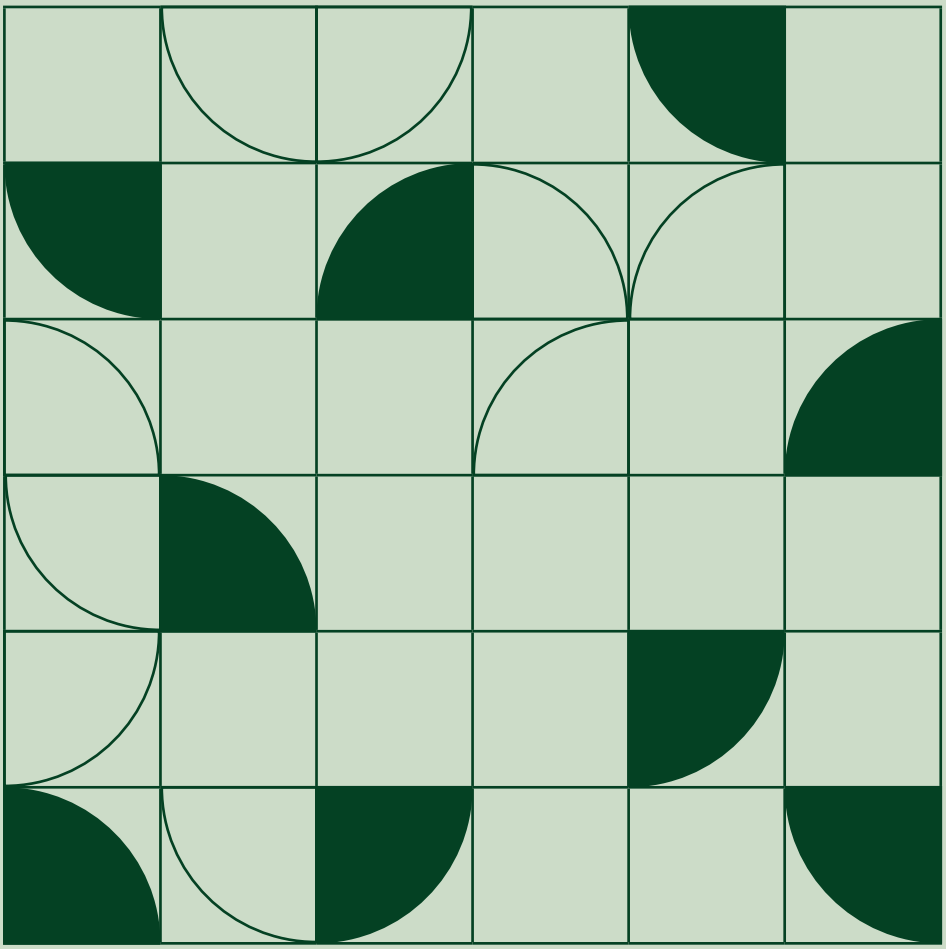
硬件和测量抽象层

Grant Gothing，Bloomy Controls公司ATE研发经理



引言

从初始规划到软硬件开发再到最终集成，自动化测试设备(ATE)的设计和开发过程面临诸多挑战。在该过程的每个阶段，更改都变得日益困难，且实现成本日益增加。此外，由于在开发周期中软件开发通常在硬件开发之后进行，因此许多开放式项目都留给软件工程师去解决。良好的规划可以大大降低常见的风险，但无法预防所有问题，尤其是测试开发周期非常短的情况，最终集成阶段可能会出现许多问题。人们常常认为软件比硬件更具灵活性，因此往往会形成“用软件解决就好了”这一误解。但是，硬件和软件是密不可分的，大部分问题通常需要对两者同时进行更新。这不会随着初始部署而结束，而是会持续出现在系统的整个生命周期中。

随着产品变得越来越复杂，测试它们所需的系统也越来越复杂。由于ATE仪器成本越来越受到重视，将仪器复用于多个产品的能力就显得非常必要。而且，由于开发时间不断缩短，工程师需要并行开发软件和硬件，但开发需求通常并不明确。另外，测试系统部署之后，由于产品生命周期较长，意味着仪器故障或淘汰以及产品和测试需求的变更可能会给测试设备带来更多挑战。因此，模块化、灵活性和可扩展性对于成功开发自动化功能测试系统至关重要。

从硬件的角度来看，这通常通过模块化仪器和由可互换测试连接件进行的互连来实现。但是如何使测试软件像硬件一样具有适应性呢？完成这个任务的设计方法有多种，硬件抽象层(HAL)和测量抽象层(MAL)是其中最有效的两种。与在测试序列中采用设备特定的代码模块不同，抽象层可将测量类型和仪器特定的驱动程序从测试序列中分离出来。由于测试程序通常根据所使用的仪器类型(例如电源、数字万用表[DMM]、模拟输出和继电器)来定义，而不是根据特定仪器进行定义，因此采用抽象层会使得测试序列更快开发，更易于维护，更适应新的仪器和要求。使用硬件抽象来将软件和硬件分开，硬件和软件工程师便能够并行工作，从而缩短开发时间。开发用于序列和底层代码实现的通用API，可帮助系统架构师维护通用函数库，进而实现标准化和可复用性。这使得测试开发人员能够专注于单个待测单元(UUT)序列的开发，减少编写底层代码所需的时间。

ATE软件挑战

开发	维护
开发周期紧迫 需求不明确 测试程序不断更新 硬件设计完成之前就要开始软件开发 软件和硬件工程师分离	产品生命周期长 ■ 仪器故障或过时 ■ 仪器变更 产品更新 ■ 测试程序变更 ■ 需要采用新硬件 制造工程师通常不是最初的测试开发人员

软件抽象的优势

开发	维护
软硬件分离 序列开发与代码(驱动程序)开发分离 提供用于仪器的通用API 优化代码复用 缩短开发时间 将架构师与测试开发人员的角色分离	降低硬件淘汰或硬件变更的风险 ■ 降低对特定仪器的依赖性 ■ 无需修改测试序列即可变更硬件 降低代码复杂性，方便未来对测试进行支持/变更提高代码在不同平台上的兼容性

了解HAL和MAL之间的区别非常重要。HAL属于代码接口，支持应用软件在通用层而非特定设备层与仪器进行交互。通常，HAL定义了仪器类别或这些仪器须符合的类型、标准参数和功能。换句话说，从仪器的角度来看，HAL提供了与仪器通信的通用接口。MAL属于软件接口，提供了可以在一组抽象硬件上执行的高级操作。从UUT的角度来看，这些操作是使用多种仪器执行同一项任务的一种方式。这些共同组成了一个硬件抽象框架。

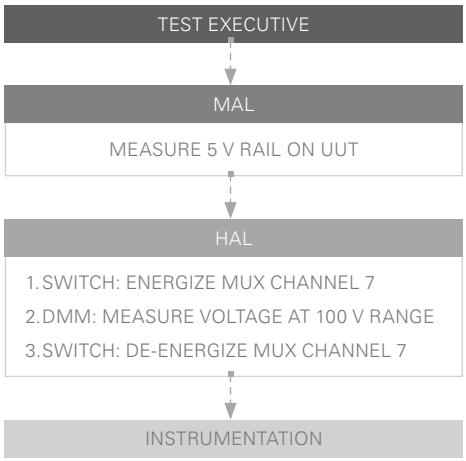


图1 抽象框架概览

打印机对话框就是一个典型的HAL/MAL日常用例。使用计算机打印时，不必打开终端，将原始串行、USB或TCP命令发送到打印机进行初始化和配置，然后再发送要打印的数据。硬件驱动程序实现用于执行配置和打印的方法。为了提升打印机的易用性，所有打印机制造商均遵循特定标准，将这些方法部署到驱动程序中。在同一硬件上执行多个任务时，所使用的通用接口就是HAL。那么，是否需要编写代码调用HAL的抽象方法来配置和打印文档呢？不需要，选择打印时，会显示打印对话框。此对话框提供了一个通用接口，用于调整配置参数，并将可打印数据发送到设备。这就是MAL，它让您能够直观地使用所有打印机，而无需了解打印机设备的底层功能。与打印文档一样，ATE HAL定义了所有仪器类型都必须遵循的通用底层任务集，而MAL则提供了执行高层操作从而使机器运行的一种通用方法。

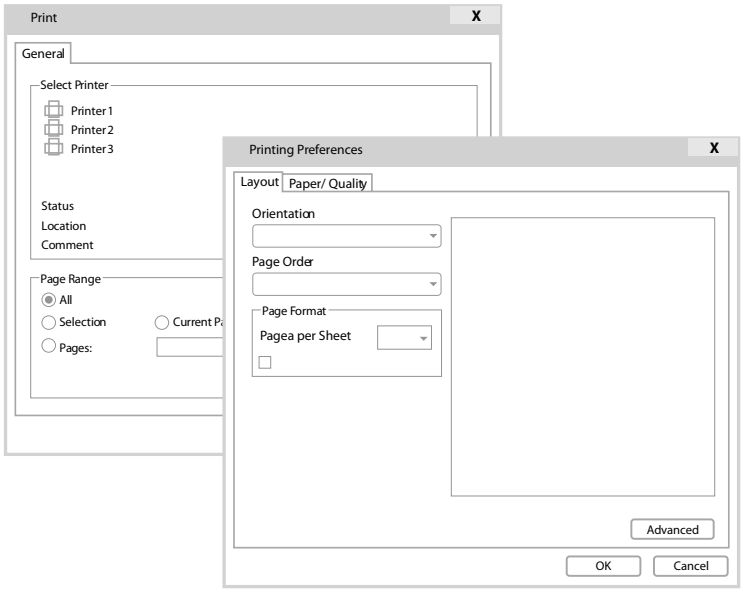


图2 打印机对话框就是一个典型的HAL/MAL日常用例

现有的HAL/MAL

测试和测量工程师采用了许多方法来解决HAL和MAL的问题。其中大部分方法可以直接使用，或者集成到更大型的自定义HAL/MAL方法中，从而以最小的投入实现功能扩展。以下是几个最常见的例子。

现成的软件抽象层

抽象	描述	类型	优点	缺点
供应商特定的驱动程序产品系列 驱动程序（NI-DAQmx、模块化仪器、Pickering PILPXI）	HAL	供应商特定的驱动程序产品系列为供应商的某些常见仪器组提供通用接口。这些驱动程序集可以连接每个特定系列的数十到数百种仪器。示例包括NI驱动程序（例如NI-DAQmx、NI-DCPower、NI-DMM、NI-Scope、NI-SWITCH和NI-FGEN）和Pickering PILPXI。	■ 为所支持的仪器提供通用的直观接口 ■ 有详细文档记录并经过测试 ■ 提供所有可用的功能 ■ 培训周期短-同一个驱动程序可控制该系列的所有仪器	■ 仅对每个供应商的特定驱动程序有效 ■ 并非所有仪器都支持所有功能
行业标准接口	HAL	IVI是仪器驱动程序软件的标准，可以增强仪器的互换性以及与符合IVI的仪器连接时的灵活性。该标准定义了13种仪器的规范，目前许多制造商都遵循这些规范，因此单个驱动程序可控制多种类型的仪器。仪器类型包括DMM、示波器、任意波形/函数发生器、DC电源、开关、功率计、频谱分析仪、RF信号发生器、计数器、数字化仪、下变频器、上变频器和交流电源。	■ 适用于从USB到PXI等各种仪器 ■ 兼容现有的许多GPIB，串行和LXI仪器 ■ 即插即用所有驱动程序都采用标准编程模式 ■ 上层仪器API支持仿真设备	■ 只指定API，而不实现-对于相同的测量，两个“可互换”的实现可能会返回不同的结果 ■ 无法用于不符合标准的仪器 ■ 可能无法实现所需的所有功能 ■ 可能会出现仪器无法支持的功能
Switch Executive	MAL	Switch Executive是一款开关管理与路径规划应用程序，用于将符合标准的开关矩阵和多路复用器仪器组合成一个虚拟开关设备。此虚拟开关可以使用命名的信号通道和路径直观地进行配置和驱动。	■ 直观的开关路径设置和操作 ■ 基于UUT或以测试为中心的名称定义通道和路径 ■ 定义无连接路径以提高安全性	■ 要求开关符合NI或IVI标准 ■ 不支持通过NI-DAQmx控制的继电器

