测试系统构建实用指南

硬件和测量抽象层

Grant Gothing，ATE研发经理，Bloomy Controls

目录

引言

从初始规划到硬件和软件开发再到最终集成，自动化测试设备(ATE)的设计和开发提出了诸多挑战。 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通常在硬件之后，因此许多开放式项目都留给软件工程师处理。良好的规划有助于降低常见的风险，但无法防止所有问题，特别是在快速的测试开发周期中，许多问题都出现在最终集成阶段。软件比硬件的适应性更强这一观点经常导致“用软件解决就好了”这一误解。 但是，硬件和软件是紧密耦合的，大部分的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仪器成本日益重要，将仪器复用于多个产品的能力就显得非常必要。 而且，不断缩短的开发时间要求工程师并行开发软件和硬件，但开发要求通常没有明确的定义。 另外，测试系统部署之后，长产品生命周期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诸多挑战。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，这通常通过使用模块化仪器和具有可互换测试连接件的互连系统来实现。 但如何让测试软件适应各种硬件呢？ 硬件抽象层(HAl)和测量抽象层(MAl)是实现该任务的其中两种有效设计方法。 与测试序列采用针对特定设备的代码模块不同，抽象层可将测量类型和针对特定仪器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，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可赋予硬件和软件工程师以并行工作的能力，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常用函数库，进而实现标准化和可复用性。 这使得测试开发人员可以专注于每个被测单元(UUT)序列的开发，减少花在编写底层代码的时间。

ATE软件挑战

开发	维护
紧迫的开发周期 需求定义不清晰 测试步骤不断变化 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	长产品生命周期 ▪ 仪器故障或过时 ▪ 仪器变更 产品更新 ▪ 测试步骤更改 ▪ 需要新硬件 制造工程师通常不是原始的测试开发人员

软件抽象的好处

开发	维护
软硬件分离 序列开发与代码（驱动程序）开发分离 为仪器提供通用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	降低被淘汰或硬件更换的风险 ▪ 降低对特定仪器的依赖性 ▪ 无需修改测试序列即可更换硬件 降低代码复杂性，方便未来测试支持/更改 提高代码跨平台兼容性

了解HAL和MAL之间的区别非常重要。 HAL属于代码接口，可使应用软件能够在通用层次上而不是针对特定设备的层次上与仪器进行交互。 通常，HAL定义了仪器类或者仪器必须符合的类型和标准参数和功能。 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工具来执行某个任务的一种方式。 这些共同组成了一个硬件抽象框架。

图1. 抽象框架的上层盖图

打印机对话框是HAL/MAL的一个典型的日常用例。 当您通过计算机进行打印时，不必打开终端，只需将原始串口、USB或TCP命令发送到打印机进行初始化和配置，然后发送要打印的数据。 硬件驱动器负责实现配置和打印方法。 所有打印机制造商均遵循特定标准，将这些方法部署到其驱动器中，使得打印机方便易用。 在一个硬件上执行多个任务所使用的通用接口就是HAL。 那么我们需要编写代码来调用HAL的抽象方法，以便配置和打印文档吗？ 不用，当您选择打印时，就会显示打印对话框。 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上。 这就是MAL，它使您能够直观地运行所有打印机，而无需了解打印机设备的底层功能。 与打印文档一样，ATE HAL定义了每种仪器类型必须遵循的公共底层任务集，而MAL则提供了执行高级操作来驱动仪器的一种通用方法。

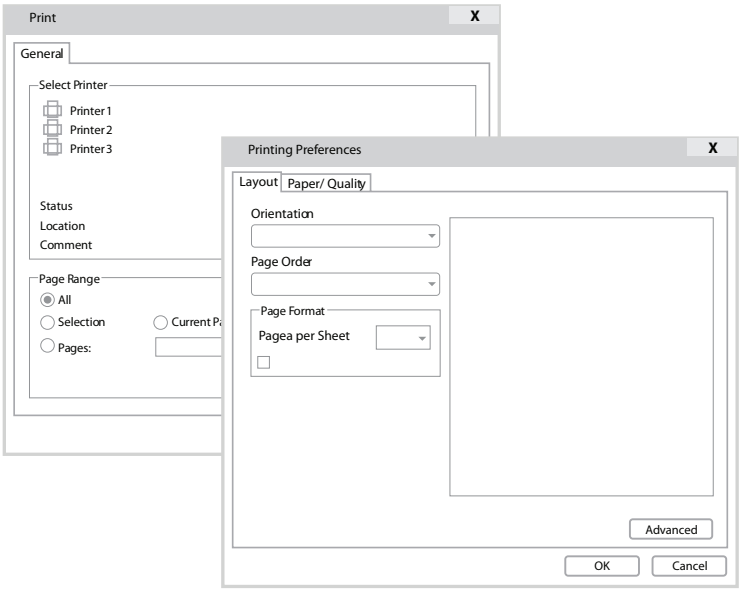


图2.打印机对话框是HAL/MAL的典型日常用例。

现有的HAL/MAL

测试和测量工程师采用了许多方法来解决HIL和MAL。其中大部分方法可以立即使用，或者集成到更大型的自定义HAL/MAL方法中，以便以最少的投入来实现功能扩展。 以下是几个最常见的例子。

抽象	描述	类型	优点	缺点
特定厂商的驱动系程系列 驱动程序（NI-DAQmx、模块化仪器、Pickering PILPXI）	HAL	厂商特定的驱动程序系列为某些供应商的通用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几十到几百个仪器。 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 ▪ 有详细文档记述且经过测试 ▪ 提供所有可用的功能 ▪ 学习周期短— 同一个驱动程序可控制该系列的所有仪器	▪ 仅对每个厂商的特定驱动程序有效 ▪ 并非所有仪器都支持所有功能
业界标准的接口	HAL	IVI是仪器驱动程序软件的标准，可提高仪器互换性，并在与符合IVI标准的仪器连接时提供灵活性。 该标准定义了13中仪器类型的规范，目前许多制造商都遵循的这一标准，这使得一个驱动程序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、直流电源、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 ▪ 兼容众多 GPIB、串行和LXI台式仪器 ▪ 即插即用 ▪ 所有驱动程序采用标准编程模式 ▪ 上层仪器 API ▪ 支持模拟设备	▪ 只指定API，而非实现 - 两个“可互换”实现可能为同一测量返回不同的结果 ▪ 无法用于不兼容的仪器 ▪ 可能无法实现所有功能 ▪ 可能会出现仪器无法支持的功能
Switch Executive	MAL	Switch Executive是一款开关管理和路由应用程序，可允许将兼容的开关矩阵和多路复用器仪器组合到一个虚拟开关设备中。 这个虚拟开关可以使用命名的通道和路由直观地进行配置和激活。	▪ 直观的开关路由设置和操作 ▪ 基于UUT或测试为中心的名称定义通道和路由 ▪ 定义无连接路由以增加安全性	▪ 要求开关兼容NI或IVI标准 ▪ 无法支持通过NI-DAQmx控制的继电器

表1.现成软件抽象层

现成的抽象能够帮助我们只需少量的自定义即可获得诸多功能。但是，这些抽象无法统一起来。IVI驱动程序和NI产品驱动程序是兼容仪器的最佳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由，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）。通过使用统一的HAL/MAL，您可以更有效地开发以UUT为中心的序列，这些序列可与各种仪器连接，更好地处理仪器通道和连接的变更。

