

# 借助机器学习实现仿真回归结果调试自动化

## 作者

Rob van Blommestein  
产品营销

## 背景介绍

在半导体验证领域，很少有什么进步的影响力能够与为仿真测试套件引入回归验证相媲美。曾几何时，几乎就不存在验证的概念。人们只寄希望于开发者能对他们的设计进行一些临时测试，通常也就是手动应用一些输入，运行一个简短的仿真，然后看看产生的波形如何。随着设计的规模不断扩大，复杂性不断增加，建立正式的验证流程变得很重要。于是，验证工程便作为一门独立于设计的学科应运而生。那时，验证开发者会为设计的特定功能手动编写脚本测试，然后在仿真中运行，并对故障进行调试。验证开发者会在验证计划中罗列出相关功能，并在测试功能时进行核对。当所有功能都检查完毕后，芯片便会进行流片。

之所以引入回归验证是因为每个测试只运行一次是不够的。验证团队发现，有些测试之前是通过的，但在项目后期重新运行时却经常会失败。这种情况是多种原因导致的。首先，为了加快进度和缩短产品上市时间（TTM），设计和验证工作会交错进行。后来加入的设计部分往往会改变之前测试过的设计的行为。其次，如果在修复某个设计错误时引入了新的错误或引发了连锁反应，也会导致测试失败。另外，在实现设计（综合、布局布线）或为设计搭载可测性设计（DFT）功能时对设计进行精简，往往也会产生新的错误。最后，在整个项目过程中，芯片规格本身发生多次变化，也会导致一些测试过时或导致设计团队与验证团队不同步。

由于以上种种原因，验证开发者会将所有通过的测试收集到一个测试套件中，并定期以回归运行方式运行这些测试。回归运行通常在夜间有空闲的仿真和计算资源时执行。直到今天，情况依然如此。虽然受约束的随机激励生成已经实现了自动化，并减少了测试创建过程中的手动工作，但这些测试仍然会被收集到套件中，并以回归方式进行仿真。多数情况下，一个项目每次回归运行都至少有几个以前通过的测试会失败。有时还会出现所有测试都失败的情况，特别是当验证环境或测试平台发生重大改变时。这些故障基本上都需要手动调试，会对资源和上市时间（TTM）造成很大的影响，代价非常高。本白皮书提出了一个新的解决方案：使用人工智能（AI）和机器学习（ML）技术来实现回归结果调试自动化。

# 回归调试面临的挑战

在编写新的测试或修改约束以生成新的随机测试时，验证开发者都预料到会有一些手动调试工作。任何设计都有很多缺陷需要被发现和修复，因此许多新的测试第一次运行通常会失败，因为它们发现了设计缺陷。当然，验证开发者也会像其他开发者一样犯错，所以很多测试会因为测试或测试平台中的错误而失败。找出故障的原因并进行修复需要手动完成，但这也正是验证的意义所在。想办法从这种挑战中找到乐趣并巧妙地诊断出错误，是验证开发者工作的一部分。

回归运行中的故障则完全是另一回事。新测试通常会先在本地“沙盒”中调试，只有在通过后会添加到项目回归套件中。理想情况下，测试只要添加到回归中，就意味着验证团队已经确认测试通过。不过，这种情况从未发生过，但验证团队还是希望回归失败不常发生，因为这些失败会打断创建新测试的进展，而且调试起来也不那么有趣。从某种程度上说，每一次回归失败都会让人失望，因为这意味着项目工作进度会延后，并且还得重新审视验证计划中之前认为已经完成的部分。对于验证开发者来说，检查夜间回归运行的结果并发现故障是非常令人沮丧的。

不幸的是，由于前面提到的种种原因，回归失败屡见不鲜。大多数情况下，项目出现故障很正常，不出错才不正常。如图1所示，回归失败调试通常是一个手动过程。在签核设计和验证代码的最新变更后，会运行回归，这时会有一些测试失败。成百上千个测试失败并不罕见，这给验证开发者带来了艰巨的任务。首先，他们需要手动检查仿真日志文件并分析问题出在哪里。通常，他们可以根据报告的错误类型，对故障进行分类，并将其划归到不同的“箱”中。然后，他们需要对这些“分箱”进行分类，以确定问题更可能在于设计还是测试平台，以及哪些模块或组件是主要的怀疑对象。

