白皮书

保障嵌入式软件安全可靠的7个方法

使用静态分析和形式化方法

MathWorks®

Edsger Dijkstra，一位计算机科学先驱曾经说过：软件测试可以用于发现软件的缺陷，但是无法知道软件没有缺陷。然而，许多嵌入式项目错误地将没有不通过的测试作为软件质量的证据。针对该问题，本文提供一个特殊的方法来应对。

使用现代的静态分析工具，应用形式化的方法，开发团队可以“证明没有缺陷”，而无需集成系统或者在硬件上测试甚至运行代码。

Polyspace®静态分析软件提供形式化验证的方法，包括两个产品。

Polyspace Bug Finder™ 能够识别源代码中可能的运行时错误、数以百计的其他类型的缺陷、检查代码是否满足编码规范和安全规范以及生成代码的度量报告。

Polyspace Code Prover™ 能够证明单元代码和集成代码是否存在严重的运行时错误，无需测试用例，无需执行代码。Polyspace Code Prover 使用形式化的方法（抽象解释法），验证中考虑所有可能的输入、运行路径和变量取值，且没有漏报。

Polyspace通过了TÜV SÜD鉴定，能够用于一些需要遵循功能安全标准的开发流程，如ISO 26262, IEC 61508和IEC 62304。同时，Polyspace可作为DO-178B/C可被鉴定的工具。

使用Polyspace静态分析和形式化方法的七种形式让嵌入式软件安全

日产、空客、德尔福和其他的跨国高科技公司的开发团队总结了Polyspace的7个关键优势。

1. **及时反映给开发人员。**Polyspace提供详细的信息，如变量在代码具体某一行的范围，潜在的溢出及其发生的条件和死代码。这些信息可以帮助你强制遵循如MISRA®的编码规范。
2. **聚焦在单元测试策略。**通过证明在所有可能输入的情况下没有严重缺陷，Polyspace可以减少并指导单元测试开发工作
3. **变量同时访问缺陷检测。** Polyspace证明多线程应用程序中没有争用条件(race condition)，并且跟踪源代码的缺陷。
4. **数据流和控制流信息记录在案。** Polyspace提供详细的控制流和数据流信息，详尽而完善的语义分析。
5. **服从功能安全标准。**Polyspace使用错误查找，代码验证和标准检查来帮助识别和避免安全漏洞并遵守CERT C，ISO 17961和CWE等标准。它可以证明黑客利用的安全漏洞不存在，例如缓冲区溢出，非法指针解除引用和未初始化的变量。
6. **可作为鉴定标准的证物。**Polyspace通过了TÜV SÜD鉴定，能够用于如ISO 26262, IEC 61508和IEC 62304这些需要符合功能安全标准的开发流程。同时，Polyspace可作为DO-178B/C可被鉴定的工具。您可以获得包括形式化方法、控制流数据流和数据范围检查鉴定证物，而这些是其他工具无法提供的。
7. **和基于模型的设计有机结合。**Polyspace是基于模型的设计工具链的一部分，提供与Simulink®和Stateflow®模型的追溯。

关于测试和形式化验证的说明

```
1  int new_position(int sensor_pos1, int sensor_pos2)
2  {
3  int actuator_position;
4  int x, y, tmp, magnitude;
5
6  actuator_position = 2; /* default */
7  tmp = 0;               /* values */
8  magnitude = sensor_pos1 / 100;
9  y = magnitude + 5;
10
11 while (actuator_position < 10)
12     {
13     actuator_position++;
14     tmp += sensor_pos2 / 100;
15     y += 3;
16     }
17 if ((3*magnitude + 100) > 43)
18     {
19     magnitude++;
20     x = actuator_position;
21     actuator_position = x / (x - y);
22     }
23 return actuator_position*magnitude + tmp; /* new value */
24 }
25
```

示例代码

```
Source
Results Statistics | where_are_errors.c | __polyspace_main.c

1  int new_position(int sensor_pos1, int sensor_pos2)
2  {
3  int actuator_position;
4  int x, y, tmp, magnitude;
5
6  actuator_position = 2; /* default */
7  tmp = 0;               /* values */
8  magnitude = sensor_pos1 / 100;
9  y = magnitude + 5;
10
11 while (actuator_position < 10)
12         {
13     actuator_position++;
14     tmp += sensor_pos2 / 100;
15     y += 3;
16     }
17 if ((3*magnitude + 100) > 43)
18     {
19     magnitude++;
20     x = actuator_position;
21         actuator_position = x / (x - y);
22     }
23 return actuator_position*magnit        value */
24 }

operator / on type int 32
  left: 10
  right: [-21474855 .. -1]
  result: [-10 .. 0]

Source | Data Range Configuration
```