虽然HAL和MAL提供了诸多好处，但它们通常需要工程师基于过去的经验进行大量预测。 我们需要考虑不同层次的抽象。 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，并且如果没有正确规划，可能风险非常高。 不恰当的初期假设或实施可能会产生积极和消极的持久后果。 重要的是找到满足您的特定需求的硬件替换产品。 如果您不确定如何进行，那么简单开始，保持可扩展性，并尽可能使用内置的抽象。

背景

为了更好地理解HAL/MAL的实现方式，您必须了解自动化测试软件的结构。 从最高层次看，自动化测试软件采用测试执行程序（或测试执行软件），例如TestStand。 执行程序会调用一系列测试步骤，通常是代码模块或函数，这些步骤使用LabVIEW软件的图形化语言、.NET或Activex进行开发。结合针对特定仪器的自定义方法，这些代码模块便具有特定用途，例如使用DMM和开关的开关式DMM；或使用电源和示波器可进行纹波测量的电源。 这样做有一定的好处，可让每个开发人员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，它要求每个测试开发人员都精通底层软件（如LabVIEW）。

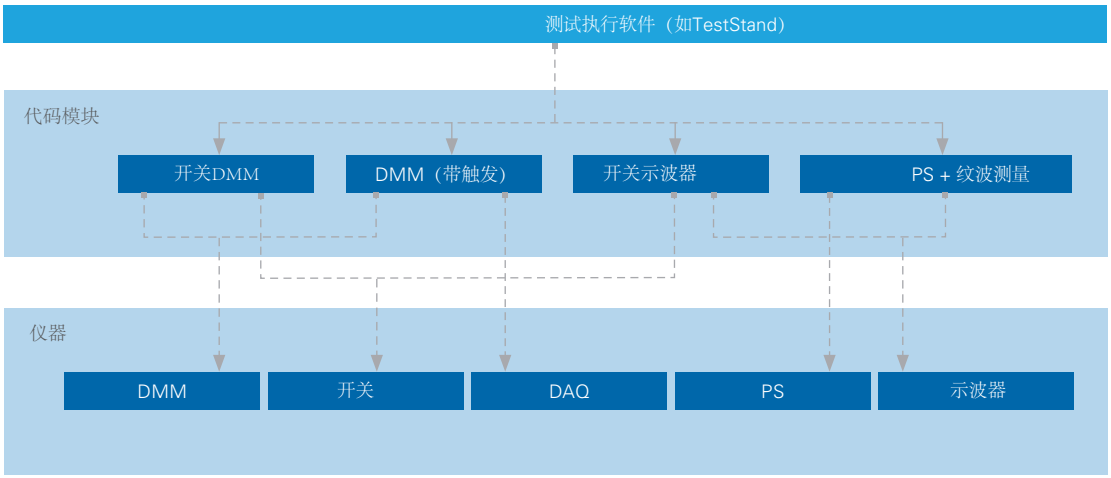


图3.非抽象自动化测试软件的解剖

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代码仪器连接。这导致测试序列与特定仪器和特定驱动程序代码紧密耦合。 不采用HAL/MAL框架时，通常会发生四个不可避免的问题：

- **由于过时或需求变化，仪器需要更改** - 如果不采用抽象，就需要更改每次调用该仪器的驱动程序，在典型的测试序列中这可能涉及几十个步骤。 每次仪器更换都会导致一系列软件更改。
- **由于新需求的出现而导致的驱动程序功能更改** - 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，尤其是在更改输入或输出时。 而且，直接调用驱动程序代码模块需要每个测试开发人员理解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。 要使用所有这些功能，测试工程师还必须是精通软件的工程师。

- **测试序列是从仪器的角度进行开发** -通过使用针对特定仪器的驱动程序，所有测试序列使用以仪器为中心的通道名称（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。由于我们从UUT的角度开发测试程序，这使得开发和调试变得非常困难。 而且，任何测试变化都需要对仪器、连线和互连系统有非常深入的了解。
- **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，通常不知道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、通道编号和连接保留占位符。 任何硬件信号更改都需要更新测试序列。

例如，使用自定义方法，多路复用DMM测量代码模块看起来会类似于下图，下图显示的是通用开关式DMM labVIEW VI。 代码模块包含针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来连接开关，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，您必须知道要填充哪些字段，以及从仪器的角度来看，需要什么通道、拓扑和配置。 您还必须确保将开关和DMM测量数据正确地传递到代码模块。

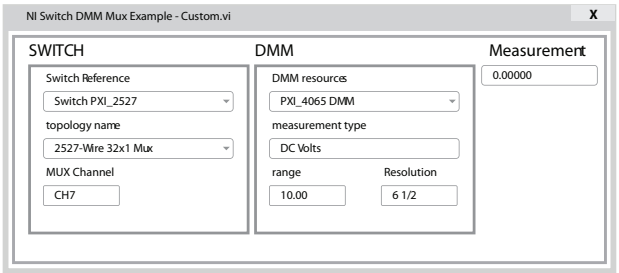


图4.LabVIEW中典型的多路复用DMM测量应用程序前面板

从测试执行程序的角度来看，我们可以调用代码模块来执行特定的功能（复用DMM）。 该函数可对开发该函数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来与特定仪器进行通信。 每个外部项都取决于其内部调用。

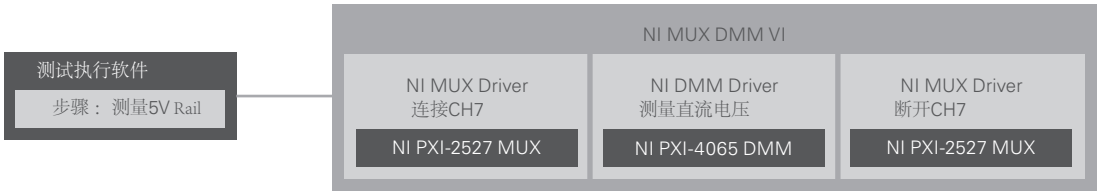
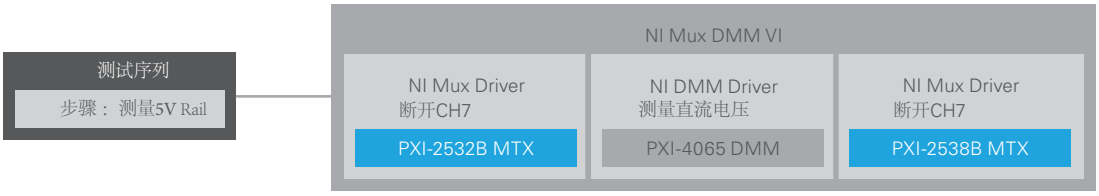


图5.执行多路复用DMM测量的嵌套式命令调用

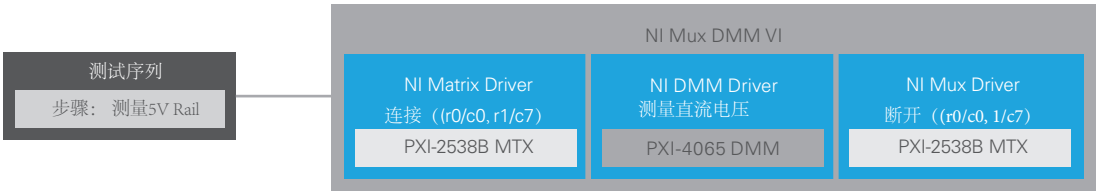
如果必须更改仪器，则依赖关系中的每个函数也必须更改。 例如，如果初始多路复用器的通道不够用，并且需要变换为较高通道数的矩阵，则由于依赖关系链，我们需要进行一系列更改：