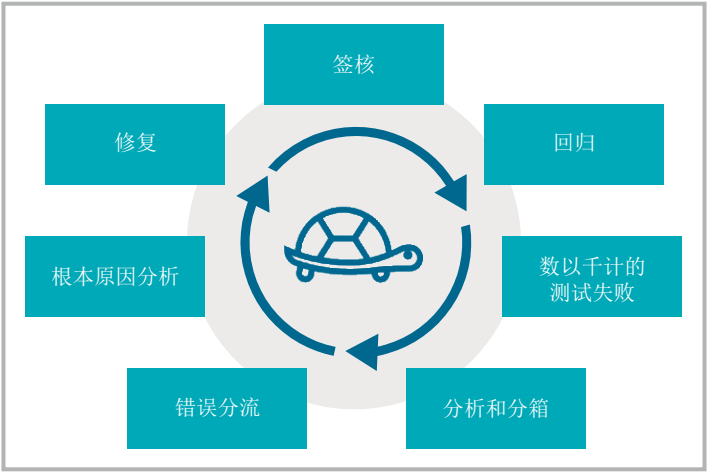


图1：包含手动操作的回归测试循环

这个时候，调试任务会转交给负责相关可疑代码的验证开发者或开发者。后者会进行根本原因分析（RCA），试图查明引起测试失败的实际错误。这个过程是手动的，而且要针对所有失败的回归测试的所有日志进行反复分析。这会消耗宝贵的项目时间，占用昂贵的资源，并且由于手动分析和分流的性质，因此很容易出错。分箱不当意味着同一错误导致的多个故障有时会被多次分流，而不是仅仅一次。手动分流意味着，在最终找到并修复根本原因之前，一个故障经常会被转给多名设计或验证开发者。芯片开发团队一直迫切希望能找到一种更好的方法来管理和调试回归。

## 机器学习前来解围

直到最近，才有了技术可以自动分析、分箱、分流、探查和发现回归失败的根本原因。机器学习是这些能力的关键促成因素，其中的原因并不难理解。手动回归调试在很大程度上依赖于验证开发者的经验。随着时间的推移，验证开发者会形成一种直觉，知道会发生什么类型的故障以及如何对故障进行正确分箱。多年的分流工作经验能帮助他们更准确地判断出最可能的故障源，然后将其分配给正确的设计和验证开发者，以进行根本原因的分析和修复。而根据从项目内数千次回归运行中收集到的大量信息，机器学习能够通过人工智能自动获得这种经验，并用它来提高调试的速度和准确性。图2显示了机器学习技术如何让回归循环中的三个阶段加速并实现自动化。

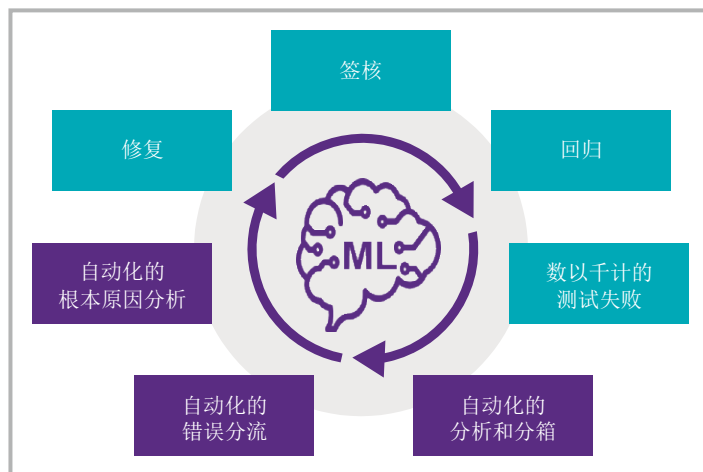


图2：机器学习辅助的回归测试循环

现如今已经有一个解决方案可以实现这种更高效的循环。新思科技Verdi®自动调试系统中的回归调试自动化（RDA）功能可以自动发现回归失败的根本原因。RDA利用机器学习对原始回归失败进行分类和分析，并找出设计和测试平台中故障的根本原因。RDA侧重于使回归日志分析、分箱、分流和根本原因分析实现自动化，从而减少典型回归流程中繁琐的手动工作。RDA自动化让用户能够比以往更快地发现、理解和修复错误，整体调试效率提高了2倍或以上。RDA的整体流程如图3所示。