现成的抽象能够帮助我们通过少量的自定义获得诸多功能。但是，这些抽象无法统一。IVI驱动程序和NI产品驱动程序对于符合标准的仪器而言是有效的HAL，但是从以仪器为中心的角度，它们仍然需要开发测试序列。从以测试为中心的角度来看，Switch Executive非常适合抽象开关路径，但它只能用于符合NI或IVI标准的开关连接(无模拟或数字I/O、DMM、示波器、电源等)。使用统一的HAL/MAL，可以更有效地开发以UUT为中心的序列，从而连接多种仪器，更好地处理仪器通道和连接的变更。

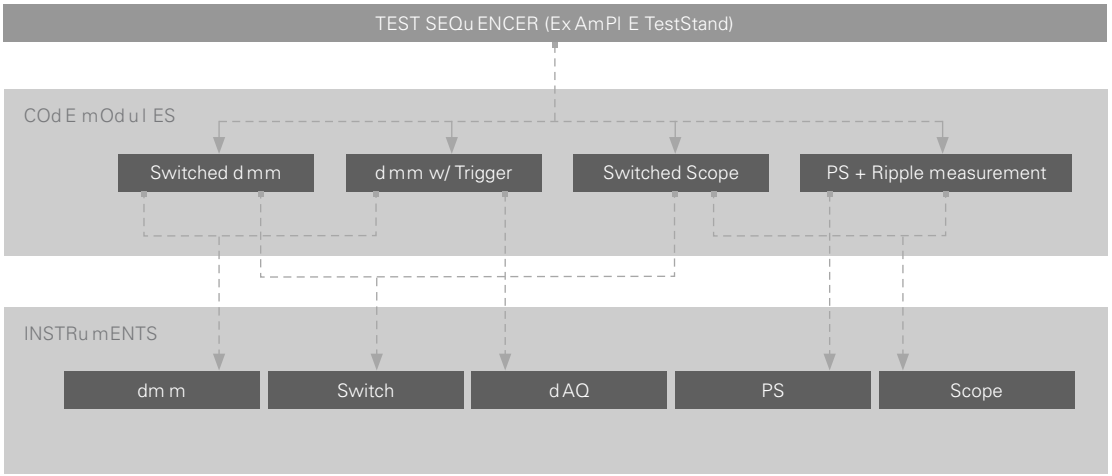


图3 不使用抽象的自动化测试软件的结构图

虽然HAL和MAL会带来诸多好处，但它们通常需要工程师基于过去的经验进行大量预测。他们需要考虑多个不同层次的抽象。有些抽象是软件和时间密集型，有些则是现成即用。一般来说，对特定仪器和测量进行抽象的程度越高，需要的框架规划和软件开发越高级。构建大型抽象框架非常耗时，并且如果没有适当规划，可能会有风险。如初始假设或实现不恰当，可能会持续产生积极和消极后果。务必要找到满足特定需求的硬件替代产品。如果不确定如何进行，请从简单处着手，保持可扩展性，并尽可能使用内置抽象。

背景

为了更好地理解HAL/MAL的实现方式，必须了解自动化测试软件的结构。从顶层看，自动化测试软件采用测试执行程序(或序列生成器)，例如TestStand。执行程序会调用一系列测试步骤，这些步骤通常属于代码模块或函数，使用LabVIEW软件中的图形化语言、C语言、.NET或ActiveX等并结合仪器特定的自定义方案进行开发；这些代码模块具有特定用途，例如使用DMM和开关的开关式DMM，或同时使用电源和示波器用于纹波测量的电源等。这样做会带来一定的好处，使每位开发人员都能够编写所需的特定功能，但同时需要大量的交叉功能，并且可能难以开发、部署和管理。而且，它要求每位测试开发人员都精通底层软件(如LabVIEW)。

不使用抽象

如果不使用硬件或测量抽象，就必须使用代码模块，直接引用驱动程序来与仪器连接。这导致测试序列与特定仪器和特定驱动程序代码密不可分。

不采用HAL/MAL框架时，通常会发生四个不可避免的问题：

- 由于仪器过时或需求变化，需要更换仪器-如果不采用抽象，则每次调用仪器时都需要更改驱动程序，在典型的测试序列中这可能涉及几十个步骤。每次更换仪器都会引起一些列软件更改。
- 由于出现新的需求，导致驱动程序的功能发生变化-如果驱动程序需要更新，则可能需要更新该驱动程序的每个例程以匹配新代码，尤其是输入或输出发生变更时。而且，直接调用驱动程序代码模块要求每位测试开发人员都要了解他们所使用的每个驱动程序的内部工作原理，特别是在使用多功能操作引擎的情况下。如要使用所有这些功能，测试工程师还必须精通软件。
- 测试序列是从仪器的角度进行开发的-通过使用仪器特定的驱动程序，所有测试序列都使用以仪器为中心的通道名称开发(例如使用以仪器为中心的名称开发测试序列)，而不是以UUT或测试为中心的名称(例如5v_Rail、LED_Control、VDD)进行开发。由于从UUT的角度开发测试程序，开发和调试变得非常困难。而且，对测试进行任何变更都需要对仪器、连线和互连系统有着非常深入的了解。
- 测试序列的开发通常与硬件开发同步进行-为了满足紧迫的时间期限，软件和硬件开发通常同步进行。因此，在开发测试序列时，并不总是能够掌握仪器和通道的详细信息。如果不采用抽象，就需要为驱动程序、通道编号和连接预留占位符。任何硬件信号发生变更，都需要更新测试序列。

例如，使用自定义方案，多路复用DMM测量代码模块可能类似于下图，即常见的开关式DMM LabVIEW VI。代码模块包含对特定仪器类型的特定调用集合。在本例中，这些集合是NI Mux和NI DMM。该代码模块基于输入通道和开关拓扑结构连接开关，基于一些输入参数使用DMM进行测量，然后断开开关。在测试执行程序中，必须了解要填写哪些字段，以及从仪器的角度来看需要哪些通道、拓扑和配置。此外，还必须确保将开关和DMM测量数据正确地传递到代码模块。

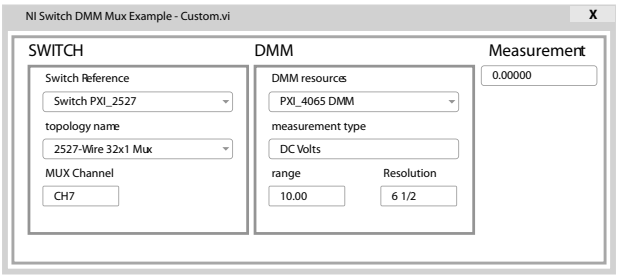


图 4 LabVIEW中典型多路复用DMM测量应用程序的前面板

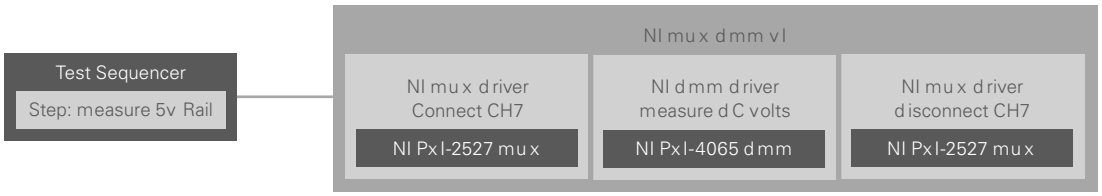


图 5 执行多路复用DMM测量的嵌套式命令调用

从测试执行程序的角度来看，可以调用代码模块来执行特定的函数(多路复用DMM)。该函数可对其适用的仪器进行特定调用。以下程序框图显示的是命令调用的嵌套。在图中，测试执行程序包含调用代码模块的步骤。代码模块使用驱动程序与特定仪器进行通信。每个外部项都依赖于其内部调用。

如果有仪器必须更换，依赖关系中的每个函数都必须更改。例如，如果初始多路复用器的通道不足，并且需要变换为通道数更高的矩阵，则由于存在依赖关系链，必须进行一系列更改：

- 仪器—PXI-2527多路复用器更改为PXI-2532B矩阵
- 驱动程序—NI Mux驱动程序更改为NI 矩阵(通道更改为行/列)
- 代码模块—NI Mux DMM VI必须更改为NI Matrix DMM VI
- 函数调用—必须更新测试执行程序对对代码模块的调用
- 序列—每次调用该代码模块时，都必须更新测试序列