1. **仪器**——PXI-2527多路复用器更改为PXI-2532B矩阵
2. **驱动程序**——NI多路复用器驱动程序更改为NI矩阵（行/列而不是通道）
3. **代码模块** - NI Mux DMM VI必须更改为NI Matrix DMM VI
4. **函数调用** - 代码模块的测试执行调用必须更新
5. **序列** - 测试序列必须针对该代码模块的每次调用进行更新

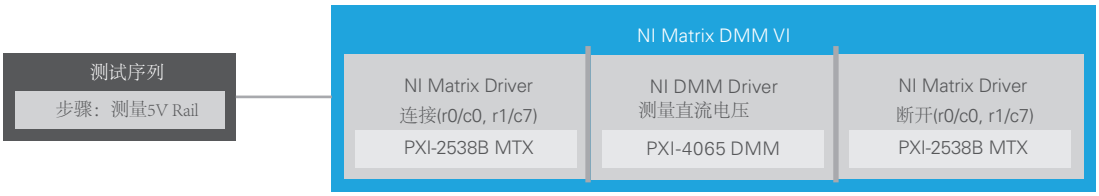
步骤1：更改仪器



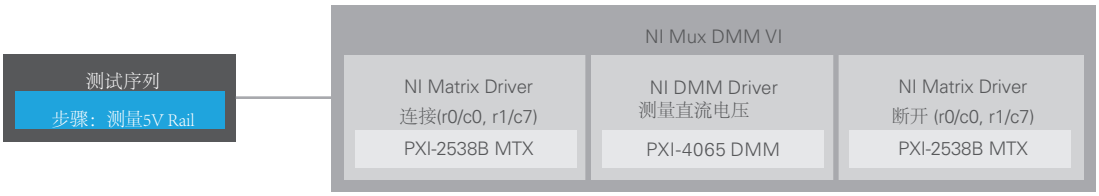
步骤2：更改驱动仪器



步骤3：更改代码模块



步骤4：更改函数调用



步骤5:更改测试序列

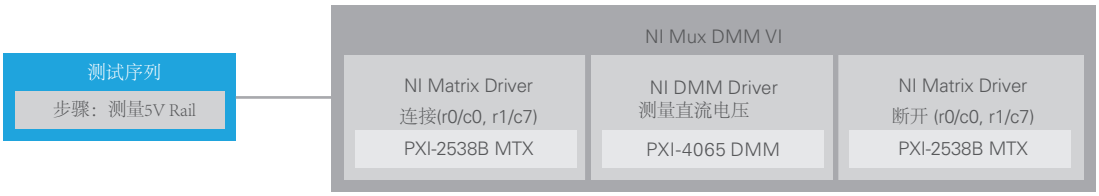


图6. 依赖关系链所需的非抽象更改

采用抽象

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。 测试执行程序没有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，如“信号输入”、“信号输出”、“连接”、“电源”和“负载”。 它们还会接收针对特定测试的参数（而不是针对特定仪器的参数），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）。 映射框架使用配置文件将通用操作的特定测试参数转换为特定仪器参数，如仪器参考、通道编号、矩阵行和列、GPIB地址和仪器配置约束。 框架与HAL连接，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，并从配置文件中提取针对特定仪器的参数。

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成分（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、扒炉），框架就是将所这些整合在一起的指令。

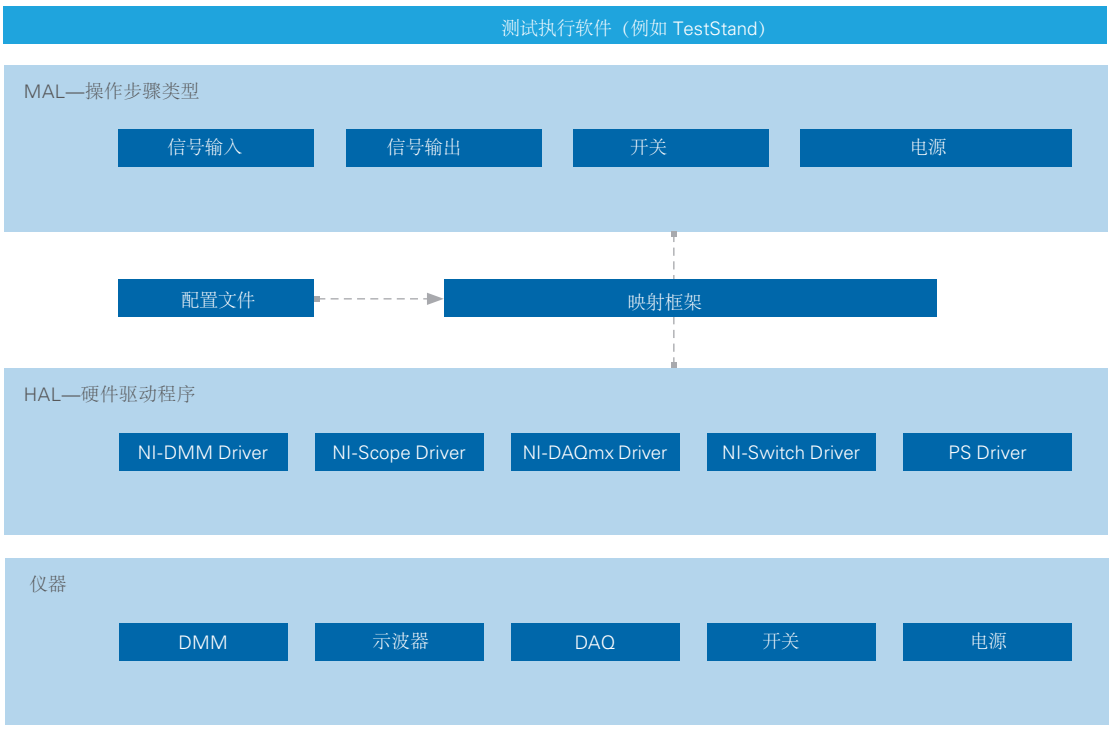


图7.抽象自动化测试软件的解析图

本节将继续介绍基于抽象的多路复用DMM示例。 在本示例中，测试执行程序使用针对特定步骤的输入参数5V Rail来调用通用步骤类型“信号输入”。 在本例的框架中，信号输入定义为三个设备操作： 连接信号路由、读取测量设备数据、断开信号路由。 这些操作使用5V Rail参数传递到映射框架。 映射框架读取配置文件以查找5V Rail的仪器和通道详细信息。 这些对应于PXI-2527多路复用通道7的连接，以及DC电压模式下的PXI-4065 DMM测量。 然后，框架调用适当的抽象驱动程序NI-Switch和NI-DMM，以与配置文件定义的特定仪器进行通信。