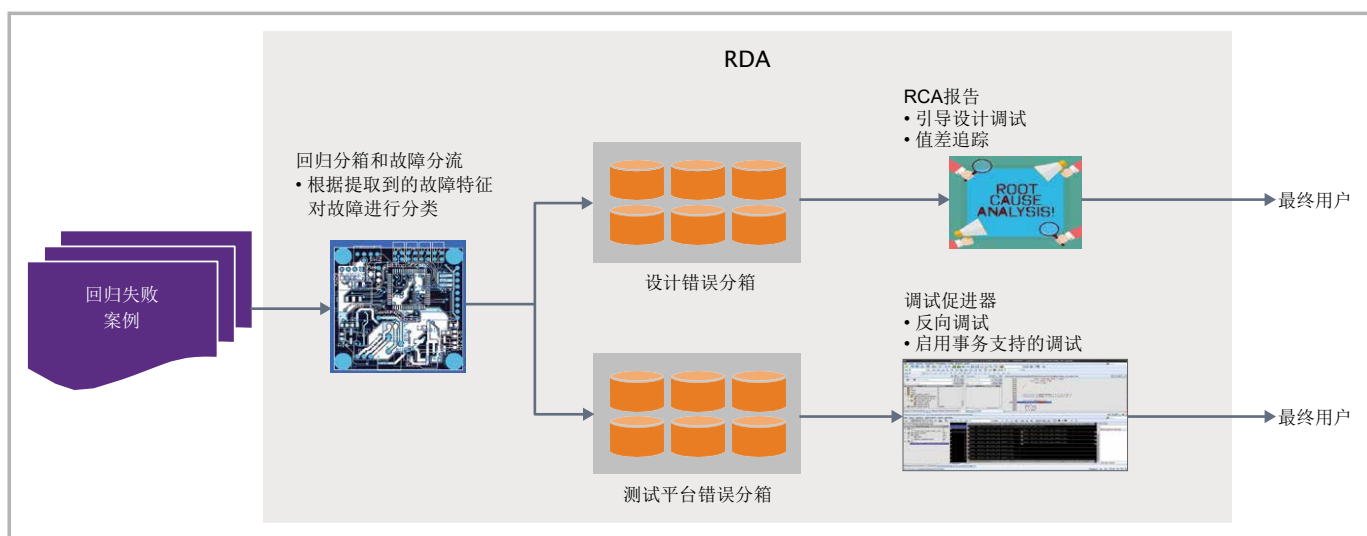


图3：Verdi RDA的整体流程

回归失败的传统分箱和分流的手动操作实现了自动化，并通常在回归后的夜间运行。因此，调试环境会在夜间设置，而不需要任何的用户操作或干预。对于任何追溯到设计的错误，Verdi RDA会自动执行根本原因分析，并向开发者提供相关报告，使得找出确切的故障原因变得更容易。并且，报告中会指出哪些信号在测试运行通过和失败之间具有不同的值，这样一来，开发者便能够将调试工作集中在设计的正确部分。而对于测试平台代码中的错误，Verdi RDA提供了事务感知调试和在仿真时间线中前后移动的能力，使验证团队能够更轻松地理解测试失败的原因并解决问题。

## 解决方案的组成部分

Verdi RDA整合了许多强大的科技和技术，旨在实现回归调试自动化并加快调试速度。该流程的第一步是收集回归运行的各种数据，包括仿真日志文件、值变转储（FSDB）文件，以及设计和通用验证方法学（UVM）测试平台的编译仿真数据库。收集到的数据会输入回归分箱应用程序，该程序会分析故障的日志文件，并将这些故障划归到相似的类别。这一步使用无监督机器学习来厘清验证日志故障之间的关系，并对结果进行分箱。它利用基于UVM的信息传递、用户定义的规则、验证知识产权（VIP）日志分箱和基于CPU的设计（指令集）分箱。其设置很简单，因此也很容易使用。结果显示，该流程的准确率达到90%，可减少整体分流时间。如图4所示，结果会显示在Verdi的RCA管理器中，供设计和验证开发者审查。

