机车电传动 2016 年第 5 期（2016-09-10）
ELECTRIC DRIVE FOR LOCOMOTIVES № 5，2016（Sep. 10, 2016）
文章编号：1000-128X(2016)05-0102-05

LDRA TestBed 在 CTCS2-200C 扩展单元软件测试中的应用

周永健，范　明，张　森，赵东旭
（中国铁道科学研究院 通信信号研究所，北京　100081）

摘　要：介绍了 LDRA TestBed 软件测试工具和 CTCS2-200C 扩展单元，以 CTCS2-200C 为例，通过对软件进行静态和动态测试，参照 EN 50128:2011 标准，提高了整个软件开发生命周期中的各项验证和确认活动效率，降低了软件的风险。

关键词：软件测试；TestBed；EN 50128；CTCS2-200C 扩展单元；列控车载系统
中图分类号：U284.48；TP311.5　　**文献标识码**：A
doi：10.13890/j.issn.1000-128x.2016.05.025

Application of LDRA TestBed in CTCS2-200C Expansion Unit Software Testing

ZHOU Yongjian, FAN Ming, ZHANG Miao, ZHAO Dongxu

(Signal & Commumcation Research Institute,China Academy of Railway Sciences, Beijing 100081)

Abstract: The application of LDRA TestBed and CTCS2-200C expansion unit was introduced. Using CTCS2-200C as an example, through the static and dynamic testing of the software, with reference to the 50128:2011 EN standard, the verification and validation activities in the entire software development life cycle were improved, and the risk of the software was reduced.

Keywords: software testing; TestBed; EN 50128; CTCS2-200C expansion unit; train control onboard system

0　引言

CTCS2-200C 扩展单元作为城际 C2+ATO 列控车载系统的关键设备，采用二乘二取二安全冗余结构，实现数据通信和安全 I/O 采集 / 驱动继电器接口的扩展，完成与 ATP、ATO 子系统和无线通信车载单元 (RTU) 的数据通信。扩展单元应用软件属于安全苛求软件 [1]，其安全完整性等级是 SIL4 级。在系统安全评估中，充分的软件测试，最大限度地发现软件中的错误并减少软件中残留的隐患，是加强软件安全性的有效途径 [2]。TestBed 是英国 LDRA 公司开发用来提高软件产品质量，在软件编程、软件测试和软件维护阶段得到广泛应用的测试工具 [3]。对于 CTCS2-200C 扩展单元软件这样的功能模块多、模块接口之间存在多种数据传递的复杂软件系统，采用 LDRA TestBed 软件测试工具，对提高测试效率、有效查找软件缺陷、保证测试的完备性都是十分必要的。参照 EN 50128:2011 标准，本文将对测试流程及策略进行介绍。

收稿日期：2016-03-11
基金项目：中国铁路总公司重点课题（2014X003-E）

1　系统架构

CTCS2-200C 扩展单元主要完成 ATP 通信前置机的功能和安全采集 / 驱动的功能。其中，通信前置机功能包括：车—地无线通信功能及与 ATP、ATO 子系统通信功能；安全采集 / 驱动主要是允许车门控制、允许 ATO 控制等新增的信息采集和驱动，如图 1 所示。

CTCS2-200C 扩展单元接口包括：直接采集动车组车辆状态继电器接点，驱动则是通过弹力式安全继电器控制动车组操作；车地间采用 GSM-R 无线通信，

GSM-R 无线通信加密 / 解密算法符合 sub037 协议；与 ATP、ATO 以及无线通信模块 RTU 之间采用串行通信接口进行通信，单通道 +CRC 校验，CTCS2-200C 扩展单元的安全完整性等级应达到 SIL4 级要求。

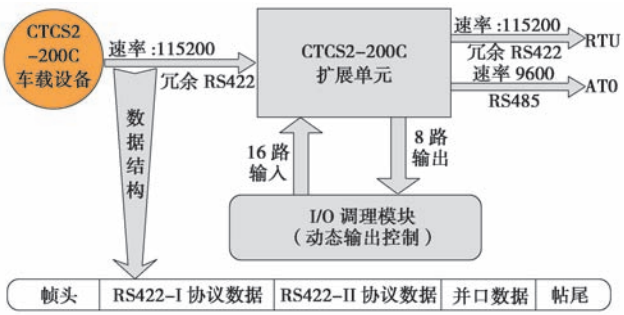

图 1　200C 扩展单元功能示意图

CTCS2-200C 扩展单元软件采用 C/C++ 语言作为编程语言，使用 CCS 5.2 集成开发环境，主要划分为 8 部分：安全平台相关功能部分、通用平台公用函数、与 ATP 通信功能、与 ATO 通信功能、与 FIMO 通信功能、与 RTU 通信功能、无线通信安全功能和维护监测功能。