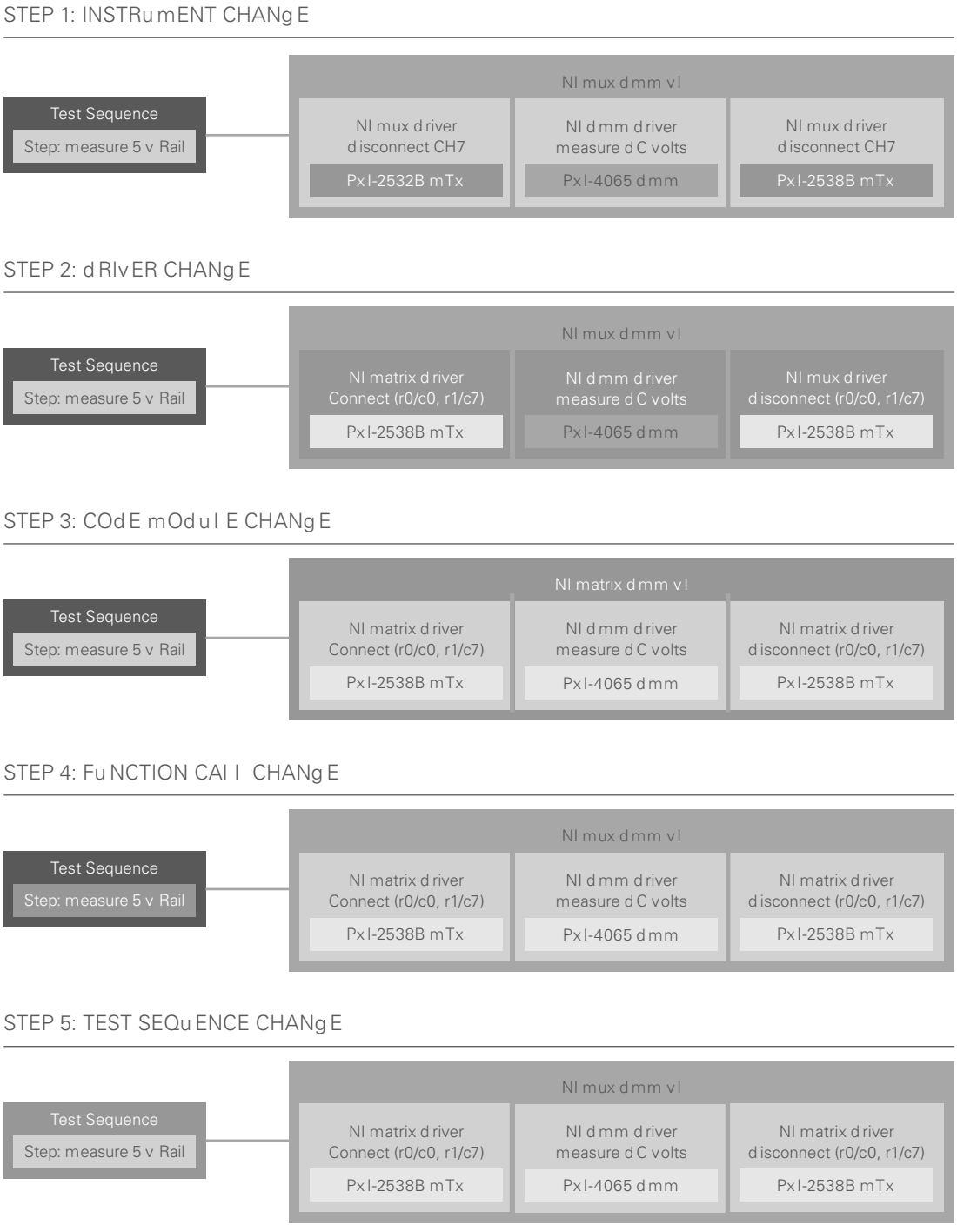


图 6 依赖关系链所需的非抽象更改

使用抽象

硬件和测量抽象打破了测试执行程序和与仪器交互的代码模块之间的耦合。测试执行程序不调用直接与特定仪器交互的代码模块，而是与MAL进行交互。这定义了基于通用仪器类型执行常见任务的操作或步骤类型。这些操作是仪器通用的，通常具有高级别名称，例如“信号输入”、“信号输出”、“连接”、“电源”和“负载”等。它们还采用测试特定的参数(而不是仪器特定的参数)，例如信号名称、连接名称、电源别名、电压/电流和负载方法(CV、CC、CP)等。映射框架使用配置文件将通用操作的测试特定参数转换为仪器特的定参数，如仪器引用、通道编号、矩阵行和列、GPIB地址和仪器配置约束等。此框架与HAL连接，以与配置文件定义的特定仪器进行通信。它基于MAL操作类型调用每个特定仪器对应的方法，并采用从配置文件中提取的仪器特定的参数。

如果将每个步骤视为一份烹饪食谱(煎饼)，则配置文件中的信息将是配料(鸡蛋、牛奶、黄油、面粉)，操作就是烹饪功能(混合、搅拌、拍打)，驱动程序就是厨房工具(碗、搅拌机、煎锅)，框架就是将所有这些整合在一起的指令。

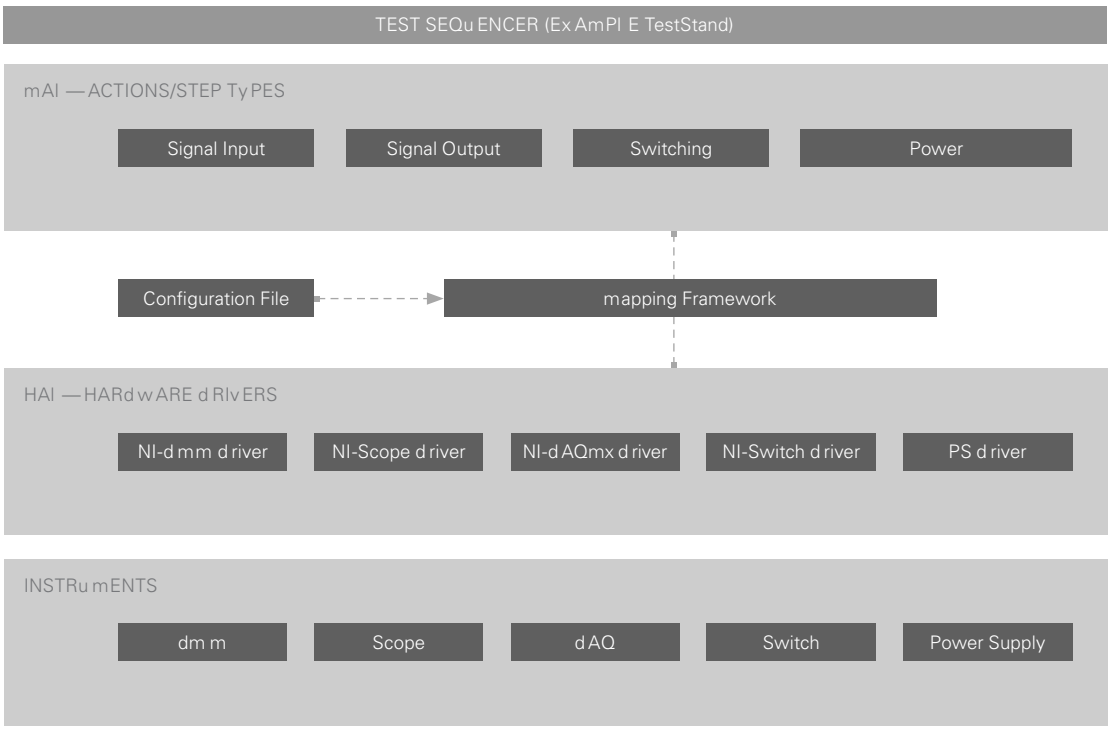


图 7 使用抽象的自动化测试软件的结构图

本节将继续介绍使用抽象的多路复用DMM示例。在本示例中，测试执行程序使用步骤特定的输入参数5v Rail来调用通用步骤类型“信号输入”。在这个特定的框架中，“信号输入”定义为三个设备操作：连接信号路径、读取测量设备、断开信号路径。这些操作使用5v Rail参数传递至映射框架。映射框架读取配置文件，查找5v Rail的仪器和通道详细信息。这些信息对应于PXI-2527多路复用器通道7的连接，以及PXI-4065 DMM在DC电压模式下执行的测量。然后，此框架调用适当的抽象驱动程序NI-Switch和NI-DMM，与配置文件定义的特定仪器进行通信。

执行不使用抽象的示例中的更改，用PXI-2532B矩阵替代PXI-2527复用器，事实证明，使用HAL/MAL框架执行更改更容易。由于所有仪器特定的详细信息都存储在配置文件中，并且HAL提供了与类似仪器进行交互的通用接口，因此只需更改配置文件即可。将PXI-2527复用器：通道7替换为PXI-2532B矩阵：r0/c0，r1，c7后，映射框架将自动提取更新后的详细信息并使用新参数调用新矩阵，无需更改测试序列或代码模块。

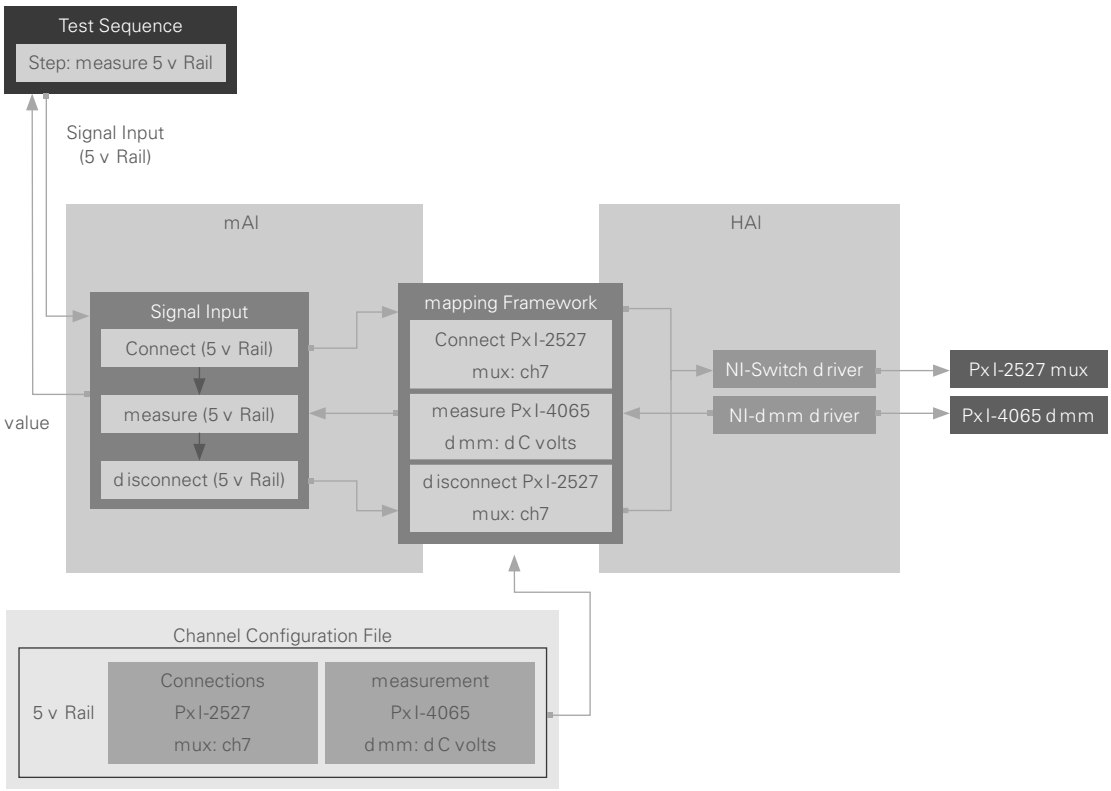


图8 在DMM测量时使用抽象框架调用函数

方法

在确定抽象框架时要考虑的首要问题是所有其他决策所基于的抽象范围。一种极端情况是不使用抽象，每个硬件接口都是对仪器驱动程序的直接调用。另一种极端情况是完全使用抽象，组件、通信协议、测量和配置格式之间每个可能的接口都定义一个抽象。本节将探讨涵盖各种可能性的一些方法。

方法1：仪器专用驱动程序

仪器专用驱动程序方法是自动化测试中最常用的方法，主要是因为该方法只需很少的编码、预测和规划工作。采用这种方法，可开发出连接特定仪器的底层代码模块。这些代码模块通常被称为底层驱动程序或仪器驱动程序，然后由更高层的代码模块或直接由测试执行程序调用。下面的框图显示了专为特定仪器开发的每个仪器驱动程序。在这种情况下，如果更换仪器，必须同时更改驱动程序和更高级别的调用。