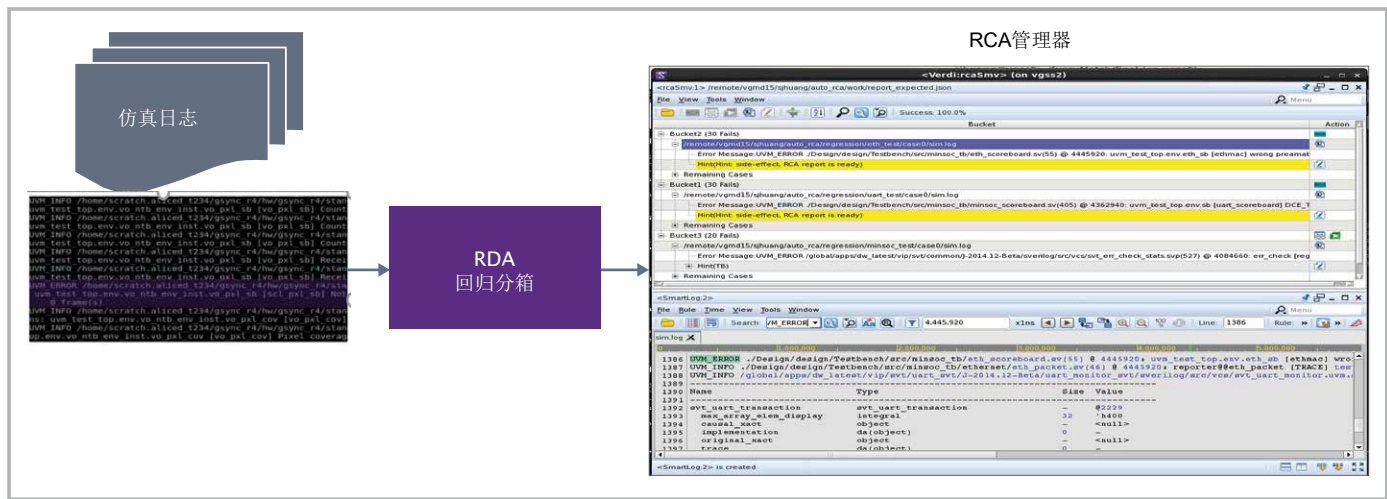


图4: Verdi RDA中基于ML的回归分箱

分箱后，RDA会进行故障分析和分流。它会对故障进行分箱，并根据故障的特征，判断问题是来自于被测设计（DUT）还是仿真测试平台。Verdi RDA之所以这么高效，其中一个原因是它采用了多种技术来找出设计和测试平台中故障的根本原因。DUTRCA方法从日志文件中识别故障，然后比较通过测试和失败测试的信号值，以找出测试错误附近不同的故障点。DUTRCA使用基于时间的回滚机制及其TraceDiff功能来自动缩小DUT中错误的原因范围。如图5所示，Verdi能够直观地显示RCA路径以及设计中信号值的变化。

