



嵌入式单元测试技术

对嵌入式系统软件进行测试是一项很有挑战性的工作。因为嵌入式系统的软件开发平台和最终运行平台是完全不一样的两个平台，开发者不可能在其运行平台上像在桌面环境那样执行测试程序。这些差异可以表现在这几个方面。在软件开发的时候目标硬件平台可能还没有准备好，软件工程师在访问硬件时可能会很麻烦，在开发环境下模拟整个系统存在困难性等。

本文阐述了单元测试在嵌入式系统软件开发过程中的作用以及其如何帮助开发者解决上述问题。简要地讲，它提升了桩函数在宿主环境下或者仿真器中的作用。这使软件工程师在代码编写完毕之后就可以立即对其进行验证，即使此时目标硬件系统还没有准备好或暂时不能进行测试。这样，绝大部分的潜在在程序中的逻辑问题都能够在早期被发现，软件工程师就可以迅捷地修复这些问题，而目标平台上的测试就可以注重于检测软硬件接口方面的问题。

为何要对嵌入式系统软件进行单元测试？

单元测试是检查嵌入式系统软件缺陷的最有效的一种方法，更确切而言也即在宿主机环境下或在仿真器上进行的 API 测试。这能使测试尽早开始并且最大程度地降低了测试工作对目标硬件平台的依赖性。嵌入式系统软件的单元测试的一个前提是能够让软件独立于硬件进行测试，这是由桩函数对目标硬件平台的模拟而实现的。在这种情况下，代码的绝大多数功能性测试都能够独立于目标硬件系统进行测试。这对嵌入式系统开发者而言有以下主要好处。

1. 单元测试能让开发者在目标硬件平台准备好之前就开始测试周期，开发者可以直接在宿主开发环境下开始初始测试（而不需要目标硬件平台）。早期测试能够为开发团队发现并修复缺陷提供尽可能充足的时间。此外，早期测试还能将测试工作较平均地分配于产品开发的各个阶段中，从而避免将所有的测试工作留到产品发布之前才进行，为开发团队争取了充足的时间。
2. 单元测试采取一种“各个击破”的策略，允许用户将复杂的系统分为相对较简单的准独立系统进行测试。测试工具能够管理模块之间的关联性并使用桩函数来模拟这些模块的行为。

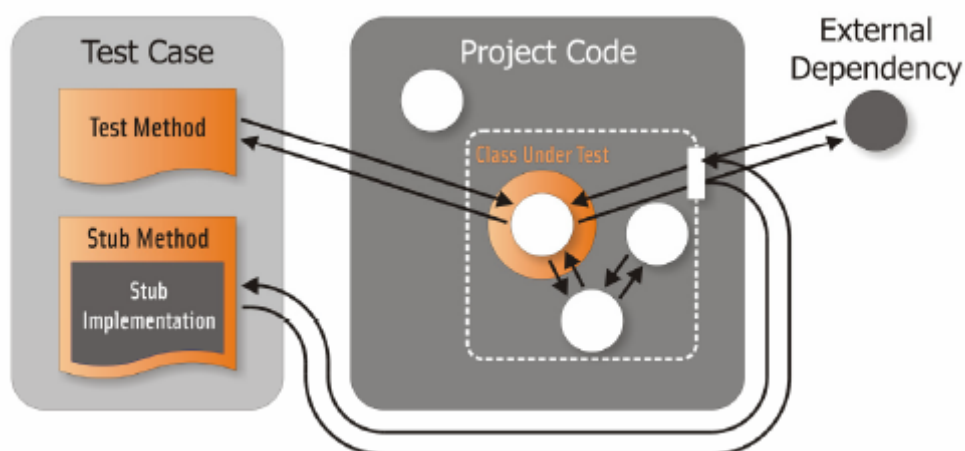


图 1：使用桩函数，开发者可以模拟真实环境和程序之间的相互影响对程序的某个模块进行测试，而不需要目标硬件平台以及暂时还未完成的其它代码模块。在测试环境中，桩函数充当了为待测模块提供外部关联性桥梁的作用。

在代码修改过程中保护代码完整性

复杂系统的开发者最大的烦恼就是不能确定对代码的修改是否改变或破坏了程序的既有功能性。为了打消这些顾虑，用户可以创建一个基准单元测试套件来捕捉代码的既有功能性。如果用户希望检查相对于基准单元测试套件的改变，只需要对修改过的代码定期地运行该测试套件即可。因为单元测试能够独立地测试系统的某些代码，所以这样一组回归测试套件可以持续地执行而无论目标硬件平台准备好与否。这种测试并不排斥对整个程序进行测试的分离回归测试套件。