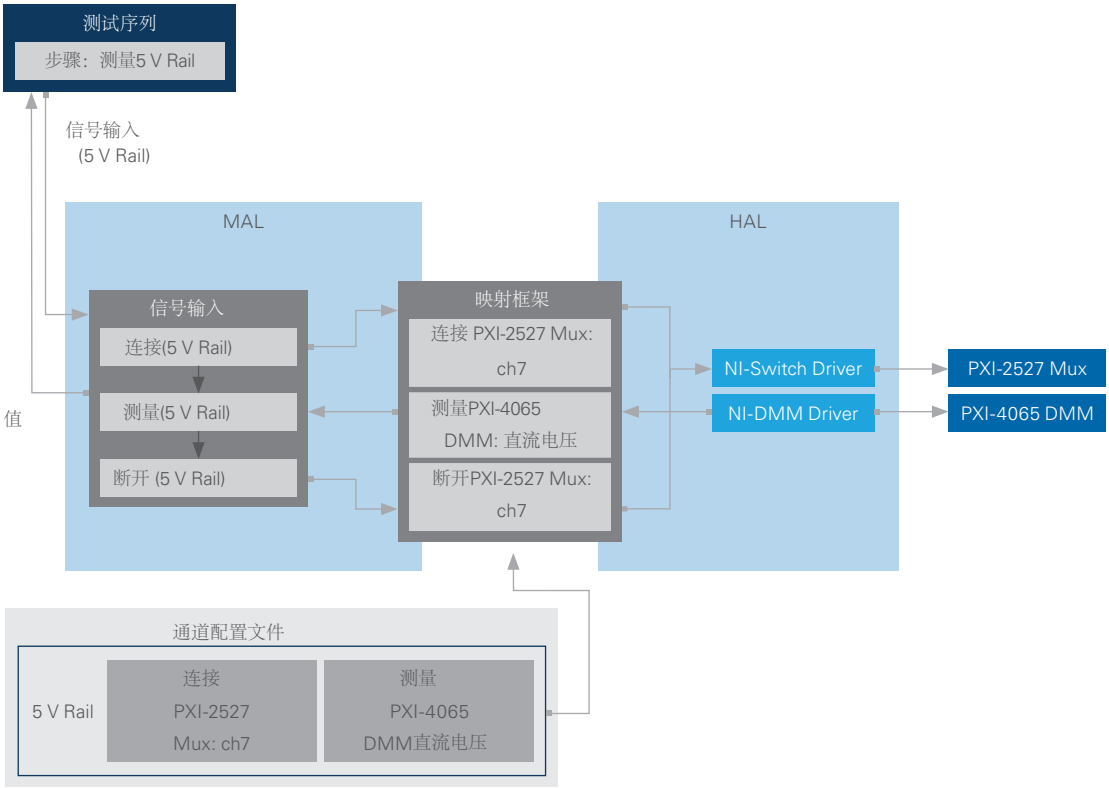


图8. 使用抽象框架对DMM测量进行函数调用

在不采用抽象情况下，PXI-2532B矩阵由PXI-2527多路复用器替代，这时使用HAL/MAL框架来执行相同的更改会更为容易。 由于所有仪器细节存储在配置文件中，并且HAL提供了一个与类似仪器交互的公共接口，所以只需要改变配置文件即可。 通过将PXI-2527 Mux： Ch7替换为PXI-2532B Mtx: r0/c0, r1, c7， 映射框架可自动提取更新的详细信息，并使用新参数调用新矩阵， 完全无需更改测试顺序或代码模块。

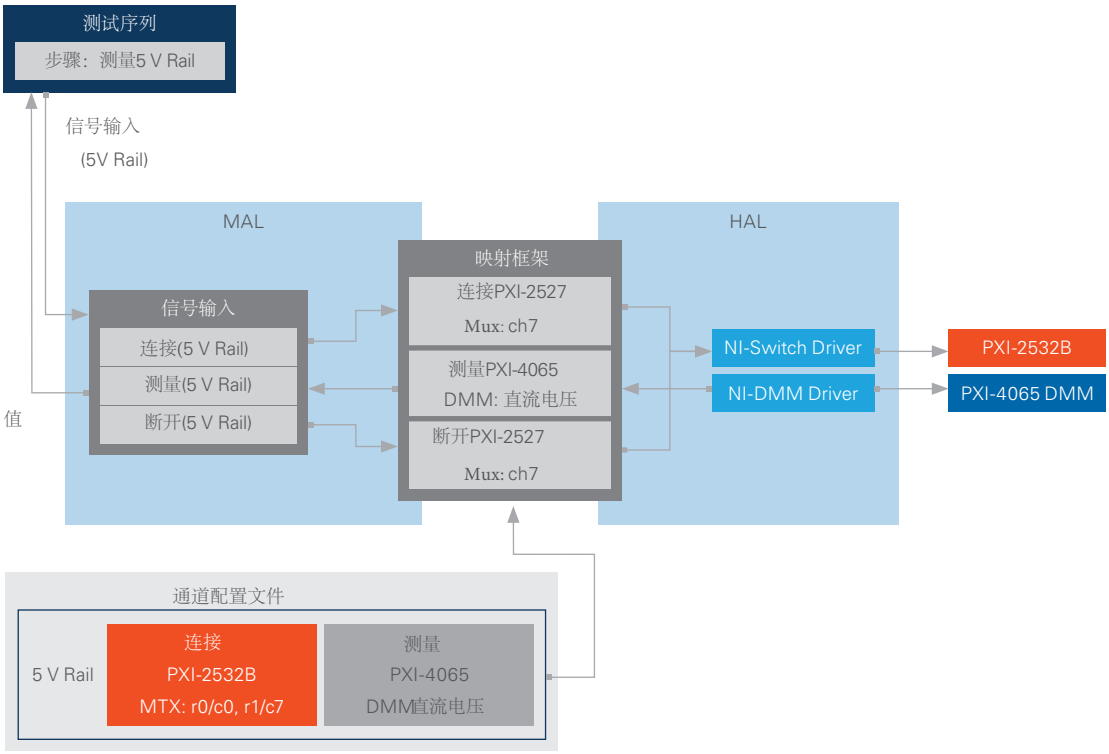


图9.抽象可帮助您以最少的软件更新来轻松更新硬件 - 只需更新配置文件即可。

方法

在决定抽象框架时要考虑的最重要问题是所有其他决策所基于的抽象范围。 一种极端是在没有抽象的情况，每个硬件接口就可直接调用仪器驱动程序。 另一个极端是完全采用抽象，组件、通信协议、测量和配置格式之间的每一个可能接口都有一个抽象定义。 本节探讨了这些可能接口的一些选项。

选项1: 仪器驱动程序

仪器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。这种方法需要开发底层代码模块来与特定仪器连接。 这些通常称为底层驱动程序或仪器驱动程序，然后由高级代码模块或直接由测试执行程序调用。 下面的框图显示了针对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，则必须同时更改驱动程序和更高级别的调用。