CTCS2-200C 扩展单元软件功能模块多、模块接口之间数据通信复杂，工程文件下包含 53 个头文件，54 个 C 和 CPP 文件，进行全面细致的分析测试其工作量极大，需要引入 LDRA TestBed 软件测试工具，发挥其在软件代码评审、质量评审、设计评审、测试管理、单元测试和集成测试中的自动化优势。

2　测试概要

通常软件开发要经过设计、测试、验证以及确认 4 个步骤，并且在测试流程启动之前，需要确认系统软件的安全完整性等级以及实现软件开发的模型。本文涉及的功能软件安全完整性等级为 SIL4 级，采用 V 型软件开发模型。根据 EN 50128:2011 标准，整个测试流程一般分为静态测试、单元测试、集成测试 3 个工作阶段，其中为了确保在测试工作中测试与设计工作独立、验证与测试过程独立、确认与验证工作独立，采用了分别配置验证员和确认员的人员组织结构。

在每个设计阶段，测试员在设计者完成设计和调试后，按照相应的测试要求执行测试，分析和总结测试结果，形成测试报告。验证员对每个设计阶段和测试阶段的正确性、一致性等进行阶段性验证。确认员对每一阶段的设计、测试和验证活动进行检查，以确认是否可以进入下一阶段活动。测试活动中如发现缺陷，则需根据缺陷等级决定是否需要回归测试，每个测试阶段的完成与否，在参考测试报告结果的同时还要考虑缺陷等级，缺陷等级的划分如表 1 所示。

表 1　缺陷等级的划分

缺陷等级	原　则
一类	系统功能缺陷，应用软件架构缺陷，属于核心安全算法的逻辑缺陷、安全功能缺陷和安全功能缺陷克服等，该类缺陷需供应商进行第三方认证或外部专家组测试、评审，且需要回归测试
二类	应用控制功能缺陷、接口功能缺陷、缺陷克服等，该类缺陷变更应执行供应商内部安全评估流程，且需要回归测试
三类	数据缺陷、非安全功能缺陷及上述以外的其他非安全相关的缺陷等，该类缺陷变更应执行供应商内部的检验测试发布流程

3　静态测试

静态测试是对代码规则、代码结构、数据流、控制流等内容进行的测试，是不需要运行程序的静态分析走查。一般采用人工代码走查配合软件工具分析的方式，如函数参数的不匹配、不应该的循环嵌套、没有被允许就使用递归、变量定义没使用、未对空指针进行防护等。

3.1　基本测试步骤

①设置分析范围（源文件和头文件），配置编码规则集（MISRA C++:2008）和编译器（与开发环境编译器一致），若被测文件数量较多，可通过建立 Set（集）的方式来集中分析。

②在 Select Analyse 列表中选择前 6 项执行静态分析（Main Static Analysis、Complexity Analysis 等）。

③看保存分析报告（Code Review Report、Metrics Report 等）。

④分析生成报告，对可疑和重点模块进行代码规则、检查控制流、数据流、接口和表达式分析，定位有缺陷的代码区域。

3.2　静态分析

以 1.0 版本代码中无线信息包处理模块（app_itf_packet.cpp）为例，功能是实现无线信息包的解析和组帧。

①分析 Code Review Report，可查看该文件中所有函数规则检查通过情况（如图 2），以 Lire_Packet_3 子函数为例，跳转定位到 Quality Report 中相应规则违