这样的测试套件作为变更检测完全能够让用户确保如果无意间破坏了既有功能性能够被立即告知。当目标硬件平台准备好后，用户可以直接用宿主环境下的测试来验证代码是否会在实际的目标硬件平台上正确地运行。虽然代码在宿主环境下进行了回归测试，但是在目标硬件平台上还是需要进行一定的自动系统测试。

验证错误处理

由于消费类产品的错误处理可靠性需求的不同，所以系统测试情景变得更加复杂了。因为不能精确地预测其使用情况，所以系统的设计和测试都不能基于某个特定的情况。相反这些系统必须经过广泛地错误以及未预期输入的验证，以保证这些输入能够被正确地处理。使用含有桩函数的单元测试同样能够大大地简化错误测试过程。总的而言，在应用程序这一层级来测试错误条件是一件相当耗时的事，因为将程序置于“正确的错误状态”需要准备相对复杂的输入数据并且同时要求程序在其大量的可能状态中处于相应的状态。与之形成鲜明对比的是，使用“错误模拟”方法来对某些函数进行错误处理测试则要简单得多。

例如，测试一个含有输入数据错误处理机制的测试是相当简单的：

```
float signalToNoiseRatio (float signal, float noise, MODE mode)
{
    if (MODE_MEASUREMENT == mode)
    {
        ...
        if (signal < 0 || noise < 0)
        {
            handle_bad_data();
        }
    }
}
```

在这种情况下，在测试程序中对 `handle_bad_data()` 函数的调用是很简单的，因为测试该语句是否受程序输入数据控制是很简单的。

尽管如此，很多情况下控制语句并不直接受控于函数接口，而是取决于系统处于的特定状态，如下例所示。将系统置于该错误状态可能会相当复杂，甚至可能牵扯到让设备接口处于某些特定状态，所以测试用例中的这个条件就需要通过模拟来实现。

```
float shutDown ()
{
```

```
if (uploadingData())  
{  
    userMessage ("Cannot execute shutdown while uploading data");  
    recoverShutDown();  
}  
else  
{  
    // shut down indeed  
}  
}
```

使用支持“智能桩函数” (smart stubs) 的高级测试工具，对于复杂错误条件的测试比其它方法就简单了很多。“智能桩函数”允许通过原始函数的桩函数以及通过实现某些必要的特定功能的方法来执行代码。实际而言，当被测程序明显没有处于错误状态时，其相应待测函数将被调用以模拟错误实际发生的情况，这就是“错误模拟”的意义。在上述例子中，对错误处理函数 `uploadingData()` 的测试需要其桩函数以保证至少让其返回一次“真”。

一些值得考虑的问题

在宿主环境下进行测试可能意味着用以创建代码的编译器和目标系统编译器不同。如果交叉编译器提供商同时提供用在宿主环境下的编译器（例如 Green Hills Software 的本地编译器），则可以直接使用。如果没有，则可以使用支持大多数平台的 GNU Compiler Collection (GCC)。虽然保证代码在宿主编译器和目标编译器上的一致性会对维护成本有些许的提升，但是与早期测试对整个项目带来的好处相比却是很合算的。

单元测试不大可能发现因同步错误造成的应用程序级的错误条件或者与实际设备接口的错误。尽管如此，在嵌入式系统的开发过程中，单元测试能帮助开发者尽早地发现很多类错误，所以提高了系统整体开发效率并且消除了测试瓶颈。

使用 C++test 来自动化单元测试过程

用户可以使用 Parasoft C++test 来自动化对嵌入式系统软件的测试。

Parasoft C++test 是一套经广泛验证的最佳编码实践的自动化解决方案，它能有效地提高软件开发团队的开发效率以及软件的质量。C++test 能帮助用户进行编码策略增强、静态代码分析、全面代码走查以及单元和组件测试，从而为用户保证其 C 以及 C++ 代码按预期运行提供一个实际可行的方法。C++test 可以在桌面环境下的主流 IDE（包括 Wind River Workbench 以及 Eclipse）以及回归测试过程中的命令行中以批处理的方式运行。C++test 集成了 Parasoft 的 GRS 报告系统，它能提供一个基于 Web 交互界面的报表并为用户提供向下挖掘 (drill-down) 功能，基于 C++test 报告的这些结果，开发团队实时把握项目状态和趋势以及其它关键指标。

对于嵌入式以及交叉开发而言，C++test 可以在基于宿主环境和目标环境下执行代码分析以及数据流分析。在宿主环境中，开发者可以通过 C++test 的编码策略增强、静态代码分析、全面代码走查以及单元和组件测试模块来对代码进行“随时测试”式的验证以及回归测试。被测代码的外部依赖性被桩函数自动地取代，桩函数能真实地模拟硬件以及其它代码在实际运行中的表现。