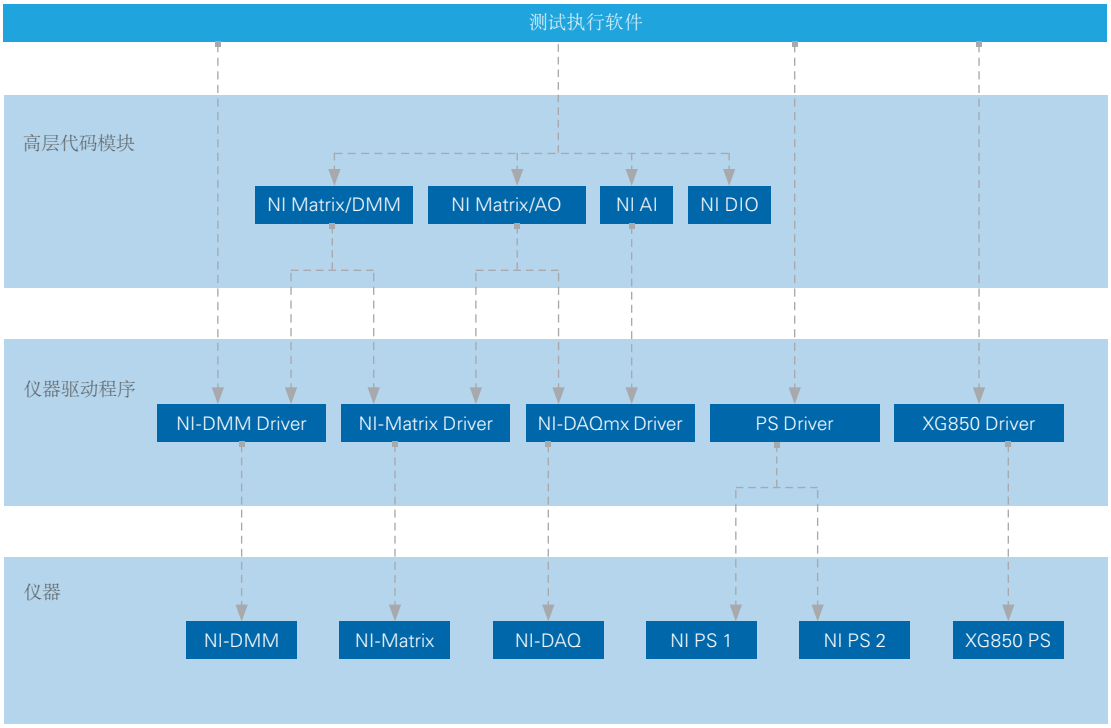


图10.无抽象自动测试软件的仪器特定驱动程序方法概述

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鲁棒性方面，仍然有值得我们借鉴的地方：

- 开发或使用仪器驱动程序包来与每个仪器连接。
  - 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例程（例如同一系统中的两个相同电源）。
- 开发包装仪器驱动程序，简化仪器界面。
  - 已有的驱动程序可能会包含几个难以理解的函数。 您可以将已有的全功能仪器驱动程序包装到更简单的包装仪器驱动程序中，以提高易用性。
- 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，减少了竞争条件，并允许通过一个统一的位置管理仪器状态。
  - 包装仪器驱动程序（如果开发的话）应该只有一个入口点。
  - 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。
- 不在驱动程序级别实现针对特定测试的功能。
  - 针对特定测试的算法应该由更高级别的代码模块或在测试执行程序中实现。

- 确保仪器驱动程序不会相互识别。
  - 高级代码模块或测试执行程序调用单个仪器驱动程序时应执行多仪器交互。

选项2: 现成即用HAL/MAL

将抽象整合到仪器驱动程序架构内的最快方法是使用已有的HAL和MAL。 虽然目前市面上仅提供有限的完全集成的HAL/MAL抽象框架购买选项，但是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是专门针对开关连接和路由而开发的MAL。 基于这些已有的抽象程序来构建代码模块，您就能够以最小的开发工作量提高ATE软件的适应性和抽象。

现成即用的硬件抽象

已有的硬件抽象会使用常见的底层接口来连接各种仪器。 这减少了所需仪器驱动程序的数量，同时降低了系统中仪器更换的影响。 测试执行程序和更高级别的代码模块可以引用常规的驱动程序，从而减少开发工作量和仪器更改的影响。当实现下面定义的抽象类型之一时，特定接口的I/O是固定的。 因此，仪器更改通常不会导致代码模块更改。

您可以使用两种方式来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针对特定供应商的驱动程序，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标准的方法来与多个供应商的某些仪器类型连接。 您可以使用这些标准来开发仪器驱动程序，以便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针对特定供应商的驱动程序，可与仪器常用的产品系列进行通信。 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-DCPower和NI-Scope）和Pickering PILPXI。 仪器系列驱动程序实现了相关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多数功能（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。 通过遵循行业标准，制造商就可以使其仪器与其他类似仪器进行互操作。 两个最常见的标准是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常用参数。 SCPI命令可以通过各种通信协议实现，包括GPIB和LAN以及串行协议。 通过开发单个符合SCPI标准的驱动程序，您可以与多个相同类型的仪器（DC电源、电子负载等）进行通信，而无需开发特定于仪器的驱动程序。在开发SCPI驱动程序时，请注意，虽然SCPI定义了一个常用的命令和语法标准，但是不同的供应商有时在实现标准时会有一些微小的差别，使得开发100%标准的驱动程序变得困难。 在选择符合SCPI标准的仪器和开发驱动程序时，请务必要密切注意每个仪器的命令细节。

IVI

IVI仪器驱动程序软件的标准，可提高仪器互换性，并为连接到符合IVI标准的仪器提供灵活性。 该标准使用VISA定义了I/O抽象层。 由于SCPI已经整合到IVI中，因此许多符合SCPI标准的仪器都符合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，这使得每种类型的单个驱动程序能够控制来自不同供应商的多个专用仪器。 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和台式仪器都遵循IVI标准，而且许多编程语言和测试执行程序已经有一些现成的驱动程序。

通过使用IVI驱动程序为IVI兼容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一样。 您可以针对每种类型使用一个驱动程序集，以连接各种可互换仪器。 相比使用仪器特定的驱动程序，如果使用类似功能来替换兼容IVI的仪器，则代码和序列更新将会大大减少。 然而，尽管IVI驱动程序可以实现兼容仪器的大多数功能，但是一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，尽管两个仪器可以执行相同的IVI函数，但得到的结果可能不一定相同。因此，每做出一次更改，都需要验证和测试仪器的功能。