Overall Results - Programming Standards

	Quality Result	Uniq Violations % in Function	No in Function	Breakdown of Violations
App_Itf_Packet::Ecrire_Packet_44_C26_Mode	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_44_C26	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_44_C25_2	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_44_C25	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_44_C24_2	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_44_C24	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_11_2	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_11	Fail	6	Unique violations - 4	2-时 2-0
App_Itf_Packet::Ecrire_Packet_0_Level	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_0_Mode	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Ecrire_Packet_0	Fail	6	Unique violations - 4	2-时 2-0
App_Itf_Packet::Transmit_Packet_72	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Lire_Packet_72	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Transmit_Packet_58	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Lire_Packet_58	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Transmit_Packet_C31	Pass	0		
App_Itf_Packet::Transmit_Packet_C23	Pass	0		
App_Itf_Packet::Transmit_Packet_C22	Pass	0		
App_Itf_Packet::Transmit_Packet_C21	Conditional Pass	1	Unique violations - 1	1-0
App_Itf_Packet::Lire_Packet_C31	Fail	4	Unique violations - 3	1-时 2-0
App_Itf_Packet::Lire_Packet_C23	Fail	4	Unique violations - 3	1-时 2-0
App_Itf_Packet::Lire_Packet_C22	Fail	4	Unique violations - 3	1-时 2-0
App_Itf_Packet::Lire_Packet_C21	Fail	4	Unique violations - 3	1-时 2-0
App_Itf_Packet::Lire_Packet_44	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Transmit_Packet_42	Fail	1	Unique violations - 1	1-时
App_Itf_Packet::Lire_Packet_42	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Lire_Packet_3	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Lire_One_Packet_sub	Fail	3	Unique violations - 2	1-时 1-0
App_Itf_Packet::Lire_One_Packet	Fail	4	Unique violations - 3	2-时 1-0
App_Itf_Packet::Transmit_Packets_sub2	Pass	0		
App_Itf_Packet::Transmit_Packets_sub1	Pass	0		
App_Itf_Packet::Transmit_Packets	Pass	0		
App_Itf_Packet::Initial_Packet	Conditional Pass	1	Unique violations - 1	1-0
App_Itf_Packet::App_Itf_Packet	Conditional Pass	1	Unique violations - 1 (70 out of 284 checked)	1-0

图 2　测试文件下各函数编码规则检查结果

反项和代码行（如图 3）。

App_Itf_Packet::Lire_Packet_3
(191 to 225 app_itf_packet.cpp) - FAIL

Summary

Code	Line	Violation	Standard
M	191	Parameter should be declared * const. : param 1	MISRA-C++:2008 7-1-1

[Top of Report | Key to Terms | Procedure Table | Contents]

图 3　Lire_Packet_3 函数编码规则检查违反项

MISRA C++:2008 编码规则涵盖了软件测试的大部分要求，但有些规则对嵌入式软件的测试并不适用，这套规则是高度可配置的，将规则分为强制执行和可选执行，可根据具体要求设置。

②从 Quality Report（Overall Quality Summary）中可定位到 MISRA C++:2008 对应的违法规则（如图 4），并能查看实例理解该规则，用于代码评审。

0	567 S	Pointer arithmetic is not on array.	MISRA-C++:2008 5-0-15
22	503 S	Parameter should be declared * const.	MISRA-C++:2008 7-1-1
0	7 C	Procedure has more than one exit point.	MISRA-C++:2008 6-6-5
0	1 D	Unused procedure parameter.	MISRA-C++:2008 0-1-11, 0-1-12
0	25 D	Scope of variable could be reduced	MISRA-C++:2008 3-4-1
0	26 D	Variable should be defined once in only one file	MISRA-C++:2008 3-2-2, 3-2-4
0	27 D	Variable should be declared static	MISRA-C++:2008 3-3-1
0	33 D	No real declaration for external variable	MISRA-C++:2008 3-2-4
0	34 D	Procedure name re-used in different files.	MISRA-C++:2008 3-2-2
0	35 D	Expression has side effects.	MISRA-C++:2008 5-0-1
0	36 D	Prototype and Definition name mismatch.	MISRA-C++:2008 8-4-2
0	38 D	More than one control variable for loop.	MISRA-C++:2008 6-6-1
0	42 D	Local pointer returned in function result.	MISRA-C++:2008 7-6-1
0	43 D	Divide by 0 found.	MISRA-C++:2008 0-3-1
0	46 D	Member function should be declared const.	MISRA-C++:2008 9-3-3
0	55 D	Modification of loop counter in loop body.	MISRA-C++:2008 6-5-3
0	56 D	Throw found with no catch in scope.	MISRA-C++:2008 15-3-4