Polyspace 代码分析结果

示例中的函数有两个输入：您如何人工审查这段简短的代码？如何确保这段代码没有缺陷？

测试法。如果考虑对代码进行测试，则只需要两个测试用例即可完成修改条件/决策覆盖范围(MCDC)。这两个测试用例足以确保健壮吗？穷举测试则需要考虑每个输入的所有可能值。

静态分析法。静态代码分析通常用于补充测试用例。静态分析可以用于自动执行许多手动验证任务，例如强制符合编码规范，也可以根据分析判断缺陷。

在这个例子中，静态分析工具可以通过检查变量的许多可能值来扩大测试覆盖范围。但这也有副作用，它会对不可能的值产生许多错误警告。

形式化方法。 Polyspace静态分析产品涵盖了上面的基本方法，但也使用形式化方法来验证运行时行为，并证明没有错误。

Polyspace使用抽象解释法来证明软件在所有运行条件下都是安全的。它还提供代码中所有变量的范围信息，并与控制流和数据流映射。

示例的解决方案。在示例代码中，Polyspace已经证明代码没有严重的运行时错误。我们看看第21行，其中可能怀疑被除零。使用形式化方法，Polyspace已经确定变量x和y的范围使证明x和y永远不能相等，因此在此行上不会出现除零。

MathWorks®

让嵌入式软件更安全的7种方法

1. 及时反馈给开发人员

在嵌入式系统中，大量的运行时缺陷是在编码阶段引入了。在代码审查期间，我们经常漏掉这些错误，并且传统的静态分析无法检测到这些错误。随着开发的推进带入到后面的流程中，经常在硬件测试期间，这些问题被检测到，导致返工和开发滞后。

Green: reliable
safe pointer access

Red: faulty
out of bounds error

Gray: dead
unreachable code

Orange: unproven
may be unsafe for some conditions

Purple: violation
MISRA-C/C++ or JSF++ code rules

Range data
tool tip

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
      *p = 0;
      p++;
    }

  if (get_bus_status() > 0) {
    if (get_oil_pressure() > 0) {
      *p = 5;
    } else {
      i++;
    }
  }

  i = get_bus_status();