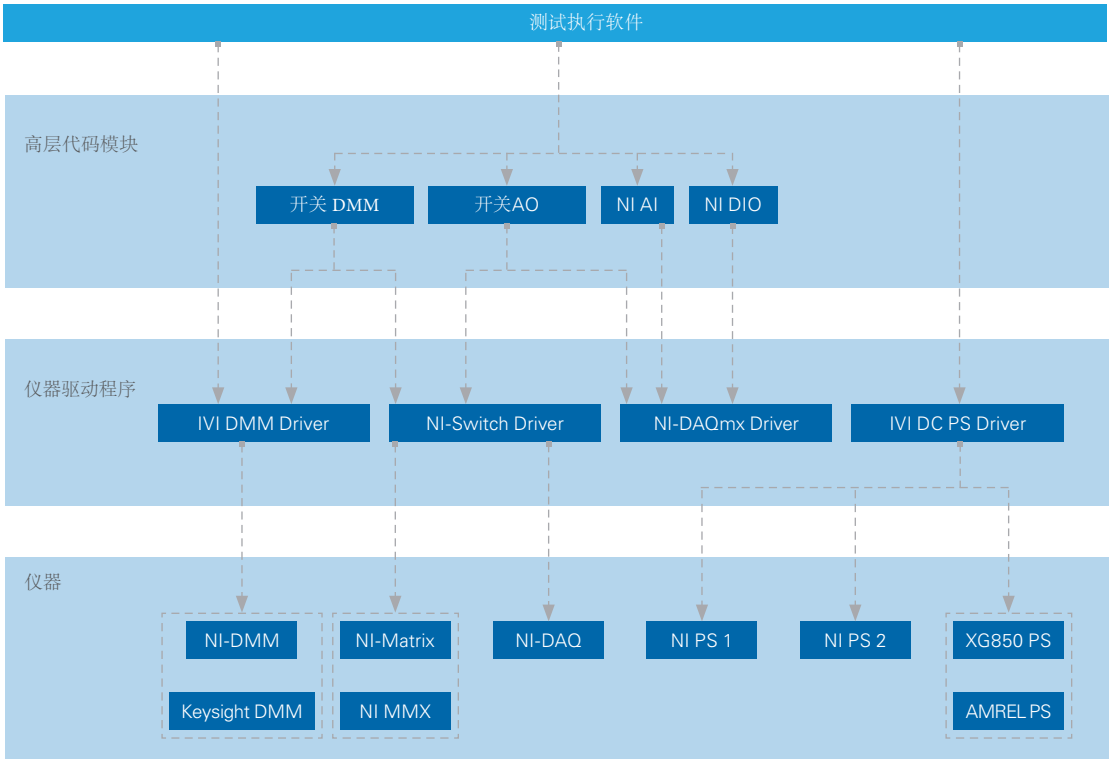


图11.采用现成抽象的自动化测试软件概述

现成即用的测量抽象

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。 测量抽象是非常有限的。由于测试系统具有高度自定义性，因而很难为测量操作定义一个标准。 最常用的现成即用测量抽象层是Switch Executive，这是一款开关管理和路由应用程序，可允许将兼容的开关矩阵和多路复用器仪器组合到一个虚拟开关设备中。 这个虚拟开关可以使用用户命名的通道和路由直观地进行配置和激活。虽然Switch Executive仅适用于符合NI-Switch和IVI标准的设备，但却提供了一种极佳的方法来从UUT或测试的角度定义开关路由。

首先，Switch Executive提供了一个图形化配置实用程序，用于在单个仪器和多个仪器之间设置开关通道名称和路由。 行、列、通道和路由组都可以进行配置和命名，以便直观地设置开关机制。

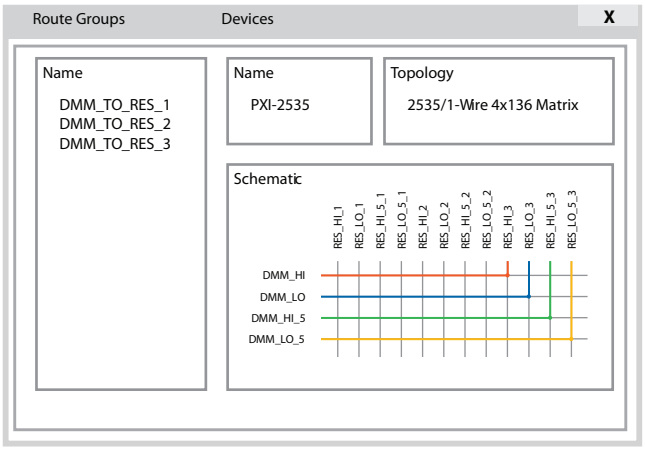


图12.Switch Executive MAL配置接口

其次，Switch Executive可集成到LabVIEW和TestStand中，提供了强大的接口，可按名称设置和查询预配置的路由。当与TestStand测试执行软件一起使用时，Switch Executive可以逐个步骤执行，以便在执行步骤的代码模块之前为开关仪器提供命名接口。

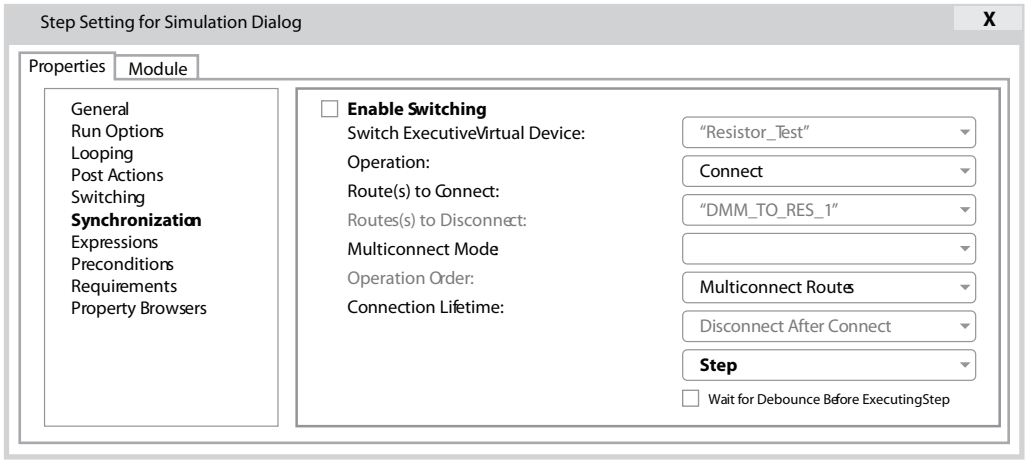


图13.Switch Executive MAL测试设置

Switch Executive是一个有用的MAL，可将开关连接抽象为特定测试名称，而不是特定仪器名称。 当与IVI开关硬件抽象结合使用时，就形成了一个非常出色的集成HAL/MAL框架。 但是，Switch Executive不适合用于使用非IVI开关或外部数字输出控制继电器的情况。 此外，Switch Executive仅适用于开关路由，无法扩展到其他测量类型。 如果需要开关之外的集成HAL/MAL框架，就需要自定义开发代码。

选项3: 集成式HAL/MAL框架

集成式HAL/MAL框架提供了一个结构来实现由测试执行程序(MAL)调用的高级动作，与底层驱动程序接口相连接，并在两者之间映射细节。 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。 这些代码模块类型都是由一组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及其之间的交互如下所示。