Number of Violations	LDRA Code	Required Standards	MISRA-C++:2008 Code
0	59 D	Parameter should be declared const	MISRA-C++:2008 7-1-1
0	60 D	External object should be declared only once	MISRA-C++:2008 3-2-3
0	61 D	Procedure should be declared static	MISRA-C++:2008 3-3-1
11	62 D	Pointer parameter should be declared const	MISRA-C++:2008 7-1-2
0	63 D	No definition in system for prototyped procedure	MISRA-C++:2008 3-2-4
0	65 D	Void function has no side effects.	MISRA-C++:2008 0-1-8
0	66 D	Non boolean control variable modified in loop.	MISRA-C++:2008 6-5-6
0	69 D	Procedure contains UR data flow anomalies.	MISRA-C++:2008 8-5-1
1	70 D	DU anomaly. variable value is not used.	MISRA-C++:2008 0-1-6, 0-1-9
0	71 D	No matching catch for throw in called function.	MISRA-C++:2008 15-3-4

图 4　app_itf_packet.cpp 中违反 MISRA C++:2008 编码规则列表

通过工具分析和人工确认可以发现，Lire_Packet_3 的输入参数（const uint8 *Buf）未进行空指针防护，需要添加空指针防护代码。对测试结果进行分析、统计，共检查到违反编码规则 135 条，人工代码走查共发现 27 项错误，代码不合格项需要设计人员反复修改才能消除，无法通过的要由开发人员给出不影响系统安全性的充分说明。

③分析 Metrics Report，可查看代码质量度量、复杂度度量 (Complexity Metrics) 结果（如图 5）及数据流信息（Dataflow Information）等，用于软件质量评审。

从图 5 可以看出 Ecrire_Packet_0_Mode 函数节点数和圈复杂度超标，需要重新分割或重构代码。通过全面分析度量报告，可以对软件的结构化程度、软件复杂度、可维护性和可测试性等指标进行量化评估，也能检查出程序注释不规范、存在不可达代码、无限循环、扇入 / 扇出数和圈复杂度不满足限制规则等问题。

④分析 Dataflow Report，对参数定义、输入 / 输出变量、全局 / 局部变量、模块调用 / 被调用关系、UR（声明后没有定义就被引用）、DU（定义后没有被引用）和 DD（两次定义之间没有被引用）等数据异常情况进行报告，用于软件设计评审。

通过分析数据流报告和人工确认可以完成：代码接口分析、数据流异常检查、指针分析、强制类型检查、数组边界检查、潜在的内存泄漏侦测，发现函数无返回值、函数输入参数过多等问题。

通过以上静态分析，以 CTCS2-200C 扩展单元无线信息包处理模块（app_itf_packet.cpp）为例，针对其主要功能，主要对包的解析和组帧函数进行接口分析、边界检查和数据流异常测试。对涉及到安全功能的通信模块，静态测试不仅有助于进一步查错，还可为单元测试、集成测试等动态测试提供参考。

4　动态测试

动态测试是通过设计测试用例，执行待测程序后比较实际运行程序的输出结果与预期结果的区别，跟踪代码的运行情况，从而发现错误的方法。这种动态方法由 3 块组成：设计测试用例、执行测试、分析测试结果。动态测试方法可分为黑盒测试、白盒测试以及灰盒测试。动态测试过程包括单元测试和集成测试。

4.1　单元测试

Procedure	Knots	Cyclomatic Complexity	Essential Knots	Ess. Cycl. Complexity	Structured Proc (SPY)
App_Itf_Packet::App_Itf_Packet	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Initial_Packet	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packets	0 (F)	1 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packets_sub1	0 (F)	5 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packets_sub2	0 (F)	5 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_One_Packet	1 (F)	4 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_One_Packet_sub	15 (F)	7 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_3	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_42	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_42	0 (F)	1 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_44	10 (F)	5 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_C21	4 (F)	3 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_C22	0 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_C23	0 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_C31	0 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_C21	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_C22	0 (F)	1 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_C23	0 (F)	1 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_C31	0 (F)	1 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_58	4 (F)	3 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_58	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Lire_Packet_72	4 (F)	4 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Transmit_Packet_72	4 (F)	2 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Ecrire_Packet_0	3 (F)	3 (F)	0 (F)	1 (F)	Yes (F)
App_Itf_Packet::Ecrire_Packet_0_Mode	120 (F)	16 (F)	0 (F)	1 (F)	Yes (F)