  if (i >= 0) {
    *(p - i) = 10;
  }
}
```

variable 'I' (int32): [0 .. 99]
assignment of 'I' (int32): [1 .. 100]

Polyspace使用代码着色的方式表示运行时错误

Polyspace通过静态分析的方法来应对该类问题，Polyspace包含详细的信息，潜在的溢出及其发生的条件和死代码。这些信息可以帮助你强制遵循如MISRA的编码规范。通过应用静态分析和形式化方法，开发人员可以在编码阶段看到并修复运行时问题，防止将问题带到后面的开发过程中。

> “动态测试很少能让我们检测到错误的症状？，*Polyspace*代码验证指出问题的根本原因，大大的提高了我们的调试效率。”
>
> — Frédéric Retailleau, Delphi Diesel Systems

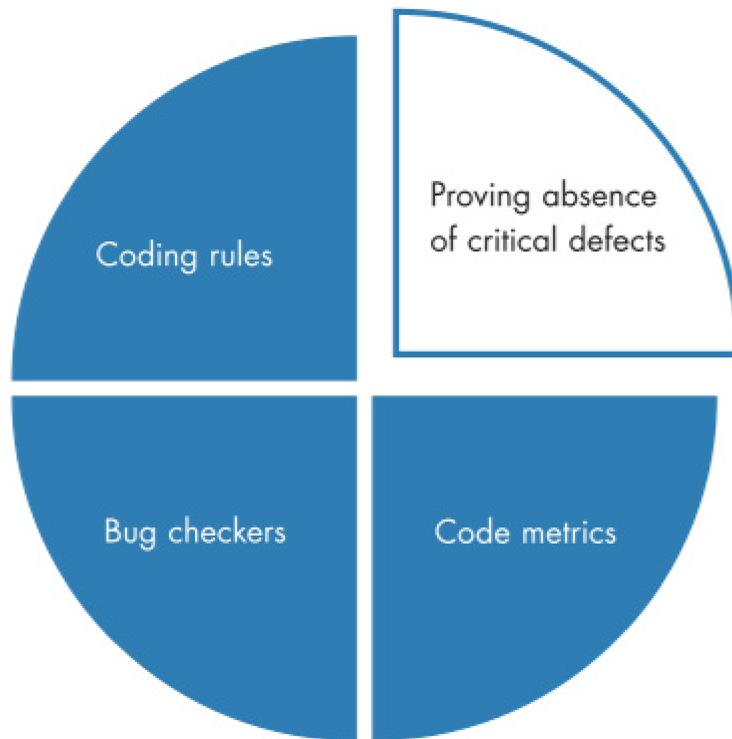
MathWorks®

2. 聚焦在单元测试策略

单元测试可以验证模块的健壮性，但是需要编写能够显示问题的测试用例。手动编写单元测试用例是比较困难的，为了提高代码的覆盖度增加测试用例，但仍然可能漏掉一些验证的错误。

Polyspace能够证明代码没有运行时错误，减少健壮性测试的需求，识别那些需要进一步分析或测试的特定的未予证明的运算。

通过证明在所有可能输入的情况下没有严重缺陷，Polyspace可以减少并指导单元测试开发工作。



> “*Polyspace Code Prover*能识别出我们手写代码的问题，同样能够找出没有任何问题的代码，还有那些要格外注意的代码。分析结果使得我们在形式化审查的过程中开展了代码的目标评估”
>
> — Dr. Karen Gundy-Burlet, NASA Ames Research Center

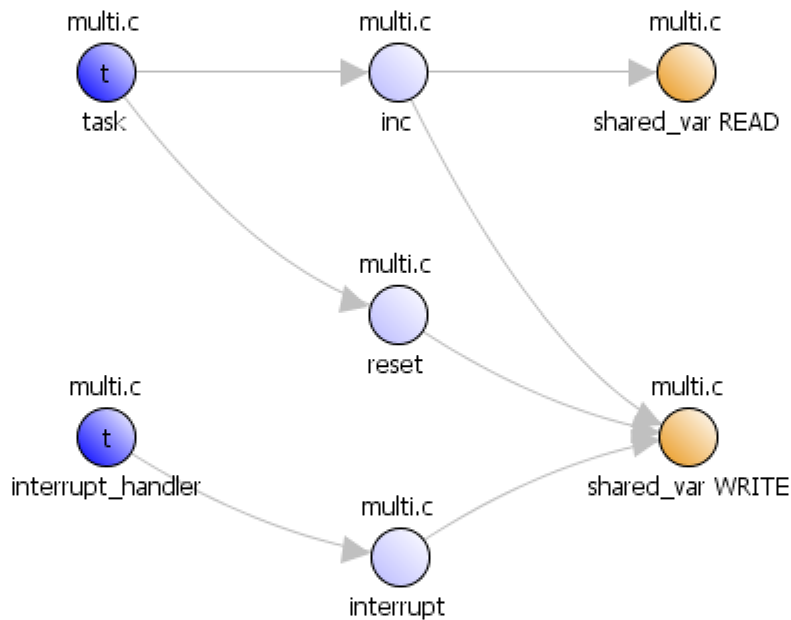
MathWorks®

3. 变量同时访问缺陷检测

Polyspace证明多线程应用程序中没有争用条件和其他并发问题，并且会向问题根源跟踪缺陷。

静态内存问题，并发问题和难以发现的运行时错误等复杂缺陷很难被发现，并且可能会进入生产阶段。它们需要正确的测试用例并且难以重现。

在代码检入代码库之前，使用Polyspace可以检测代码中的可能缺陷，避免测试和调试隐藏的错误。通过在调试过程中逐步执行事件跟踪，Polyspace可减少复杂缺陷的调试工作。



> “*Polyspace*不但能找到哪些运算会产生运行时错误，而且能确保不可能发生运行时的操作，不管这些运算的条件是什么。这些可以在单元测试之前的，也就是编码阶段完成。这对我们的供应商有极大的价值。”
>
> — Mitsuhiko Kikuchi, Nissan

MathWorks®

4. 数据流和控制流信息记录在案

Polyspace提供详细的控制流和数据流信息，详尽而完善的语义分析。

Polyspace以函数调用树、全局数据字典和变量数据范围三种形式呈现给用户，帮助用户对复杂的运行时错误进行调试。

Variable Access

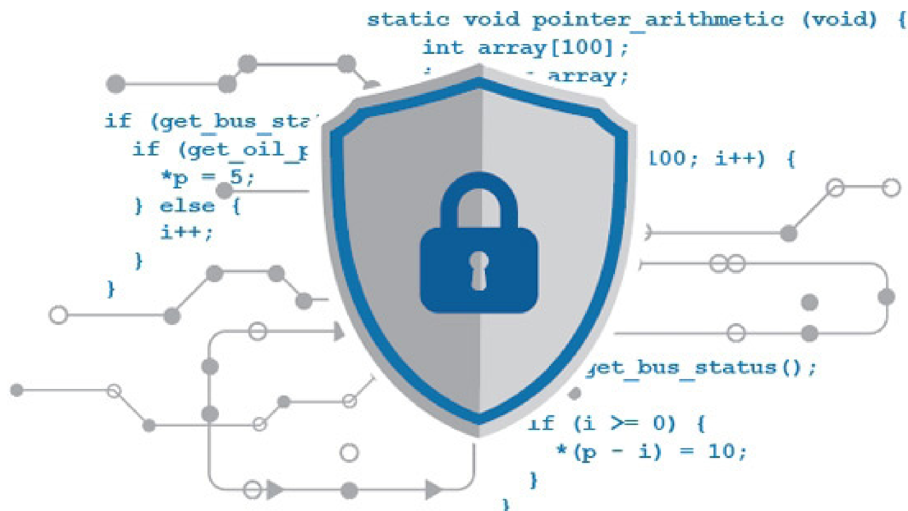
Variables	Values	# Reads	# Writes
tasks1.PowerLevel	[-214748	4	3
main.main	-10000		
tasks1._init_globals	0		
tasks2.Increase_PowerLevel	[-214748		
tasks1.orderregulate	[-214748		
tasks2.Increase_PowerLevel	[-214748		