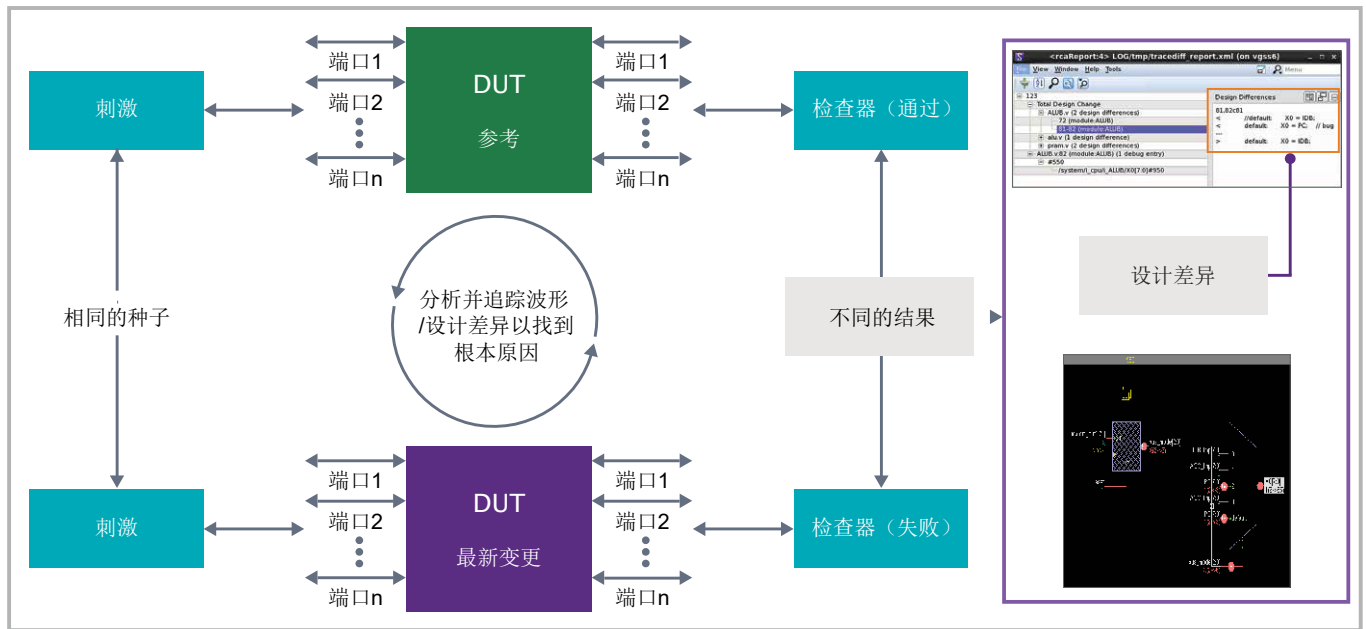


图5：比较DUTRCA中的信号值

为了找出测试平台故障的根本原因，调试促进器会自动收集每个故障箱的调试数据。然后，它利用协议分析器来协助进行调试，并显示事务和相关详情，以及反向调试能力，以便及时查看问题的来源。如图6所示，失败的测试会自动在仿真中重新运行，同时会启用反向调试和其他关键调试功能。调试促进器会在仿真过程中自动捕捉检查点，基本上就是测试失败时环境的全部状态。另外还会捕获其他重要的检查点。当RCA结果出现在Verdi的交互模式中后，就可以使用这些检查点。Verdi的调试功能（包括反向调试）可以快速分析测试平台中已确定故障的根本原因。