图 5　app_itf_packet.cpp 中各函数复杂度度量指标

单元测试通常是指把软件划分为最小可测试模块单元 (例如函数或类)，需要运行程序来进行的检查和验证测试。为方便验证和确认，可以将静态测试中的控制流和数据流分析结果用来辅助参与测试案例的设计，运用白盒测试方法，对模块单元的语句覆盖、分支覆盖和修正 / 判定覆盖指标进行统计，验证所有程序语句（S）、分支（B）、条件（MC/DC）

都能正常运行覆盖。

测试基本步骤及注意事项：

①确认函数测试案例说明，如需对代码进行修改应先完成修改（如涉及到静态局部变量影响测试覆盖率的模块时），且保证所作修改不会影响函数功能并能够通过编译。

测试案例的设计方法是先采用黑盒测试常用方法：边界值测试、等价类分析、错误推测法及结构性测试，再结合白盒测试中的逻辑覆盖和基本路径测试法，形成充分的测试数据。

②选择“Isolate fully all code elements”模式启动 LDRA TestBed 下的 TBrun，建立测试序列并设置桩函数，根据函数输入输出要求对其进行输入设置、输入检查、输出赋值、输出检查。

③设置用户变量：用户变量是测试人员使用的临时变量，设置后将出现在代码头部作为全局变量，在当前测试序列中有效。

④设置桩函数：打桩的函数在用例执行时不会被调用，但可以通过设置桩函数返回值或改变全局变量来模拟函数功能，LDRA TestBed 提供灵活的打桩方法，可根据代码具体情况灵活处理。

⑤初始化代码：可在此处进行用户变量以及全局变量的赋值操作，LDRA TestBed 提供灵活的初始化方法，处理数组、指针、结构体时比较方便。

⑥根据软件执行的动态流程图及语句覆盖、分支覆盖和 MC/DC 覆盖信息，快速定位错误代码和接口。

当涉及到条件判断复杂的函数模块时，LDRA TestBed 可自动分析被测软件中 MC/DC 条件组合，并产生 MC/DC 测试计划（如图 6），指导软件测试人员设计测试数据。例如函数代码中出现：

```
if ( (ValidOperation(x) == y) || (count < MAX_VAL) && (count > MIN_VAL) || !bFinished)
```

通过分析动态测试报告中的 MC/DC 覆盖条件执行情况，可以设计最少的测试案例完成要求的覆盖率。

```
Full Truth Table For Decision
-----------------------------

COMBINATIONS                | EXECUTED BY RUNS  | COMBINATION PAIRS                  POSSIBLE MINIMAL
INDEX   C1 C2 C3 C4      R  | PREV  CURR  COMB  | SHOWING INDEPENDENCE               MC/DC SETS
1    f( T, T, T, T )  -  T    NO    NO    NO
2    f( T, T, T, F )  -  T    NO    NO    NO
3    f( T, T, F, T )  -  T    NO    NO    NO
4    f( T, T, F, F )  -  T    NO    YES   YES    c1                                  #  #
5    f( T, F, T, T )  -  T    NO    NO    NO     |
6    f( T, F, T, F )  -  T    NO    NO    NO     |  c1
7    f( T, F, F, T )  -  T    NO    NO    NO     |  |
8    f( T, F, F, F )  -  T    NO    NO    NO     |  |  c1
9    f( F, T, T, T )  -  T    NO    NO    NO     |  |  |
10   f( F, T, T, F )  -  T    NO    YES   YES    |  |  |  c2 c3                        #  #  ESSENTIAL
11   f( F, T, F, T )  -  T    NO    NO    NO     |  |  |  |  |  c4                    #
12   f( F, T, F, F )  -  F    NO    YES   YES    c1 |  |  |  c3 c4                    #  #  ESSENTIAL
13   f( F, F, T, T )  -  T    NO    NO    NO        |  |  |           c4              #
14   f( F, F, T, F )  -  F    NO    NO    NO        c1 |  c2          c4              #  #  ESSENTIAL
15   f( F, F, F, T )  -  T    NO    NO    NO           |                    c4
16   f( F, F, F, F )  -  F    NO    NO    NO           c1                   c4
-----------------------------------------------------------------------------------------------------
INDEPENDENCE SHOWN BY PAIR EXECUTED              C1  -   -   -   C3  -   -   -
-----------------------------------------------------------------------------------------------------
```