> "*Polyspace*的解决方案独一无二，它发现运行时错误，而无需执行代码，另外一个优势是，其进行穷举分析"
>
> — Airbus

5. 服从功能安全标准

随着嵌入式软件的关联度越来越高，如ADAS，安全，尤其是涉及到人身安全，成为一个关注的重点。整体安全方法涵盖整个开发周期，因此需要额外的验证和确认。

Polyspace通过查找软件缺陷，代码证明和标准检查来帮助识别和避免安全漏洞，符合安全规范，诸如CERT C，ISO17961和CWE。 他能够证明没有安全漏洞，这些漏洞可能被黑客利用攻击系统，如缓存溢出，非法的指针引用和解除引用、未初始化的变量。



MathWorks

6. 鉴定标准的证物

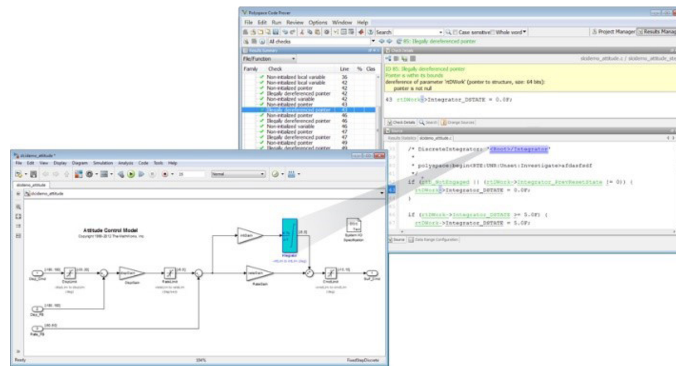
Polyspace通过了TÜV SÜD鉴定，能够用于如ISO 26262, IEC 61508和IEC 62304这些需要符合功能安全标准的开发流程。同时，Polyspace可作为DO-178B/C可被鉴定的工具。您可以获得包括形式化方法、控制流数据流和数据范围检查鉴定证物，而这些是其他工具无法提供的。

*ISO 26262*的鉴定需求表格

> “在我们获得*FDA*的上市批准前，我们已尽最大努力证明代码正确性和确保代码质量，而*Polyspace Code Prover*是我们努力证明的核心。”
>
> — Lars Schiemanck, Miracor Medical Systems

7. 和基于模型的设计有机结合。

Polyspace是基于模型的设计工具链的一部分，提供与Simulink和Stateflow模型的追溯。



> 案例：*Alenia Aermacchi*使用*Polyspace*检查代码的运行时错误，确保符合*MISRA C*编码标准，并生成了鉴定的证物。 他们使用*DO Qualification Kit*对*Polyspace*代码验证器和*Simulink*验证产品进行了*DO-178*鉴定

你可以对从Simulink模型和dSPACE® TargetLink®模块生成的代码进行静态分析。你可以在Simulink中启动分析并将分析结果追溯的模型中。

MathWorks®

了解更多

*Solar Impulse*使用*Polyspace*为太阳能飞机进行静态分析

使用Polyspace帮助Solar Impulse更早、更快的查找和消除问题，节约了1-2工程师年的开发时间，并满足DO-178目标，确保起符合性。

```
Source
where_are_the_errors-orange.c ×
1  int where_are_the_errors(int sensor_pos1, int sensor_pos2)
2
3  int actuator_
4
5  actuator_position = 2; /* default */
6  tmp = 0;               /* values */
7  magnitude = sensor_pos1 / 100;
8  y = magnitude + 5;
9
10 while (actuator_position < 10)
11   {
12     actuator_position++;
```

破除关于静态分析的误解

破除关于静态分析的误解。这些说法包括"我不需要静态分析，因为我做了足够的测试"，或者"只有需要通过认证时才需要进行静态分析"。

申请试用 | 与*MathWorks* 专家联系

© 2019 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.