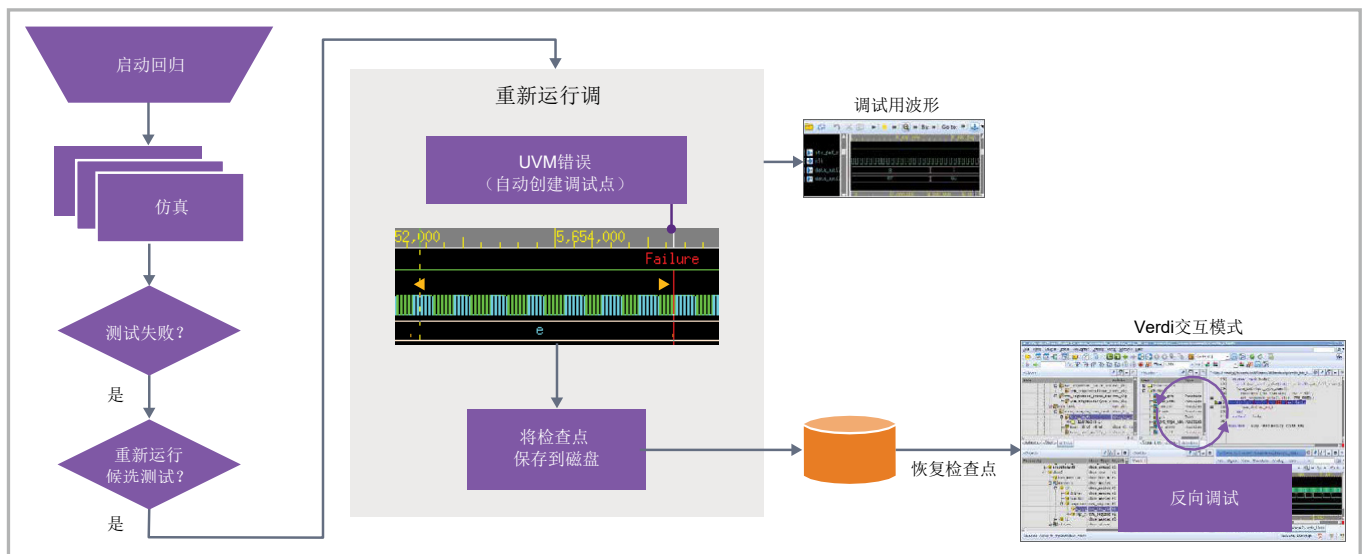


图6：使用Verdi中的检查点

Verdi RDA包含一些很有用的功能，能减少用户必须分析的未知（X）值相关的故障数量。这很重要，因为众所周知，X值非常难以调试。它们通常涵盖了许多周期和逻辑层次，也带来了许多具有单一根本原因的逻辑扇出锥。这时会用到一项技术——XRCA，该技术用于分析单一和多个X路径以找到根本原因。如果该技术分析得出多条路径来源于同一个根本原因，那么这些路径就会被归入同一组。如图7所示，这些结果会记录在报告中，该报告会显示在Verdi中，供开发者进行分析和理解，以便快速修复问题。XRCA会自动扫描FSDb文件中的X信号，并能处理大量这类信号，以减少用户的调试时间。

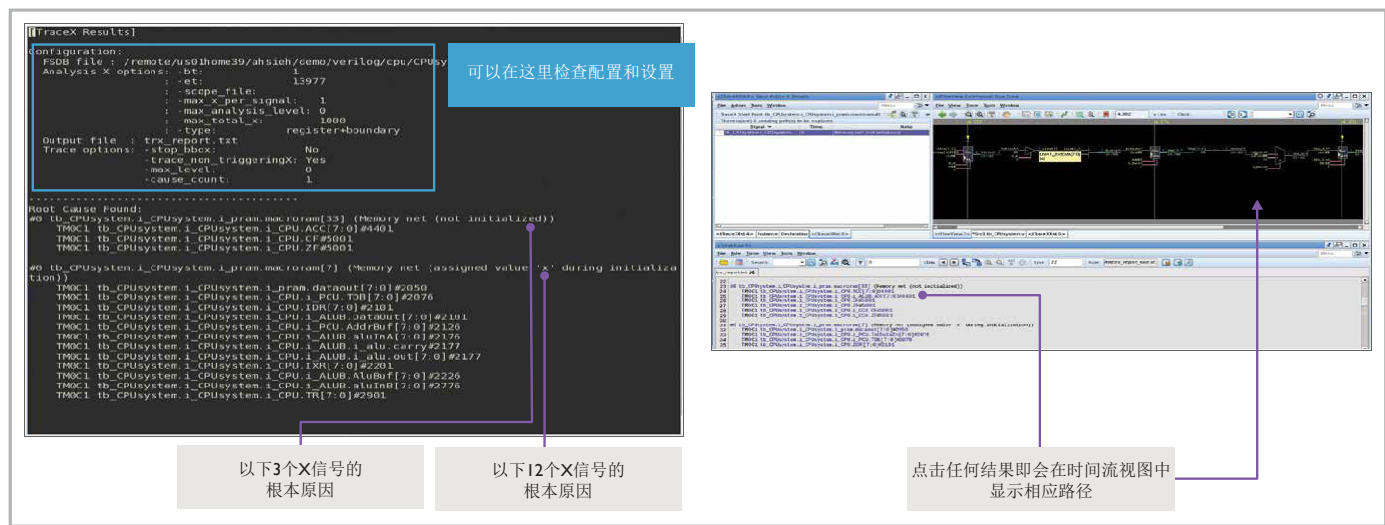


图7：借助XRCA减少未知状态导致的故障

XRCA还解决了X状态的悲观处理问题，这是另一个导致难以对未知相关故障进行调试的问题。仿真假设门的任何未知输入都应传播到其输出。这可能是悲观处理，在整个设计中传播“false” X值。图8中所示的电路就是一个简单的例子。一个信号及其补码都被输入一个“或”门。如果信号在仿真中是未知的，一个X值就会传播到该门的输出。但是，由于“或”功能的输入只能是0或1，因此它的输出始终为1。从调试工作中排除这种故障对开发者非常有帮助。XRCA包括一个形式化引擎，该引擎可以从真实的X值中过滤出这些类型的情况。它会分开报告这两种情况，所以两者都可以在Verdi中审查，但它只对实际X值执行完整的RCA，从而增加分析的吞吐量。

