒测试. 与通用的软件测试不同, 在嵌入式软件测试中会遇到很多技术上的难题, 其中的一项: 加入测试代码后, 可能会导致嵌入式系统的内存不足, 影响原先的软件代码功能. CUnit 单元测试框架内存的使用情况是最少的, 但是增加测试用例, 内存的使用会相应的增长. 当测试用例非常多且失败的断言也非常多时, CUnit 单元测试框架的内存占用量就非常大, 甚至会超过 Seatest 和 Embedded Unit. , 凌阳单片机 SPT6608A 上的 RAM 只有 27K, 所以应分析 CUnit 单元测试框架的内存使用, 尽量将内存的使用降到最低.

从图 2 可以看出, 框架的内存使用主要是在增加测试包、测试用例和失败断言上. 受 Seatest 和 Embedded Unit 单元测试框架占用内存量为一个固定值的启发, 在二次开发时尽量将内存使用设置为小的固定值.

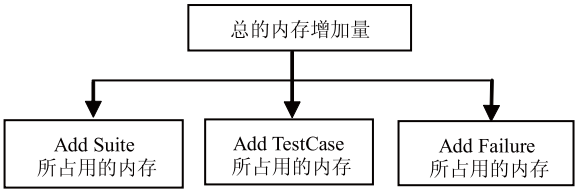

图 2 框架内存使用分布

2.1.1 增加测试包

原框架 TestSuite 内存占用=CU_MALLOC(sizeof(CU_Suite))+CU_MALLOC(strlen(strName)+1);其中 CU_Suite 是结构体, 对此结构体进行分析, 知其修改的可能性比较小, 而 strName 是设计者为测试包所取的名字长度, 所以, 可以从这个命名长度入手, 将测试包的命名长度限定在一个固定的范围. 对 create_suite(...)函数进行修改如下: 在函数 create_suite(...)中添加断言“assert(strlen(strName)<=7);”, 由于规定了测试包命名最大长度不超过 7, 因此就不会出现不同设计者对测试包命名长度相差很大的情况. 框架修改后流程图如图 3 所示.

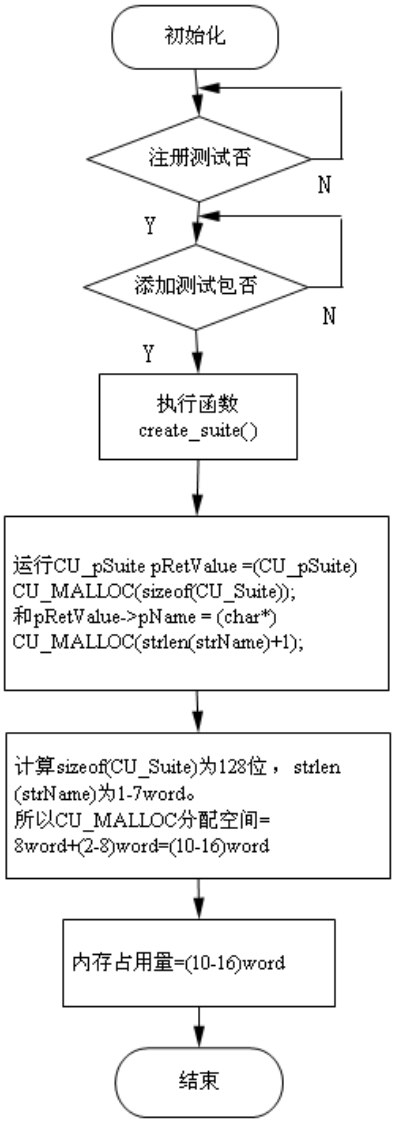

图 3 添加测试包流程图

2.1.2 增加测试用例

原框架 TestCase 内存占用=CU_MALLOC (sizeof(CU_Test))+CU_MALLOC(strlen(strName)+1), 其中 CU_Test 是结构体, 对此结构体进行分析, 知其修改的可能性比较小, 而 strName 是设计者为测试用例所取的名字长度, 所以, 可以从这个命名长度入手, 将增加测试用例的命名长度限定在一个固定的范围. 对 create_test(...) 函数进行修改如下: 在函数 create_test(...)中添加断言“assert(strlen(strName)<=7);”, 规定命名最大长度不超过 7, 那样就不会出现不同设计者对测试用例命名长度相差很大的情况. 具体的流程图参考图 3.