图 6　MC/DC 覆盖测试计划

⑦调整测试案例得出各类覆盖率数据验证测试。

以类型转换及 CRC 模块（app_itf_common.cpp）为例，测试结果见图 7。

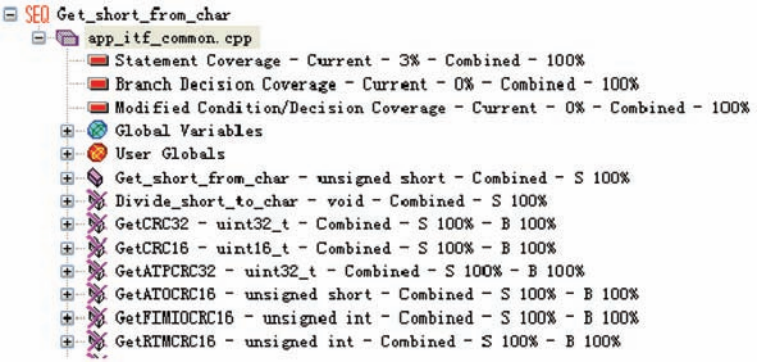

图 7　app_itf_common.cpp 中单元测试结果

单元测试的优势是将精力集中在程序的较小单元上，便于定位错误和纠错，并且可以划分多个模块后并行开展测试任务，也为多个函数配合的集成测试打好基础。

4.2　集成测试

集成测试通常是指：根据软件结构功能需求的设计划分为各个功能模块和接口模块，将已经通过的单元测试模块组合，采用一定的集成方法，将软件作为一个整体进行测试；测试需要动态运行软件并且需要调用实体单元模块，其目的是证明各软件模块的接口以及接口之间都满足规定的需求。常见的集成测试有自顶向下集成、自底向上集成和混合集成 3 种方法。

自顶向下集成能较早发现程序功能模块中控制和决策逻辑的缺陷或错误，但是测试时底层模块通常采用桩函数调用，底层数据无法透明传输至顶层，导致测试并不完整；自底向上集成能较早测试比较容易出错的输入输出模块（和底层硬件相关）、加密解密等基础算法模块等，能有效划分多个功能模块并行测试，提高测试效率，但是测试代码无法作为一个整体进行实体测试，控制逻辑无法得到充分测试。由以上分析可知，自顶向下集成和自底向上集成测试策略各有利弊，需要根据代码视实际情况制定合适的测试方法，通常在测试复杂程度较高的代码时，结合自顶向下集成和自底向上集成方法的混合集成策略更能提高测试效率。

以 ATO 接口层模块发送 ATO 数据 (Write_Ato) 功能为例，介绍集成测试一般步骤。

①根据软件结构设计的要求对软件接口和功能，选定入口函数 (Write_Ato)，采用黑盒和灰盒测试技术设计测试用例。

ATO 接口层模块的功能是实现与 ATO 的通信功能，模块应满足以下功能要求：

功能 01：读取、校验 ATO 接口的数据；

功能 02：与 ATO 通信链路的冗余管理；

功能 03：解析 ATO 接口的数据；

功能 04：向 ATO 组帧发送数据。

根据 Write_Ato 的具体功能特点确定测试案例的输入数据以及覆盖的语句、分支路径。

②应根据被测系统功能的特点，指定合适的集成测试策略。

LDRA TestBed 可以生成清晰直观的调用关系图，它反映的是函数之间的层次关系，发送 ATO 数据函数静态调用关系如图 8 所示。

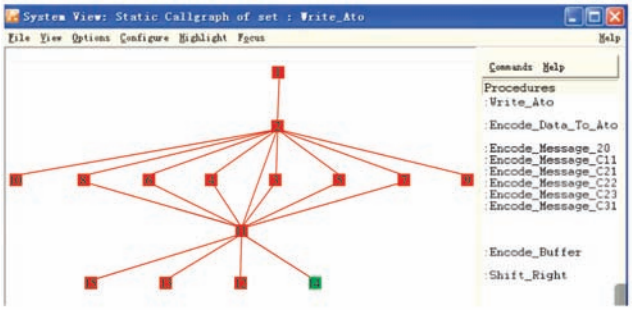

图 8　Write_Ato 函数静态调用关系图

分析代码调用关系和数据流，发现其函数调用层次达 5 层；调用函数编码 ATO 消息函数 (Encode_Data_To_Ato) 调用函数较多并且存在跨层调用现象；各模块的复杂程度不同。根据这些特点，选用“混合集成”的方法。