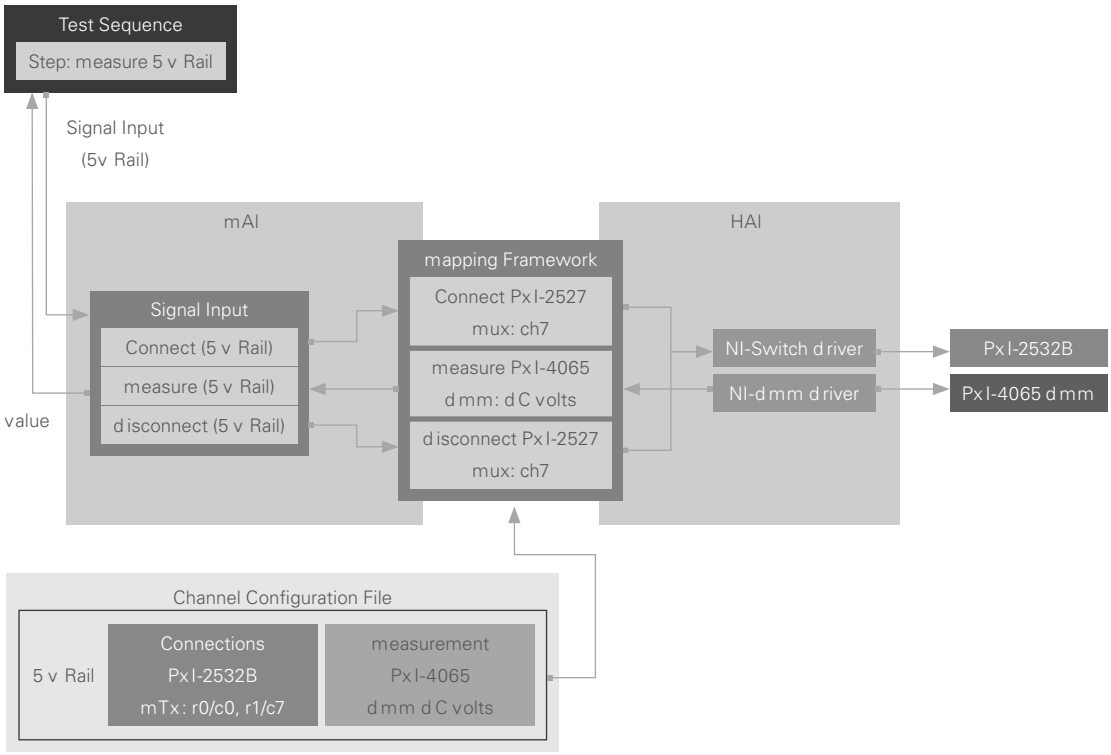


图 9 使用抽象时，能够轻松更新硬件且只需很少的软件更新操作—只需更新配置文件即可

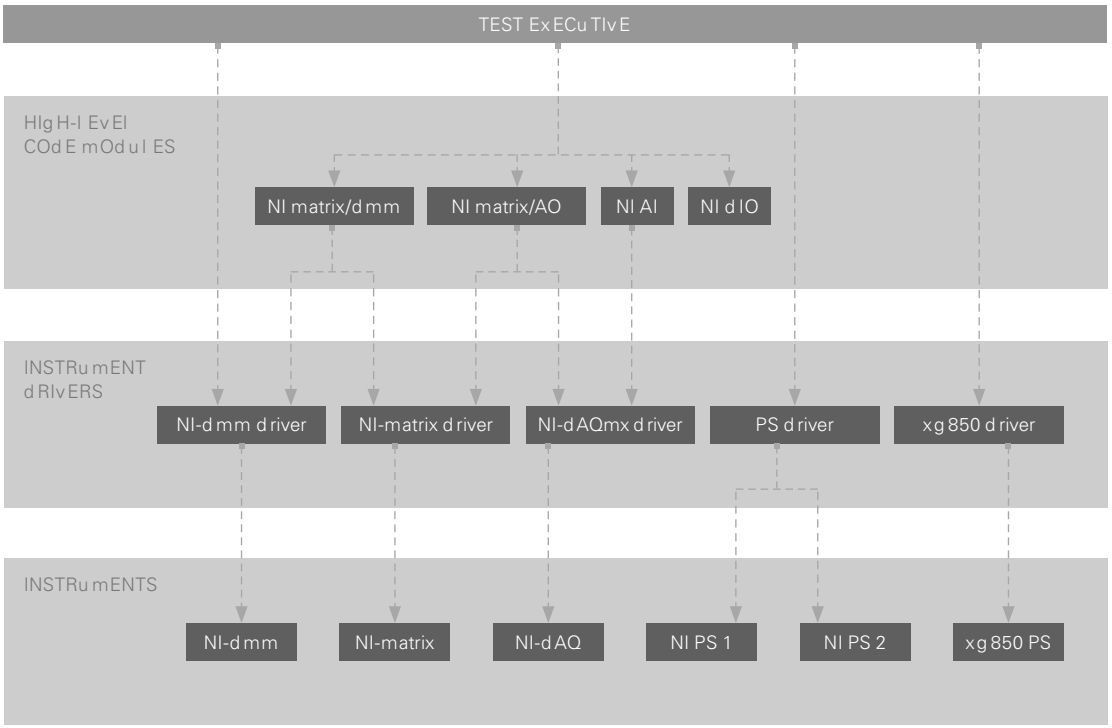


图 10 不使用抽象时自动化测试软件的仪器专用驱动程序方法概览

虽然此方法不包括任何抽象，但在提高驱动程序开发和交互的可靠性方面，仍然值得借鉴：

- 开发或使用仪器驱动程序包来连接每个仪器。
  - 底层驱动程序包实现了初始化、交互以及关闭与仪器的连接所需的所有功能。
  - 功能应简单、单一。
  - 驱动程序应该能够处理同一仪器类型的多个实例(例如同一系统中的两个相同电源)。
- 开发包装器仪器驱动程序，简化仪器接口。
  - 已有的驱动程序可能会包含数十个难以理解的函数，可以将已有的全功能仪器驱动程序封装到更简单的包装器仪器驱动程序中，提高易用性。
- 确保所有仪器连接都经过仪器驱动程序。
  - 这为所有仪器通信提供了单一进入点，从而简化了调试，减少了争用条件，并支持通过一个统一的位置管理仪器状态。
  - 如果已开发包装器仪器驱动程序，它应该成为单一进入点。
  - 驱动程序可以由测试执行程序直接调用，或者由更高级别的代码模块调用。
- 不要在驱动程序层实现测试特定的函数。
  - 测试特定的算法应该由更高级别的代码模块实现或在测试执行程序中实现。
- 确保仪器驱动程序之间不会交互。
  - 应由调用各个仪器驱动程序的更高层级代码模块或测试执行程序执行仪器之间的交互。

方法2：现成的HAL/MAL

若要将抽象整合到仪器驱动程序架构内，最快的方法是使用已有的HAL和MAL。虽然目前市面上提供的完全集成的HAL/MAL抽象框架有限，但许多硬件供应商已经在其仪器中实现了一定级别的硬件抽象；Switch Executive就是专门针对开关连接和路径规划而开发的MAL。基于这些已有的抽象来构建代码模块，就能够以最小的开发工作量提高ATE软件的适应性和抽象性。

现成的硬件抽象

已有的硬件抽象会采用通用的底层接口连接各种仪器。这能够减少所需仪器专用驱动程序的数量，同时降低系统中仪器更换所造成的影响。测试执行程序和更高级别的代码模块可以引用常规驱动程序，从而减少开发工作量并降低仪器更换造成的影响。当实现下面定义的其中一种抽象类型时，特定接口的I/O是固定的。因此，仪器更换通常不会导致代码模块更改。

可以通过两种方式利用已有的硬件抽象：仪器系列驱动程序和通信标准。仪器系列驱动程序往往是供应商特定的驱动程序，可以控制该供应商目录中特定仪器类型的许多仪器。通信标准提供了一种行业公认的方法，用于连接多个不同供应商提供的某些类型的仪器。可以使用这些标准来开发仪器驱动程序，从而控制各种类似的仪器。

通过仪器系列驱动程序实现硬件抽象

仪器系列驱动程序是供应商特定的驱动程序，可与常见的仪器产品系列进行通信。与IVI驱动程序类似，仪器系列驱动程序使用通用驱动程序实现与多个不同仪器的通信。常见的示例包括NI模块化仪器(NI-DMM、NI-Switch、NI-d CPower和NI-Scope)和Pickering PII PXI。仪器系列驱动程序可增强其所适用的产品系列之间的互换性。虽然它们不支持跨供应商或跨产品系列复用，但这些驱动程序通常直观、易于实现，并且包含每个仪器的大部分(就算不是全部)功能。

基于通信标准的硬件抽象

许多仪器制造商都遵循设备通信的行业标准。如遵循行业标准，制造商的仪器就可以与其他类似仪器进行互操作。两个最常见的标准包括可编程仪器的标准命令(SCPI，通常发音为“skippy”)和可互换虚拟仪器(IVI)。

SCPI

SCPI定义了测试和测量行业中用于控制可编程仪器的语法和命令标准。通过这些命令，用户可以设置和查询仪器的通用参数。SCPI命令可以通过多种通信协议实现，包括GPIB、LAN和串行协议等。开发符合SCPI标准的单一驱动程序后，就可以与同一类型的多台仪器(DC电源、电子负载等)进行通信，而无需开发仪器特定的驱动程序。在开发SCPI驱动程序时，请注意，虽然SCPI定义了一个通用的命令和语法标准，但不同的供应商实现此标准时有时会有微小的差异，因此开发100%标准的驱动程序有些困难。在选择符合SCPI的仪器和开发驱动程序时，务必要密切关注每个仪器的命令详细信息。

IVI

IVI是仪器驱动程序软件的标准，可增强仪器的互换性以及与符合IVI的仪器连接时的灵活性。该标准使用VISA定义了I/O抽象层。由于SCPI已经整合到IVI中，许多符合SCPI的仪器在定义上也符合IVI标准。IVI标准定义了13个仪器类型的规范，许多制造商都在遵循这一标准，因此每种类型的单一驱动程序就能够控制不同供应商的多种特定仪器。仪器类型包括DMM、示波器、任意波形/函数发生器、DC电源、开关、功率计、频谱分析仪、RF信号发生器、计数器、数字化仪、下变频器、上变频器和交流电源。许多PXI和现有仪器都遵循IVI标准，并且已有驱动程序支持多种编程语言，并可用于测试执行程序。

通过使用IVI驱动程序为符合IVI标准的仪器开发测试序列和代码模块，不同供应商的仪器看起来就会相同。您可以针对每种类型使用同一个驱动程序集，以连接许多可互换的仪器。相比使用仪器特定的驱动程序，使用具有类似功能的仪器替换符合IVI标准的仪器时，代码和序列更新工作将会大大减少。然而，虽然IVI驱动程序可以实现符合标准的仪器的大多数功能，但是有些仪器可能仍需要特定代码来执行自定义功能，而有些仪器则可能无法处理所有符合IVI标准的功能。最后，虽然两台仪器可以执行相同的IVI功能，但获得的结果可能不一定相同。因此，只要进行更换，就需要验证和测试仪器的功能。