2.1.3 失败断言

原框架 Add Failure 内存占用=CU_MALLOC (sizeof(CU_FailureRecord))+CU_MALLOC(strlen(szFileName)+1)+CU_MALLOC(strlen(szCondition)+1); 其中 CU_FailureRecord 是结构体, 对此结构体进行分析, 知可删除此结构体中的一些变量, 减小内存的占用; 且分析知失败断言中的路径名 strFileName 和断言内容 szCondition 可以直接通过串口的方式在 PC 机上进行显示, 这样可以大大减少框架的内存占用量, 所以对 add_failure(…) 函数做如下修改: 省略 pFailureNew->strFileName = (char*)CU_MALLOC (strlen(szFileName)+1); 和 pFailureNew->strCondition = (char*)CU_MALLOC(strlen(szCondition)+1); 直接对内容进行显示, 具体的流程图参考图 3.

对 CU_FailureRecord 结构体修改如下:

```
typedef struct CU_FailureRecord
{
struct CU_FailureRecord* pNext;
struct CU_FailureRecord* pPrev;
} CU_FailureRecord
```

2.2 CUnit 框架输出方式修改

本文利用串口将数据显示在 PC 机上. 首先查看终端上的串口配置文件: ComName="COM6", 对串口调试工具进行相应的设置, 选择串口号为 COM6、波特率为 115200、数据位 8 位、1 位停止位、无校验位、无流控制位. 要实现终端与 PC 机的通信, 还必须在程序初始化时打开串口: OpenFile(DEV_UART, BAUDRATE_115200, NO_PARITY, 256,UART_PORT_OUTER);函数中 DEV_UART: 要打开文件的名称; BAUDRATE_115200: 选择的波特率; NO_PARITY: 无奇偶校验位; 256: UART 数据缓冲区长度; UART_PORT_OUTER: 端口输出.

当确定串口已经打开后, 在需要数据显示的地方做如表 1 所示的修改. 框架中涉及 fprintf 的地方较多, 无法一一举例, 因为通过串口输出的报告由于排版较乱, 因此对单个的 fprintf 语句进行修改. 修改的内容是一样的. 此外在串口方式中对内容进行显示和 fprintf 函数是一样的道理, 框架中还涉及到 snprintf 语句, snprintf 语句格式和 sprintf 语句格式有很大的差别, 所以在框架中有涉及 snprintf ( )函数的地方需改为 sprintf( ), 而且还要特别注意参数问题, 因为 snprintf( )函数中无显示功能, 所以要避免修改后的内容与框架中的其他内容重复. 输出报告如图 4 所示, 分析输出报告可知, 输出报告和框架在 PC 机上的效果相同, 框架经内存占用修改、头文件不兼容修改和输出方式修改后能在 SPT6608A 上编译成功.

表 1　框架移植对 fprintf 语句修改

Basic.c 文件	代码段(184行开始)

源代码	if (CU_BRM_SILENT != f_run_mode)
	fprintf(stdout, "\n\n %s" CU_VERSION"\n %s\n\n",
	_("CUnit - A unit testing framework for C - Version "),
	_("http://cunit.sourceforge.net/"));

修改后代码	char buffer[128];
	unsigned short wDataLen;
	CU_set_error(CUE_SUCCESS);
	if (CU_BRM_SILENT != f_run_mode)
	{
	sprintf(buffer,"\n\n %s" CU_VERSION"\n %s\n",
	_("CUnit - A unit testing framework for C - Version "),
	_("http://cunit.sourceforge.net/"));
	wDataLen=strlen(buffer);
	WriteFile(DEV_UART, buffer, wDataLen , 0);
	}