③将测试入口函数涉及的相关文件以集的方式分析，建立测试序列，输入测试数据执行测试案例验证覆盖率信息，测试结果如图 9。集成测试的基本步骤和单元测试相似，不同点在于单元测试中调用的相关函数只有底层涉及到硬件和系统函数可以打桩外，其他都尽量调用实体，确保集成的有效性。

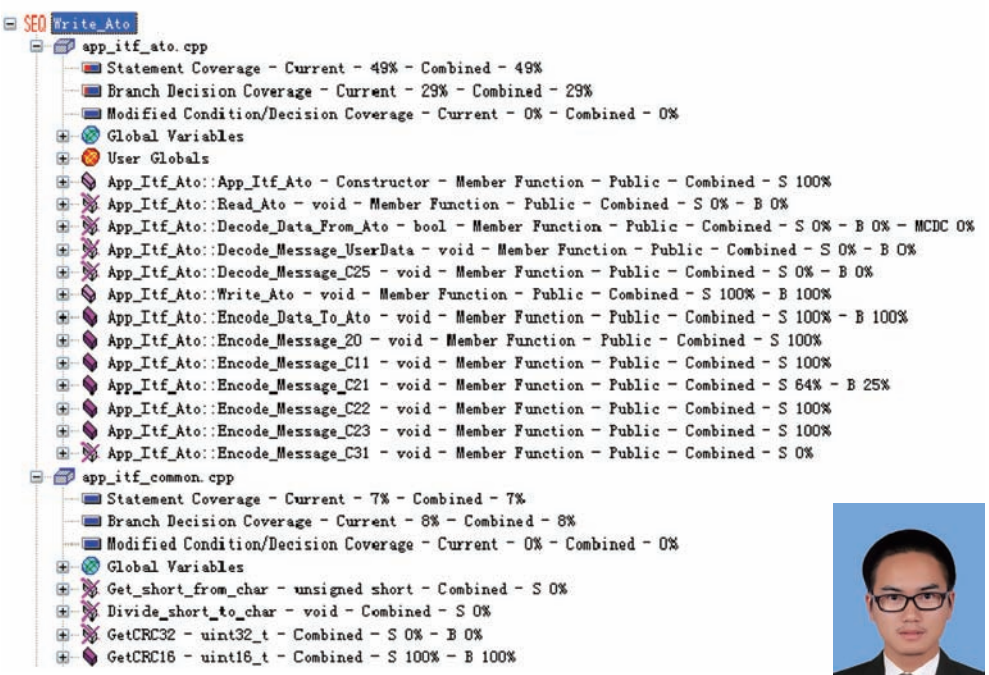

图 9　Write_Ato 集成测试覆盖率结果图

集成测试中的测试要点及经验有：

①在设计集成测试案例时，需要以软件设计的功能需求为测试对象，对功能函数接口数据要着重设计案例，主要验证函数模块间的接口一致性，同时可以提高测试人员对系统软件的熟悉程度。

②在测试某些特殊功能模块时，如果存在与其有调用关系但是还未开发完或不能直接调用实体代码 (例如与底层硬件相关的输入输出函数) 的情况，可以设定桩函数。

③对于具有高层控制调用功能或者复杂易错等特点的关键软件模块，应把这类测试工作提前安排。

5　总结

本文研究了 LDRA TestBed 自动化测试工具在软件测试主要活动中的应用。以 CTCS2-200C 扩展单元应用软件为例，着重介绍了工具在静态测试和动态测试时的使用方法和经验。参照 EN 50128:2011 标准，该方法提高了整个软件开发生命周期中的各项验证和确认活动效率，也为其他安全相关软件的安全评估中的测试、验证和确认活动提供了有价值的参考。

参考文献：

［1］EN 50129:2003　Railway applications-Communication, signalling and processing systems-Safety related electronic systems for signaling［S］.

［2］EN 50128: 2011　Railway applications-Communication, signaling and processing systems-Software for railway control and protection systems［S］.

［3］上海创景计算机系统有限公司 . TN_5_LDRA_Testbed 可行性报告 2.0［R］. 上海：上海创景计算机系统有限公司 .

［4］中国铁路总公司 . 城际铁路 CTCS2+ATO 列控系统暂行总体技术方案［S］. 北京：中国铁路总公司，2013.

作者简介：周永健（1989–），男，硕士，研究方向为列车自动控制。