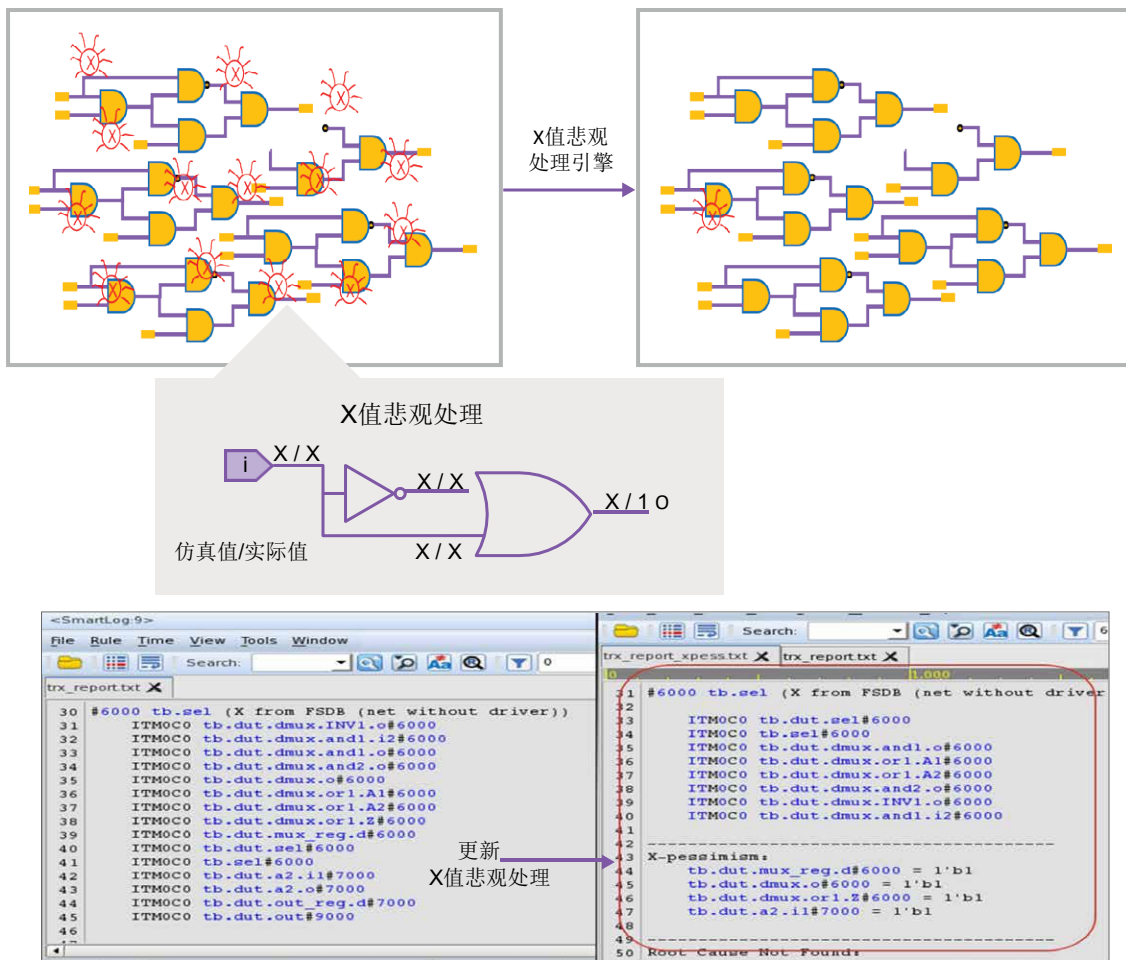


图8：借助XRCA减少X值悲观处理

## 总结

Verdi中的回归调试自动化功能为大型仿真回归运行结果的调试问题提供了出色的解决方案。而自动回归分箱功能几乎无需设置，使验证团队从检查成百上千个故障的手动任务中解放出来。自动分流功能能够将设计故障和测试平台故障分开，而根本原因分析功能可以找出最可能的错误来源。调试促进器会自动重新运行失败的测试，并提供先进的可视化功能来发现和修复错误。Verdi RDA为调试每一个失败的测试节省了大量的时间和精力，同时也大大减少了此类测试的数量。因此，该解决方案能够最大限度地提高回归验证的利用率，让开发者可以将精力集中在真正的调试工作上，而不是用于完成可实现自动化的任务上，同时可以缩短调试过程的周转时间（TAT），并将芯片项目的整体调试回归工作减半。