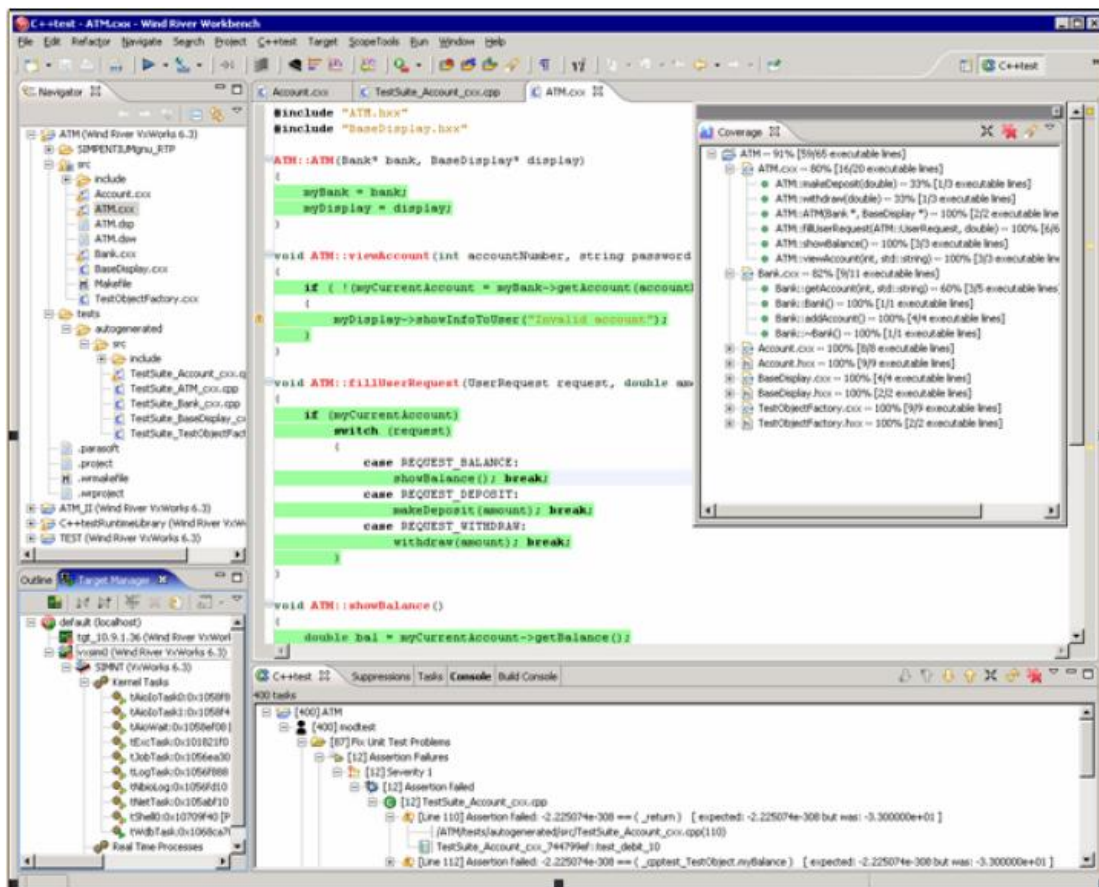


图 2: Wind River Workbench 中的 C++test 插件能为其用户提供一个简便的方法来使用 C++test 进行完整的代码分析和单元测试。

通过扩展的基于宿主环境的测试，C++test 能让用户在某个模块代码完成后立即对其进行测试，即使此时目标环境还未准备好或不能进行测试。这样，绝大部分的潜在程序中的逻辑问题都能够在早期被发现，从而让开发者修复这些错误变得十分迅捷，而在目标硬件平台上的测试则能更加专注于验证软硬件接口问题。此外，基于宿主环境的测试相对于目标系统而言更加容易自动运行和维护，这使得用户可以不用其它嵌入式开发工具就可以验证相对平台独立的代码正确性。

当开发者在仿真器或实际目标硬件平台上进行测试时，在宿主环境下生成和优化过的测试套件可以被重用来验证软件在目标硬件上的功能性。先前的桩函数这时可以用实际代码或者系统接口来代替从而完成完整系统的测试，而不需要修改测试代码。C++test 同时还提供一个内建功能来自动捕捉执行测试输出以及将其转变成后续回归测试时的“黄金”数据集。C++test 让整个测试流程自动进行，包括测试用例生成、交叉编译、部署、执行以及将测试结果（包括覆盖率指标）导入 GUI。测试可以通过 GUI 交互地进行或者通过命令行自动执行，以及以批处理的方式进行的回归测试。在交互模式下，用户可以对单个模块或者一组代码进行测试，从而让调试和验证都变得更简单。在批处理模式下，测试可以针对用户提供的代码或者根据文件名字或在磁盘中的位置来进行。