3 框架正确性验证

虽然框架可以在 SPT6608A 上编译成功, 但是还无法验证移植后框架的正确性, 本文取一个软件模块进行自动化测试并分析测试输出报告, 从而来验证框

架经移植后是否能正常可靠运行. 如下所示框架测试 StringEx.c 文件的输出报告, 由于篇幅较长以下仅列出部分代码:

/*****************************/

CUnit - A unit testing framework for C - Version 2.1-2

http://cunit.sourceforge.net/

Suite: Sui_1

Test:

test1 ...C:/C7S_Platform/DebugPrj/MiniDemo/Source/CUnit/Simple_Test/simple_test.c, 18

*CalcStringWrap(p,4,3)=='w'

FAILED

Test:

test2 ...C:/C7S_Platform/DebugPrj/MiniDemo/Source/CUnit/Simple_Test/simple_test.c, 32

CalcStringLines(p,5)==8

……

/*********************************/

从报告可以看出, 改进后的测试输出报告和未经修改的测试输出报告形式是相同, 差别是改进后的测试输出报告没有对程序运行的时间进行显示. 框架的显示情况、排版和原框架相同, 因此框架中对串口的修改部分完成且可用. 输出报告中提示失败的断言, 的确都是测试结果和预期结果不符的情况, 测试输出结果是正确的. 对框架中正确的断言进行分析知, 所有测试执行为正确的断言, 的确都是测试结果等于预期结果的情况.

输出报告中 Run Summary 部分如下:

Run Summary:Type Total Ran Passed Failed Inactive

suites	1	1	n/a	0	0
tests	7	7	0	7	0
asserts	42	42	35	7	n/a

通过分析得知, 输出报告中关于提示失败的断言、运行的数量、失败的数量和实际程序结果相同, CUnit 单元测试框架移植完毕, 经改进后的测试框架能正常运行.

CUnit 单元测试框架经二次开发, 移植到 SPT6608A 上, 总的内存增加量={(Suite 个数)*(10-16)+(TestCase 个数)*(7-13)+(FailureAssert 个数)*2}word. 因为终端的内存是 27K, 按目前的内存使用情况, 增加一个测试包, 内存的使用增加 10-16 word; 增加一个测试用例, 内存的使用增加 7-13word; 有一个失败的断言, 内存的使用增加 2 个 word. 即使有 10 个测试包、100 个测试用例、1000 个失败的断言, 内存的最大增加量也只为 3460word, 内存的占用情况还是比较可观的. 加入 CUnit 代码后, SPT6608A 的 flash 占用增加了 12K, 而 SPT6608A 系统终端上面有 1M 多 ROM, 这个值是终端所能接受的.

4 总结

本文从开源测试框架的评估入手, 筛选出开源测试框架 CUnit, 然后对此开源测试框架进行二次开发并移植到凌阳单片机 SPT6608A 系统中, 最后通过选取某公司的软件模块进行自动化测试, 通过对输出的测试报告进行分析, 验证了该框架测试的正确性. 最终形成适合在嵌入式系统中使用的单元测试框架, 从而降低测试工作带来的复杂性, 减少测试执行所需的时间.

参考文献

1 刘波.基于 cunit 的自动测试框架.电脑知识与技术,2007,(18):1631−1633.

2 Kim EH, Na JC, Ryoo SM. Test automation Framework for implementing continuous integration.6th International Conference on Information Technology: New Generations. April 2009:784−789.

3 Jerry St.Clair.CUnit. http://cunit.sourceforge.net/documentation.html,2005.

4 Hamill P. Unit Test Frameworks.O'reilly,2004:45−50.

5 Beck K. Test Driven Development: By Example. Addison-Wesley Longman,2002:60−62.

6 杜庆峰,李娜.白盒测试基路径算法.计算机工程,2009,(8):100−102,123.