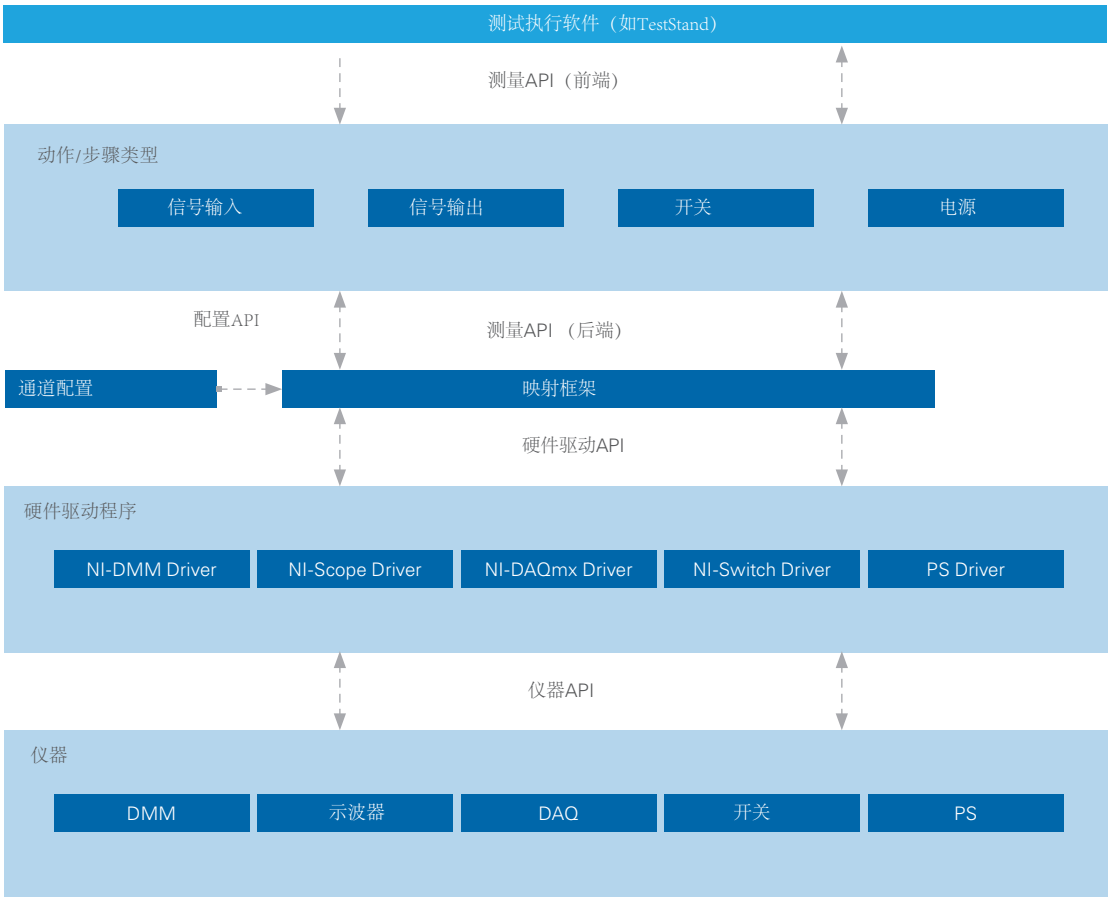


图14.采用集成式MAL和HAL的自动化测试软件概述

三种类型的代码模块是：

- **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）。 动作可以简单到调用一种仪器类型的一个函数，例如进行开关连接； 也可以复杂到调用多个仪器的多个函数，例如将开关连接与设置电源电压、电流和启用状态组合。 这些代码模块通过测量API来定义它们的方法和参数。
- **映射框架** - 映射框架是使用配置文件中的默认值将高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与操作交互。
- **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）转换为特定仪器通信（SCPI、IVI、NI-DCPower和专用通信）。 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的其中一个：

- **测量API** - 测量API定义高级动作及其特定参数。 这是MAL的定义。 测量API定义了所有动作必须遵循的公共框架，然后允许每个操作定义执行特定功能所需的API（参数和方法）。每个动作必须至少实现后端测量API，映射框架使用该后端测量API将人类可读别名链接到特定的开关和测量仪器以及适当的信道。或者，可以开发API的前端，为每个操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还可为别名定义连接、测量，然后断开连接。

- **配置API** - 映射框架使用配置API填写有关如何从测量API转换为硬件API的详细信息。 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可以规定配置文件为Microsoft Excel文件，并且每个信号别名应该具有以下属性： 名称、类型、连接详细信息、仪器、仪器配置和换算关系。

- **硬件API** - 硬件API是定义特定类型仪器必须实现的通用参数和方法的抽象API。 此API定义了HAL。 例如，DMM Hardware API可指示所有DMM必须能够初始化、配置（电压、电流、电阻、量程、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负责实现用于控制其特定仪器的必要函数和命令。 这是将在仪器特定代码接口中使用的同一个API，并且负责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， 其中详细解释了每个代码模块的输入和输出。

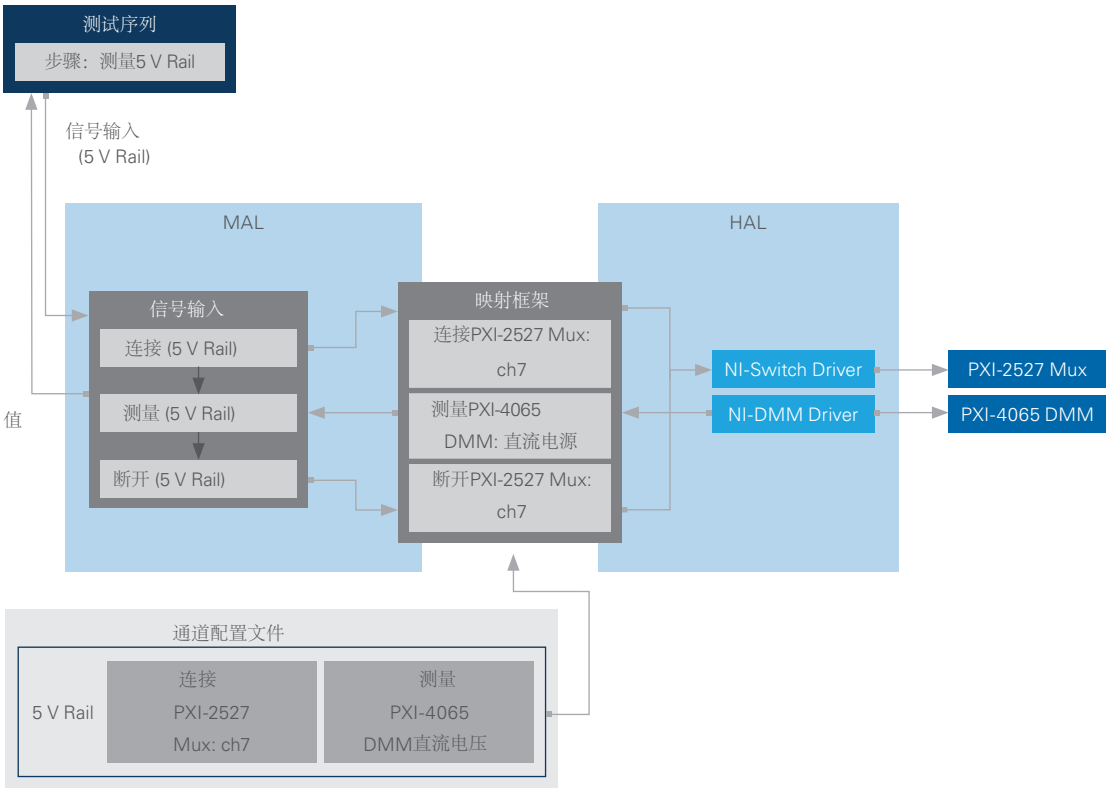


图15.使用抽象框架进行复用DMM测量

在该示例中，信号输入块是操作代码模块，定义了信号输入应当执行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定义了代码模块需要从测试执行程序接收的别名，并将其传递到映射框架，然后从映射框架获取返回值传递回测试执行软件。