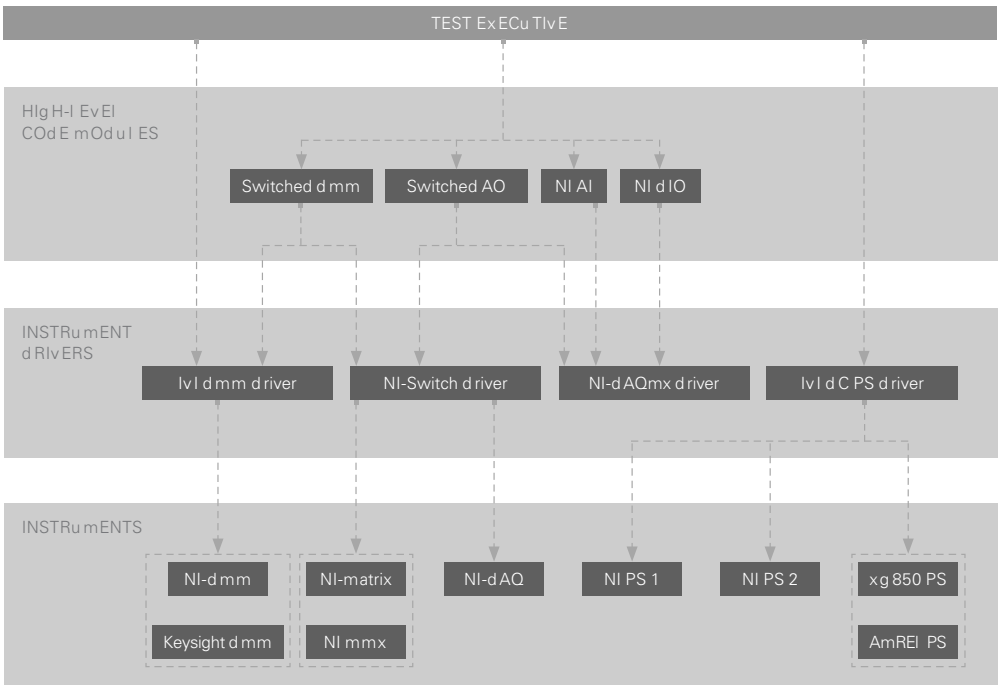


图 11 采用现成抽象的自动化测试软件概览

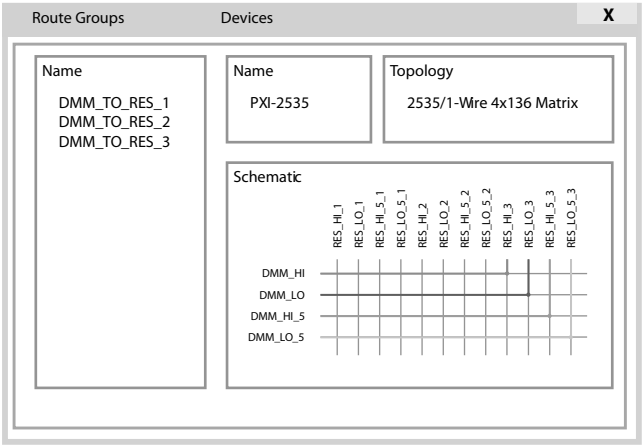


图 12 Switch Executive MAL配置界面

现成的测量抽象

虽然仪器普遍已经具有硬件抽象，但它仅从仪器的角度来实现抽象，而测量抽象是非常有限的。由于各测试系统的定制化程度都非常高，很难为测量操作定义一个标准。Switch Executive是最常用的现成测量抽象层，这是一款开关管理和路径规划应用程序，用于将符合标准的开关矩阵和多路复用器仪器组合成一个虚拟开关设备。此虚拟开关可以使用用户命名的通道和路径直观地进行配置和驱动。虽然Switch Executive仅适用于符合NI-Switch和IVI标准的设备，但却为从UUT或测试角度定义开关路径提供了一种极佳的方法。

首先，Switch Executive是一种图形化配置实用程序，用于在单台仪器内和跨多台仪器设置开关通道名称和路径。行、列、通道和路径组都可以进行配置和命名，以便直观地设置开关机制。其次，Switch Executive可集成到LabVIEW和TestStand中，为按名称设置和查询预配置的路径提供了强大的接口。当与TestStand测试执行软件一起使用时，Switch Executive可以逐步执行，以便在执行步骤的代码模块之前为开关仪器提供用户命名的接口。

Switch Executive是一个有用的MAL，可将开关连接抽象为测试特定的名称而不是仪器特定的名称。当与IVI开关硬件抽象结合使用时，会形成一个非常出色的集成HAL/MAL框架。但是，当使用非IVI开关或外部数字输出控制的继电器时，Switch Executive不适用。此外，Switch Executive仅适用于开关路径规划，无法扩展到其他测量类型。如果需要开关之外的其他集成HAL/MAL框架，就需要开发自定义代码。

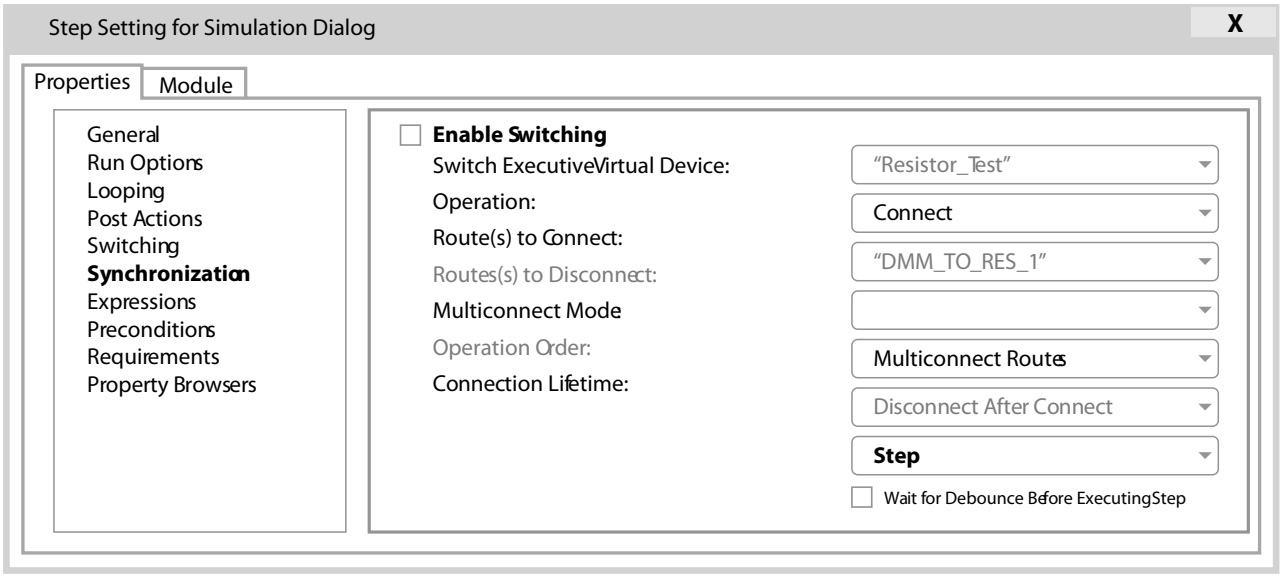


图 13 Switch Executive MAL测试设置

方法3：集成式HAL/MAL框架

集成式HAL/MAL框架提供了一种架构，用于实现由测试执行程序(MAL)调用的高级别动作，连接低层级驱动程序以与仪器(HAL)进行通信，以及在两者之间映射详细信息。此框架由三种主要类型的代码模块实现：动作、映射框架和硬件驱动程序。每种代码模块都由一组API定义。API是一组用于软件应用程序的工具(函数、协议、参数、语法)，定义了代码模块的运行方式及其与周围软件的交互方式。在基本的HAL/MAL框架中，有四种常见的API：测量API、配置API、硬件驱动API和仪器API。代码模块、API及其交互如下图所示。

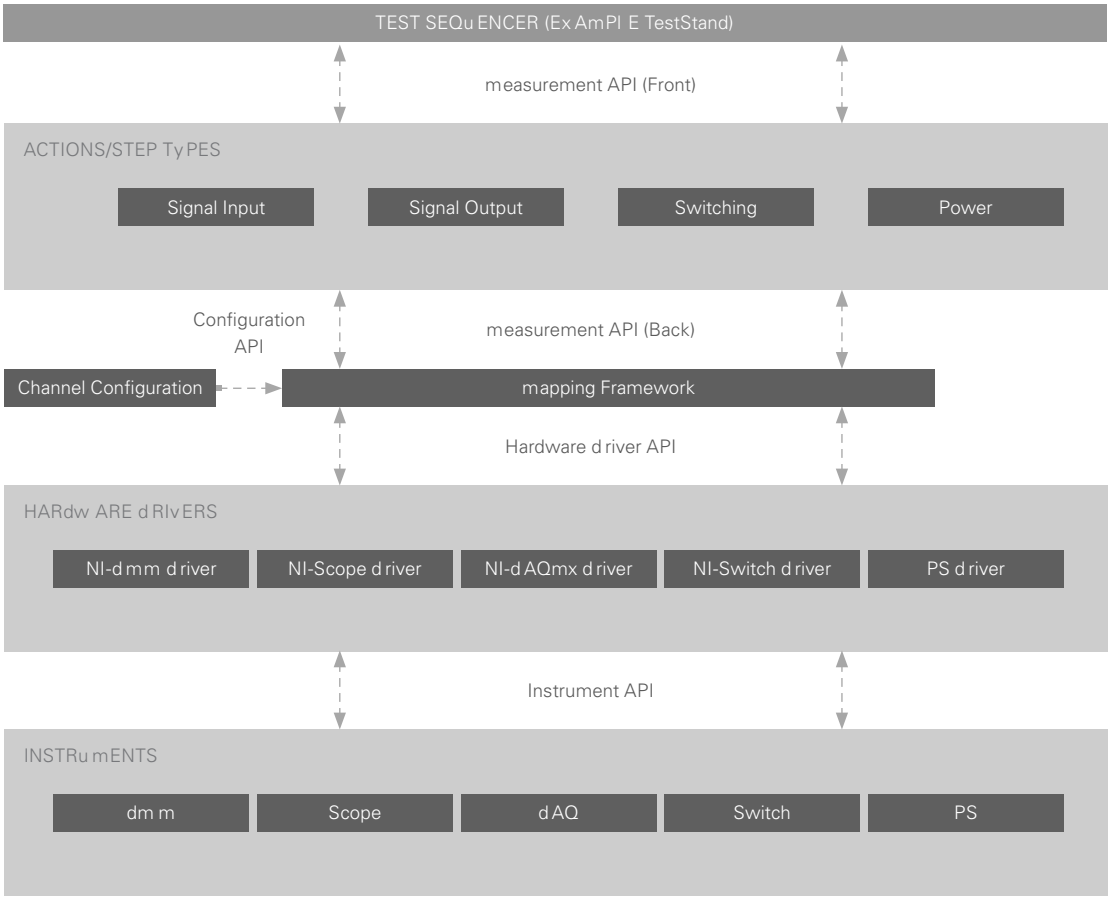


图 14 采用集成式MAL和HAL的自动化测试软件概览

三种类型的代码模块包括：

- **动作/步骤类型**—动作定义了MAL的功能。特定动作定义了每个测量类型(输入或输出)。动作可以像调用一种仪器类型的一个函数那样简单，例如进行开关连接；也可以像调用多个仪器的多个函数那样复杂，例如将开关连接与设置电源电压、电流和启用状态相结合。这些代码模块实现了用于定义其方法和参数的测量API。
- **映射框架**—映射框架是指使用配置文件中的默认值将高层级动作链接到底层仪器设备的内部代码。映射框架代码模块通过硬件驱动程序API与硬件驱动程序交互，并通过测量API与动作交互。
- **硬件驱动程序**—硬件驱动程序代码模块将通用设备类型函数调用(DMM、电源、开关等)转换为仪器特定通信(SCPI、IVI、NI-d CPower和专有通信)。因此，硬件驱动程序在一端实现硬件驱动程序API，在另一端实现仪器特定的API。