同时，C++test 还支持将其执行顺序进行完全的用户自定义。除了内建的自动化测试，用户还可以引入自定义测试脚本以及通过 shell 命令将测试工具根据代码具体结构和测试环境进行自定义。C++test 的运行库也能被自定义并且针对不同的目标操作系统进行交叉编译。这种无可比拟的灵活性能够任意实现其希望的测试流程而不需要预设工具的功能。

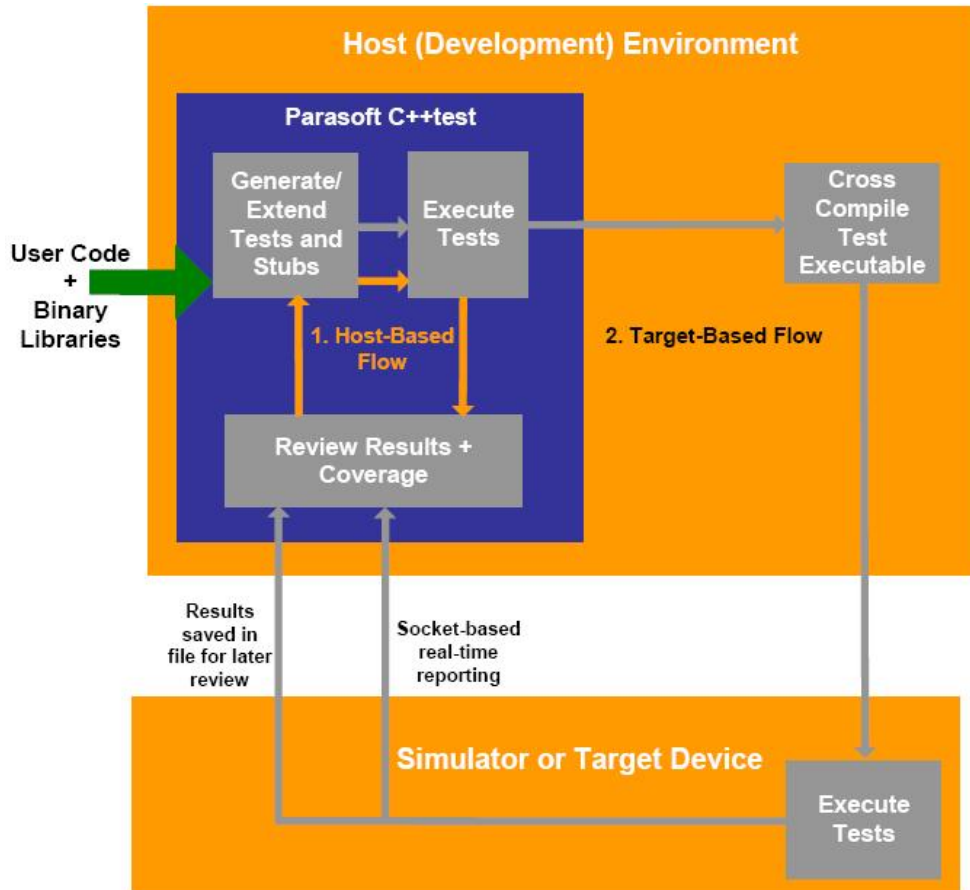


图 3: C++test 的可自定义工作流程让用户可以根据代码的构造来进行测试，然后使用这个测试套件在目标硬件环境下测试其功能性和可靠性。

嵌入式开发支持的单元测试

- 真正的单元（函数 / 类）和组件测试
- 自动生成 C 或 C++ 格式的测试套件
- 支持数据源
- 单一测试或者任意测试的任意组合方式的交互执行
- 捕捉实际测试结果后自动生成回归测试套件
- 宿主平台和目标平台的统一测试环境
- 测试流程完全可自定义并提供运行时支持
- 语句、代码块、分支 / 条件以及路径指标覆盖率分析
- 提供 HTML 和 XML 格式的测试结果报告
- 支持 GUI / 桌面环境以及命令行方式

支持的目标编译器

- Wind River GCC 3.4.x 和 DIAB 5.4+
- GCC 2.95.x – 4.1 交叉编译器
- Green Hills 4.0.x

如需了解更详尽地了解 Parasoft 的测试方案如何帮助用户对嵌入式系统进行自动测试，
请以下述方法联系 Parasoft。

联系 Parasoft

美国

101 E.Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

欧洲

法国： Tel: +33 (1) 64 89 26 00
英国： Tel: +44 (0) 1923 858005
德国： Tel: +49 89 4613323-0
Email: info-europe@parasoft.com

亚洲

Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

其它地区

请浏览 <http://www.paraosft.com/jsp/pr/contacts>