HAL/MAL抽象框架至少包含以下四个API中的一个：

- **测量API**—测量API定义了高层级动作及其特定参数。这是MAL的定义。测量API定义了一个所有动作都必须遵循的通用框架，然后支持每个动作定义执行其特定函数所需的API(参数和方法)。每个动作都必须至少实现后端测量API，映射框架使用此API将人类可读别名链接到特定的开关和测量仪器以及相应的通道。或者，可以开发API的前端，为每个动作提供更直观的接口。这个前端通常是一个配置对话框/向导。信号输入的测量API示例可定义信号输入别名和返回值输出。API还将为别名定义连接、测量，然后断开连接。
- **配置API**—映射框架使用配置API来详细说明如何将测量API转换为硬件API。配置API用于定义配置文件或数据库的参数、语法和内容。只有映射框架使用此API。例如，配置API可以规定配置文件是Microsoft Excel文件，并且每个信号别名应具有以下属性：名称、类型、连接详细信息、仪器、仪器配置和扩展。
- **硬件API**—硬件API是一种抽象的API，用于定义特定仪器类型必须实现的通用参数和方法。此API定义了HAL。例如，DMM Hardware API可能规定所有DMM必须能够初始化、配置(电压、电流、电阻、范围、分辨率)、测量(返回值)和关闭。
- **仪器API**—仪器API由每个单独的仪器定义，因此不是抽象层。每个仪器特定的硬件驱动程序都用于实现控制其特定仪器所需的函数和命令。这与在仪器特定的代码接口中使用的API相同，并且可实现用于该特定仪器的特定通信协议和命令。

为了更好地理解代码模块和API之间的交互，请查看多路复用DMM示例，其中详细说明了每个代码模块的输入和输出。

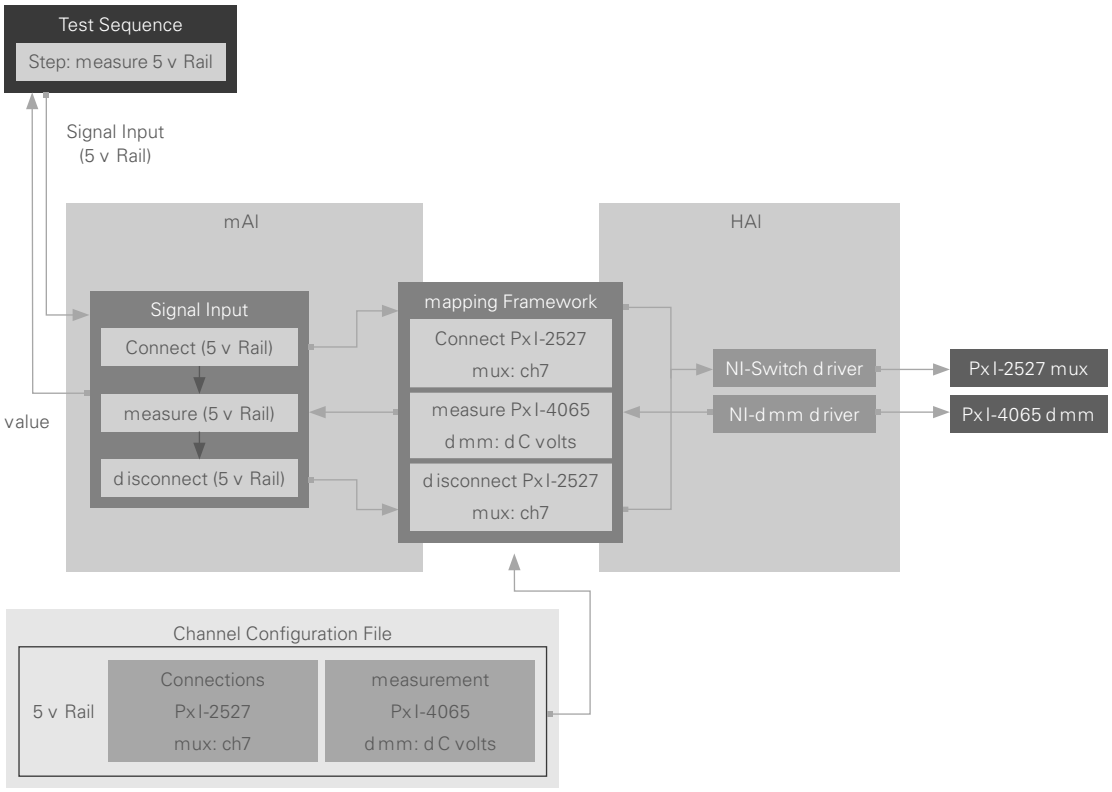


图 15 使用抽象框架的多路复用DMM测量

在本例中，信号输入块是动作代码模块，定义了信号输入应执行Switch Device Connect函数、Measurement Device Measure函数，然后执行Switch Device Disconnect函数。此函数的测量API明确了代码模块需要一个别名，此别名从测试执行程序接收，并传递到映射框架，然后从映射框架获取返回值传递回测试执行程序。

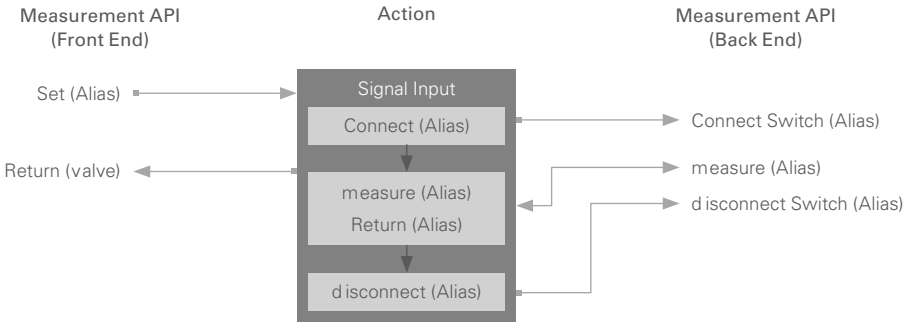


图 16 信号输入的MAL动作API示例

映射框架通过测量API接收来自动作的命令。然后，通过配置API解析配置文件中的别名数据，以获得正确的仪器ID和参数。配置API定义了系统配置的文件格式、语法和字段。然后，映射框架通过硬件驱动程序API将仪器特定信息传递到适当的驱动程序。

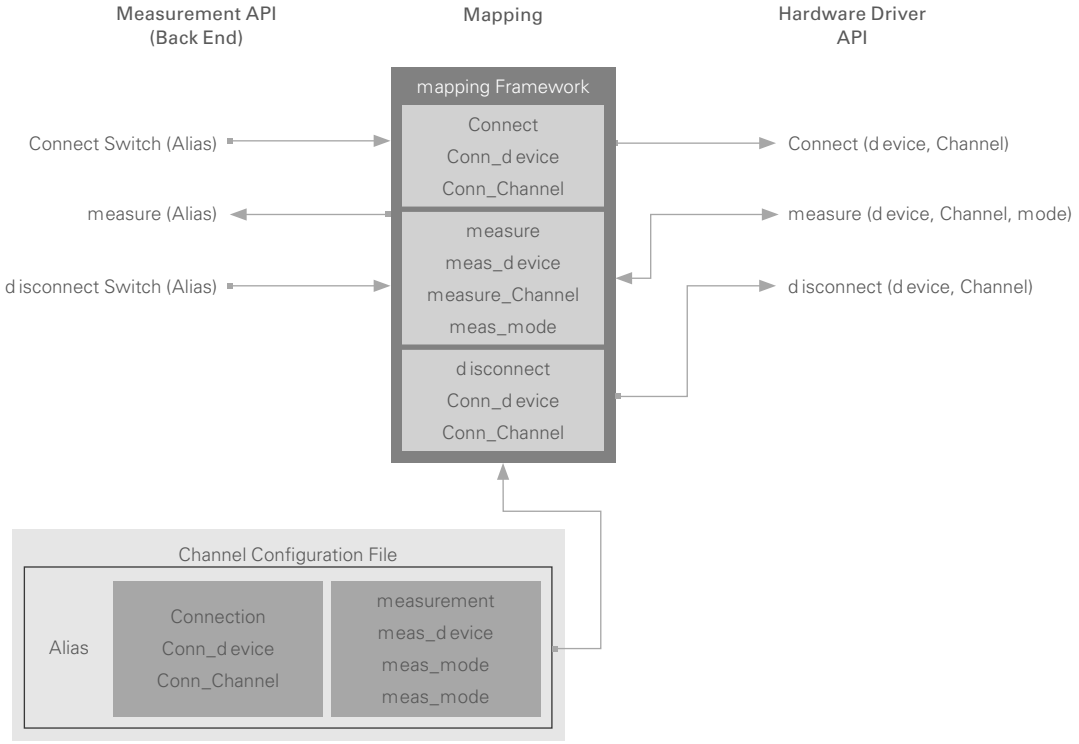


图 17 信号输入的映射框架API示例

映射框架使用通用硬件驱动程序API调用各个硬件驱动程序。然后，每个驱动程序都会说明通用设置的详细信息，并使用现成的方法和参数与特定仪器进行通信。

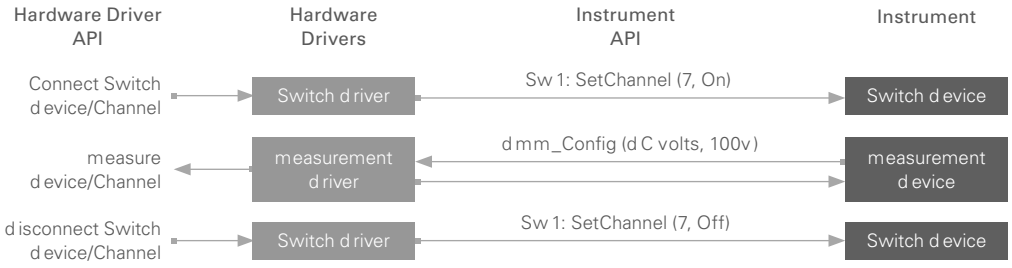


图 18 信号输入的硬件驱动程序API示例

方法4：HAL/MAL插件架构

插件是对集成框架的补充，可能非常有价值。真正的插件只是一个软件组件，可以在部署后进行修改，而无需重新部署整个应用程序。插件与主应用程序和/或框架分开存储在磁盘上，并在运行时动态加载。

虽然开发插件架构会带来一些挑战，但也能明确限制添加或修改功能的范围和风险，从而简化软件回归测试。开发时不使用插件的框架，在每次需要新的测量类型、仪器驱动程序或配置格式时，都必须重新构建。在不使用插件的情况下，整个源代码都内置到单个EXE中，因此即使对仪器库进行微小变更，也不能保证其不会无意中影响应用程序的其他特性。由于难以完全掌握源代码修改可能产生的所有影响，必须进行彻底测试。

插件架构便于开发人员添加或修复插件代码，而无需修改或重新部署底层框架，因而可实现最高的软件模块化水平。这通过编写一个仅依赖于抽象类或模块的框架来实现，该框架可动态加载所需的具体插件，并且通常仅在需要时才加载。插件架构成功与否取决于接口设计是否完善。换句话说，为了在测试框架中使用插件，框架必须知晓如何调用任何可能的插件组件。如果所有插件都采用一致的软件接口，那么在运行时加载这些插件只需要框架或测试应用程序知晓在哪里找到这些插件即可。

虽然这些是抽象框架的一些更常见的过程、API和代码模块，但它们肯定不是唯一的。每个框架都是独一无二的，并且有自己的要求、过程和实现方法。对于某些团队来说，这种抽象层级可能超出了需求。然而，在其他情况下，系统架构师可能需要注入额外的抽象层。根据框架架构师和用户的需求和能力，也可以采用不同的方法来实现这些API。有些工程师使用简单的动作引擎实现所有抽象，有些使用更高级的面向对象的编程，有些使用插件，还有一些更倾向于使用单个代码库。关键在于找到适合特定需求和能力的抽象和实现范围。同时还要明白，并非所有问题都可以通过抽象来解决，有时仍然需要采用仪器特定的代码。因此，在开发抽象层时，务必允许开发自定义代码来实现高级功能。这可通过允许更高级别的代码模块或测试执行程序获得仪器引用来实现。高级开发人员永远不应被框架所束缚。

抽象层方法的特性比较

无 ○ 部分 ◒ 全部- ●

抽象方法	方法1 不使用抽象	方法2 现成抽象	方法3 基本自定义	方法4 采用插件
支持使用以下组件替换各个仪器：				
采用相同通信协议的仪器	●	●	●	●
符合IVI标准或属于同一产品系列的仪器	○	●	●	●
采用不同通信协议的仪器	○	◒	●	●
无需修改测试序列即可更改仪器通道/接线(修改配置文件)	○	◒	●	●
从测试/UUT的角度执行测量/任务	○	◒	●	●
无需修改框架即可添加新仪器或新的测量	○	○	○	●

实际场景1

您是一家商业产品公司的测试工程师，负责开发新产品电子组件的功能测试。您需要测试三个PCBA和一个最终成品板。现在已有一个通用ATE仪器平台，但它已经过时，并且先前的测试项目因设备故障和仪器过时进展并不顺利。幸运的是，在项目进行过程中，已经开发了一个新的ATE平台，该平台支持可互换的测试头根据不同的成品板对仪器进行调整。

您的任务是开发测试序列和代码模块软件，以便与硬件工程师开发的仪器和连接件进行交互。您具备使用测试执行程序(与之前的平台使用的测试执行程序相同)的相关经验，并且有多年的软件应用程序开发经验。为了解决问题，已经讨论过使用抽象来帮助缓解之前系统中存在的过时问题。您必须确定这种方法是否可行以及如何实现。

是否使用抽象……

首先，您必须决定是否开发抽象框架，而不考虑抽象级别如何。考虑到已有IVI等现成抽象，答案几乎总是肯定的。只有在唯一一种情况下不需要开发抽象，即项目生命周期100%已知，并且永远不需要进行变更，但这几乎是不可能的。

您是否需要HAL?

接下来，您必须决定使用什么级别的硬件抽象。此时决策变得更为复杂，因为这涉及诸多因素。硬件抽象通常更容易理解，因此实现成本低于MAL。如果可以合理地使用预抽象驱动程序(如IVI和产品系列驱动程序)，则成本将更低。但是，一旦必须使用不属于某类驱动程序的仪器，则可能需要为每种仪器类型开发一个通用接口。例如，如果系统中有一些符合IVI标准的电源以及一个不符合该标准的电源，则可能需要开发适用于其中一种类型的抽象电源定义。确定抽象硬件定义通常需要了解该特定类型的大多数仪器的工作原理，然后可以使用这些信息来定义系统内每种仪器类型的通用方法和参数。

开发的目标是覆盖每台设备在合理预期下所使用的80%的功能。与团队讨论，确定每个仪器类型必须通过每个抽象仪器驱动程序实现的核心功能和参数。例如，团队可能会确定所有电源的核心功能应该是初始化、设置电压/电流/启用状态、回读电压/电流/启用状态以及关闭。虽然某些其他功能可能会在以后被电源使用，但这些功能并不一定必须作为系统标准架构的一部分。如果您对某种特定仪器类型不够了解，或者不确定需要哪些功能，请先构建小型架构。未来，您可以随时将新功能添加到标准架构中，但是在该框架被多个驱动程序使用后，很难更改该功能的参数或详细信息。

以下流程图可以帮助您确定适合选用哪种级别的硬件抽象。如果您不确定答案，可以假设一个更抽象的解决方案，或者假设一个不太抽象的解决方案。更抽象的解决方案需要进行更多的前期设计，但从长远来看可能会节省时间，而不太抽象的解决方案可以更快地启动和运行，但未来可能会出现问题。需要注意的一点是，要先问一下自己：我是否需要MAL。这是因为如果没有精心设计的HAL，就不能有效地实现MAL。

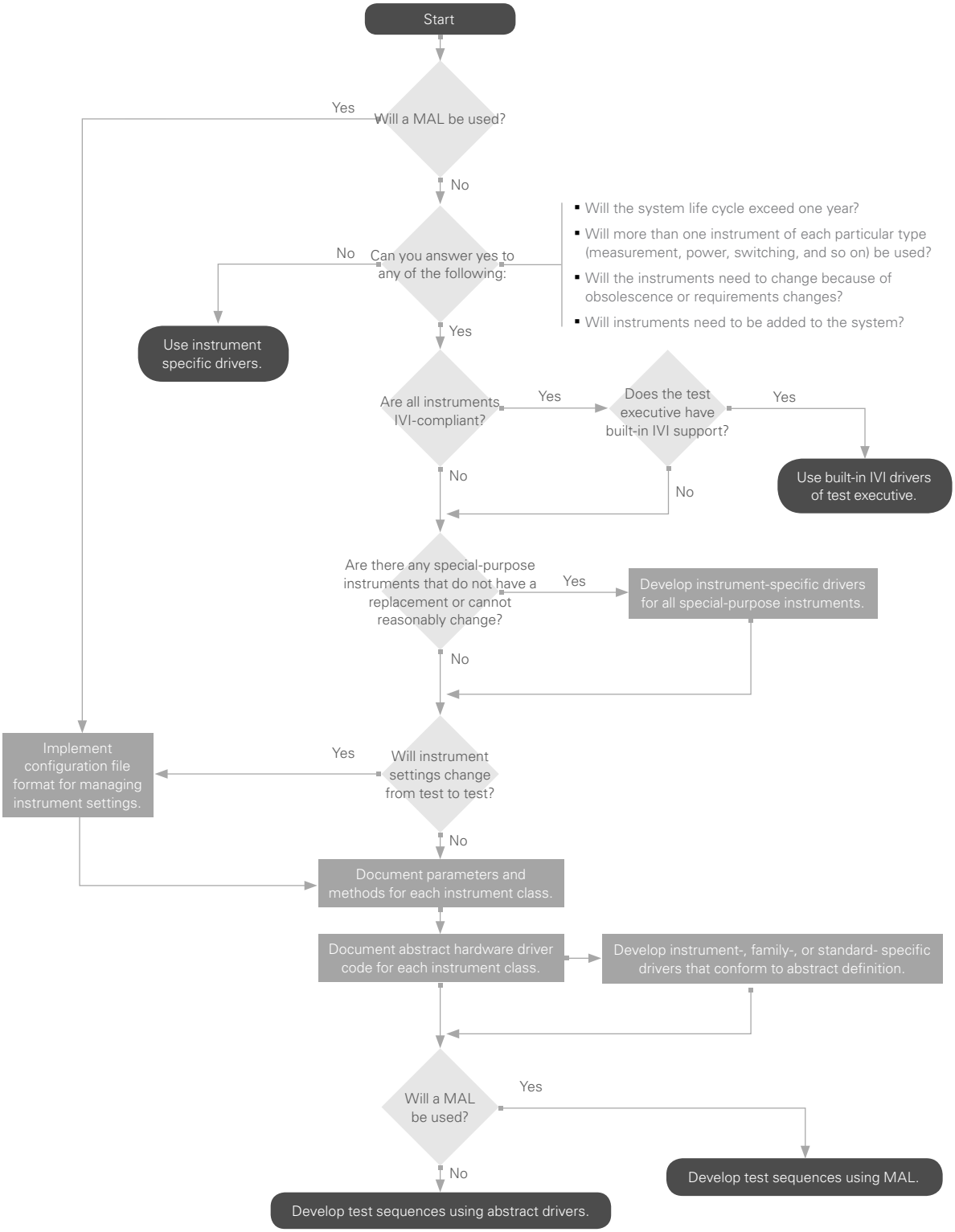


图 19 确定所要实现的抽象级别的决策流程图

您是否需要使用MAL?

要确定是否使用HAL，首先要确定是否需要使用MAL。这是因为如果不依赖硬件抽象，MAL几乎无法实现。因此，这个问题实质上是在问您是否需要采用集成式抽象框架。如果团队中的多名测试开发人员都缺少底层软件开发经验，则适合采用HAL/MAL。

以下几个主要问题可以帮助您决定是否开发MAL：

- 是否会有软件架构师可以规划和支持该框架？如果没有架构师/负责人，HAL/MAL难以获得有效的支持。
- 是否有多名测试开发人员缺乏软件开发经验？抽象框架的一个突出的优势在于，它能够缩短测试开发的培训周期。
- 系统生命周期是否足够长，可以支持多种产品？这可能需要一大笔前期投资，但使用频率越高，回报就越大。
- 您是否愿意开发和支持MAL？不使用抽象的效果要优于使用定义不明确和未有效实现的抽象。从长远来看，简单易用的HAL/MAL可以节省大量时间；但是，如果过于复杂或设计不当，可能会很麻烦，反而会增加开发和调试时间。

如果您对这些问题中的大多数回答都是肯定的，那么开发一个集成式抽象框架从长远来看很可能会获得回报。

实际场景2

即使抽象的所有好处已知，仍然存在如何衡量成本与回报的关键问题(单位通常是时间)。虽然抽象决策的第一部分通常从技术角度考虑，但是成本/效益决策必须在更高的业务层面做出。

成本是多少?

这个问题很难回答，因为它在很大程度上取决于开发人员的经验、编码能力和所需的抽象级别。但是，您可以估计各种组件的大致数量级，如下表所示。

抽象框架任务和成本

类别	任务	描述	估计小时数(低)	估计小时数(高)
规划	架构定义	记录动作类型、设备类型以及动作和设备之间的通用接口类型	24	48
	每种设备类型的HAL定义	记录每种设备的输入和输出以及方法	8 (每台设备)	16 (每台设备)
	每个动作的MAL定义	记录每种测量/动作的输入和输出以及方法	8 (每个动作)	16 (每个动作)
	配置定义	定义配置文件或数据库的格式、语法和内容	24	48
实现	映射框架开发	实现所有的软件，将配置文件映射到动作和抽象驱动程序-大部分底层框架都在此时开发	60	120
	抽象设备驱动程序开发	按设备类型进行抽象设备接口代码的软件开发-实质上是在构建仪器	4 (每台设备)	24 (每台设备)
	仪器驱动程序开发	为使用HAL的每个仪器特定驱动程序进行软件开发-填写到每个特定驱动程序的模板中	4 (每台仪器)	24 (每台仪器)
	动作开发	为MAL定义的每个动作进行软件开发-实现用于连接测试执行程序和映射框架的前端和后端API	4 (每个动作)	24 (每个动作)
总计		开发框架所需的总时间(不包括单个仪器驱动程序)-假设有五种设备类型和五个动作，并且每种设备有一个仪器特定的驱动程序	248	776

这表明完全集成式HAL/MAL抽象层所需的开发时间可以低至250小时，也可能超过750小时。根据抽象的级别，甚至可能超过1000小时。

如何降低成本?

涉及软件开发时，成本与复杂性密切相关。复杂性包括有利的复杂性和不利的复杂性，具体取决于其性质。目标是增加有利的复杂性，同时避免不利的复杂性。可以增加功能的复杂性是有利的。每个特性通常都会增加功能。采用灵活、可扩展的模块化代码来实现这些目标往往会更加复杂。但是当以简洁的方式实现时，这种复杂性会带来益处。由于规划不当、功能冗余和繁琐的意大利面条式代码而产生的复杂性是不利的，因为它增加了开发成本，但却没有增加功能。

您可以通过以下四种方式降低ATE抽象框架的复杂性：

- 预先规划架构。与大多数开发过程一样，前期规划和编写文档可以在开发过程中节省大量的时间和避免诸多麻烦。在前期进行规划并为API和代码模块编写文档，可以减少交叉功能和不必要的相互依赖，从而增强代码的可靠性，并减少不必要的复杂性。您不必针对每个API和代码模块的每个细微差别进行规划，但需要定义软件的主要交互、参数和基本功能。
- 规划不要太过超前。在开发大型架构时，工程师往往会过度设计，并尝试规划所有可能的场景。虽然前瞻性思维方式可能会带来益处，但最好是针对已知的情况进行设计。工程师在设计系统时经常会考虑最坏但通常不会发生的情况。这些工作只占总工作量的20%，但需要80%的时间去完成。结果将花费更多的时间来尝试处理假定的边缘情况，而不是专注于软件中大多数时间都会使用的功能。
- 认清一个事实，抽象并不能解决一切问题。抽象非常有用，但试图抽象出每一个可能的接口并不值得。相反，应在框架中包含自定义硬件交互，以解决通用接口无法实现的情况。为系统制定切实可行的规则，以减少抽象层。例如，将配置文件限制为单一格式(.INI、.XLS、database)，以降低映射框架的复杂性，或将动作限制为三个独立的硬件调用，以防止需要实现递归硬件驱动程序API调用的情况。
- 保持灵活、可扩展和模块化。虽然灵活性、可扩展性和模块化确实会增加复杂性，但在开发大型架构时大有裨益。此时非常适合采用插件架构，因为插件架构可定义低层级框架，而细节通过唯一的代码库实现。这意味着新功能可以在旧功能的基础上进行扩展，而不会对原有功能造成破坏。精心规划的插件架构不仅可满足已知情况的需求，还可在未来根据新的挑战进行扩展。

是否值得?

虽然抽象框架的开发可能非常耗时，即使顺利实现也是如此，但由于其回报往往大于付出而得到广泛应用。有多个关键因素可以提高回报率，让框架更为有效。其中许多回报可以通过节省的时间或精力来量化。下表列出了与任务相关的一些典型成本，并比较了非抽象系统与使用HAL/MAL抽象框架的系统之间的差异。

非抽象和抽象系统中与任务相关的成本

任务	估计用时(标准)	估计用时(抽象)	如何产生回报?
为新测试工程师提供测试软件平台培训	60小时 (每名工程师)	40小时 (每名工程师)	掌握如何使用抽象框架通常需要了解测试执行程序以及框架。在任何一种情况下,开发人员都必须了解如何与测试系统硬件进行交互。当使用仪器特定的驱动程序时,工程师必须了解每个驱动程序的详细信息和使用方法。然而,在学习抽象框架时,工程师只需要理解要执行的高层级动作,仪器详细信息都留在框架中。通常,这些高层级动作比各种仪器特定的驱动程序更直观、更容易实现。
基本功能测试序列的开发和调试(由经验丰富的工程师完成)	80小时 (每个序列)	40小时 (每个序列)	会加快测试序列的开发速度,因为硬件的详细信息都存储在一个固定位置,而不是存储在序列中的每个驱动程序调用中。从UUT的角度来看,测试序列会与硬件进行交互,从而使测试序列更直观、更好地匹配测试步骤。一般来说,如果框架直观易懂,可以将开发和调试时间缩短一半。
由新工程师执行的测试序列开发和调试	120小时 (每个序列)	60小时 (每个序列)	如果由新工程师或经验不足的工程师开发测试序列,会进一步缩短开发时间。由于框架规定了一组规则和功能,与使用仪器特定的驱动程序和代码相比,经验不足的工程师可以更好地使用已有步骤来开发序列。此外,如果框架直观易用,有产品意识的测试工程师无需掌握底层软件语言即可开发序列。
因仪器故障/过时或出现新的仪器需求而更新测试序列	驱动程序开发需要8小时,再加上4到20小时(每个序列)	驱动程序开发需要8小时,再加上<1小时 (每个序列)	当系统中的仪器需要更换时,测试序列也必须随之更改。对于非抽象平台,这意味着必须针对新仪器更新驱动程序调用的每个实例。仪器被引用的次数越多,更新所需的时间就越长。而采用抽象框架时,工程师虽然需要开发一个新的仪器驱动程序,但在开发完成之后,只需修改配置文件/数据库即可。
将测试序列转移到新的ATE硬件平台	40至80小时 (每个序列)	<8小时 (每个序列)	有时,整个系统都会升级,所有测试序列都必须迁移到新系统。通常,这些新系统采用的仪器与旧系统完全不同。在这种情况下,无论是否使用抽象框架,都必须开发新的驱动程序,但这些驱动程序开发出来之后,必须更新测试序列才能使用。如果使用非抽象序列,这一过程非常麻烦,有时还不如从头编写序列来得简单。然而,抽象序列通常可以在一天之内通过配置文件进行更新,而不必触及测试序列软件。

您可以对前面假设的商业产品公司的场景进行扩展,并基于这些数字看看开发集成抽象框架是否有意义,或者何时开发有意义。

首先,假设您自己开发全部四个测试序列。您必须从开发框架开始。在标准的非抽象场景中,必须开发仪器特定的驱动程序。在第二个场景中,主要使用现成的抽象开发驱动程序。在第三个场景中,您需要开发集成式HAL/MAL。

任务	开发时间（标准）	开发时间（现成的抽象）	开发时间（集成式HAL/MAL）
框架/驱动程序开发	80小时	100小时	500小时
测试开发(4个测试)	80×4=320小时	80×4=320小时	40×4=160小时
总计	400小时	420小时	660小时

表5 集成式HAL/MAL所需的前期开发工作量最大

当完成初始开发时，集成式HAL/MAL方法比标准方法多花费约260小时，但使用现成抽象的方法仅多花费约20小时。然而，初始开发结束并不意味着测试项目结束。

六个月后，研发部门发现需要执行更多测量，因此系统中的32通道多路复用器已无法满足需求，需要使用4×128矩阵代替。现在您必须开发一个新的驱动程序，并更新每个测试序列，以使用矩阵代替多路复用器。然而，如果您使用现成的抽象驱动程序，则不需要开发任何驱动程序即可处理新矩阵，并且序列中的函数调用也不需要更改—只需更改详细信息即可。如果使用集成式HAL/MAL，序列更新只需要在通道配置文件中完成。

任务	开发时间（标准）	开发时间（现成的抽象）	开发时间（集成式HAL/MAL）
新驱动程序开发	4小时	0小时	0小时
由于采用新矩阵，需要更新2个简单的测试序列	2×4=8小时	2×2=4小时	2×1=1小时
由于采用新矩阵，需要更新2个复杂的测试序列	2×16=32小时	2×12=24小时	2×2=4小时
额外时间	44小时	28小时	7小时
总计	444小时	448小时	667小时

表6 集成式HAL/MAL方法更容易更新，但仍需要执行更多的开发工作

即使到现在，集成式抽象层还未开始获得回报，而现成的硬件抽象几乎已经达到收支平衡。现在假设又有一个新的测试项目，需要对额外四个成品板进行测试。遗憾的是，您的工作太忙，无法自己开发这些序列，因此招聘了两名新的测试工程师。您必须对他们进行系统培训，然后让他们开发序列。

任务	开发时间（标准）	开发时间（现成的抽象）	开发时间（集成式HAL/MAL）
培训/学习时间	60×2=120小时	50×2=100小时	40×2=80小时
测试开发(4个测试)	120×4=480小时	100×4=400小时	60×4=160小时
额外时间	600小时	500小时	240小时
总计	1044小时	948小时	907小时

表7 当开发更多测试或需要进行重大变更时，集成式HAL/MAL方法从长远来看能够获得回报

此时，最初开发框架的500小时投入已经开始获得回报，比标准开发方法缩短了100小时。随着新测试的开发、变更以及产品生命周期的持续，初始投资将持续获得回报。

在主观层面，使用抽象还会获得更多回报，但这些回报难以用数字来呈现。开发测试所需的时间也会大大缩短，因为采用HAL/MAL后，在硬件尚未完全定义之前开发软件变得更容易。通过维护标准框架，您可以确保在一个单独的存储库中添加新的驱动程序和测量，可以管理错误，并且可以减少工程师之间的代码差异。标准化有助于让每个人(测试工程师、制造工程师和技术人员)保持一致，从而更好地为系统提供支持。尽管如本文档所述，抽象还有诸多其他优势，仍需根据个人能力与ROI计算结果确定哪种级别的抽象符合您的特定需求。

附加信息

TestStand

TestStand是一款行业标准的测试管理软件，可帮助测试和验证工程师更快速地构建和部署自动化测试系统。TestStand包含可立即运行的测试序列引擎，支持多种测试代码语言、灵活的结果报告功能和并行/多线程测试。TestStand不仅包含了许多现成的功能，而且也具有高度的可扩展性。因此，全球数以万计的用户均选择TestStand来构建和部署自定义的自动化测试系统。NI提供培训和认证业务，每年培训和认证的TestStand用户超过1000名。

了解有关TestStand的更多信息。

关于Bloomy

Bloomy致力于提供电子功能测试，航空电子设备、电池和BmS硬件在环(HIL)测试以及航空航天系统集成实验室(SIL)数据系统相关的产品和服务；同时提供一流的LabVIEW、TestStand和VeriStand应用程序开发服务。Bloomy是NI联盟合作伙伴，自24年前联盟合作伙伴成立之初便已加入，一直是NI最重要的白金级和精选级合作伙伴。

了解有关Bloomy UTS软件套件(包括集成式HAL/MAL)的更多信息。