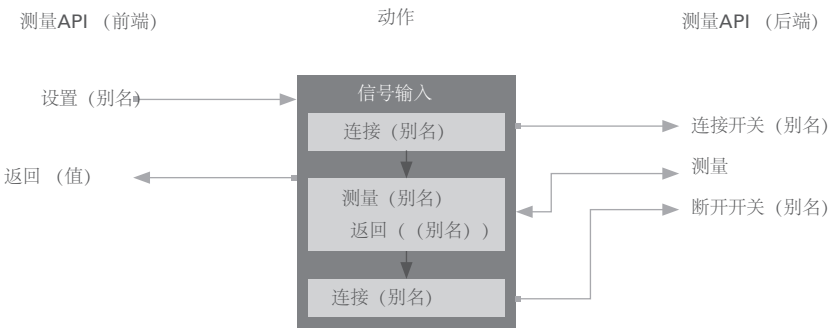


图16. 信号输入的MAL Action API示例

映射框架通过测量API从操作接收命令。 然后通过配置API解析配置文件中的别名数据，以获取正确的仪器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API将特定仪器的信息传递到适当的驱动程序。

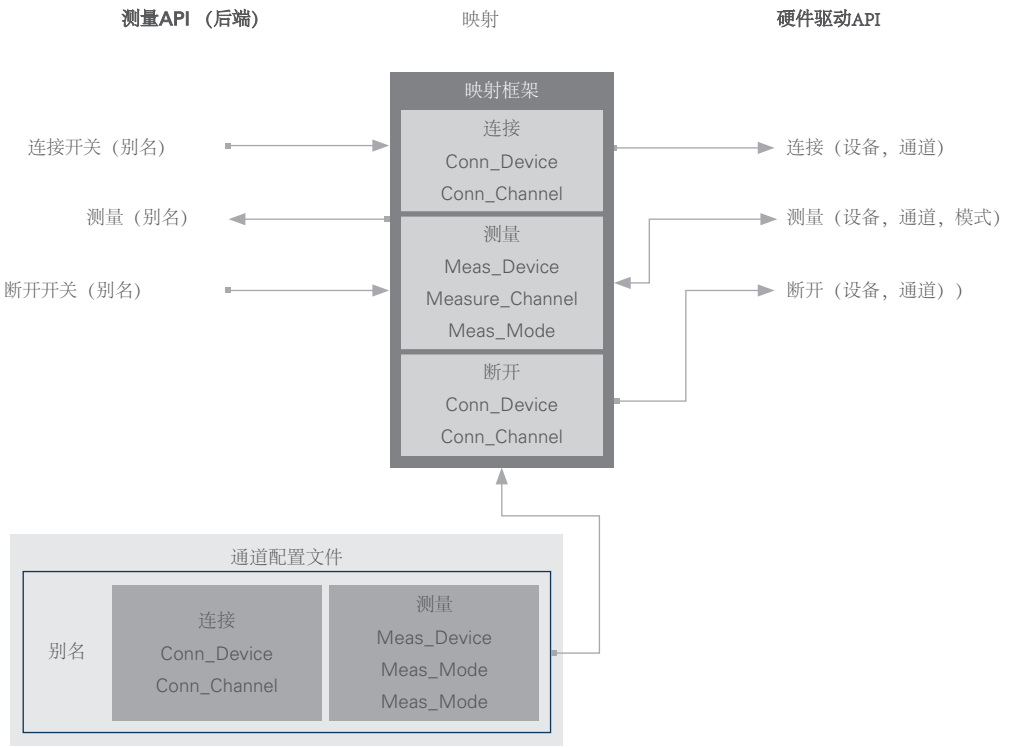


图17.信号输入的映射框架API示例

映射框架使用通用的硬件驱动API调用各个硬件驱动程序。 然后每个驱动程序解释通用设置的细节，并使用现成即用的方法和参数与特定仪器进行通信。

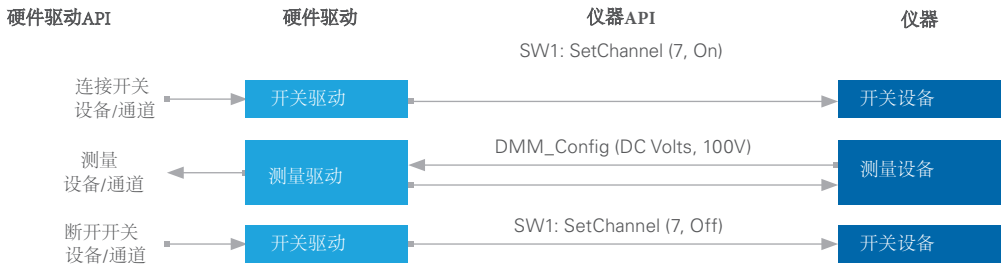


图18.信号输入的硬件驱动程序API示例

选项4: HAL/MAL插件架构

插件是集成框架的有用补充。 一个真正的插件仅是一个软件组件，可以在部署后修改而不需要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，但同时也通过明确地限制添加或修改功能的范围和风险来简化软件回归测试。无插件框架在每次需要新测量类型、仪器驱动程序或配置格式时都必须进行重新构建。由于没有插件，整个源内置到单个EXE中，所以即使对仪器库进行一个微小的改动，也不能保证其不会无意中影响其它应用特征。由于很难知道源修改的所有可能影响，测试必须非常彻底。

插件架构使得开发人员能够添加或修复插件代码，而无需修改或重新部署底层框架，因而提供了最高级别的软件模块化。 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，通常仅在需要时才加载。 成功的插件架构依赖于完善的接口设计。 换句话说，为了在测试框架中使用插件，框架必须知道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道在哪里找到这些插件即可。

尽管这些是抽象框架一些常见的过程、API和代码模块，但它们当然不是唯一的。 每个框架都是独一无二的，并且有自己的要求、过程和实现。 对于一些团队，这种抽象层次可能超过需要。 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。 基于框架架构师和用户的需求和能力，这些API的具体实现的解释因人而异。 一些工程师使用简单的操作引擎来实现所有抽象，一些工程师则使用更高级的面向对象编程，一些工程师使用插件，还有一些工程师更倾向于使用单个代码库。 关键是找到适当的抽象和实现程度来适应您的特定需求和能力。 此外需要理解的是，不是所有问题都可以通过抽象来解决，有时仍然需要针对特定仪器的代码。 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。这可通过允许仪器参考由更高级别的代码模块或测试执行程序获得来实现。 高级开发人员不应被框架所束缚。

无 ○ 部分 ◒ 全部 ●

抽象选项	选项1 无抽象	选项2 现成抽象	选项3 基本自定义	选项4 采用插件
允许使用以下组件替换各个仪器:				
采用相同通信协议的仪器	●	●	●	●
兼容IVI或产品系列的仪器	○	●	●	●
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/连线（只需修改配置文件）	○	◒	●	●
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测量	○	○	○	●

表2.抽象层选项的功能比较

实际场景1

您是一名来自商用产品公司的测试工程师，负责开发新产品的电子组件的功能测试。 您需要测试三个PCBA和最终组装。 现在有一个通用ATE仪器平台，但它已经过时，并且先前的测试程序由于设备故障和过时而存在许多问题。 幸运的是，作为该项目的一部分，一个新的ATE平台已经开发出来，通过可互换的测试头来适应不同组件的仪器测量需求。

您的任务是开发测试序列和代码模块软件来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软件（与先前平台使用的测试执行软件一样）相关的经验，并且有几年的软件应用程序开发经验。 之前已经有多次关于如何使用抽象来减缓现有系统过时这一问题的讨论。您必须决定这是否是正确的解决方法以及实现方法。

是否采用抽象……

您必须做的第一个决定是是否要开发一个抽象框架，且不管抽象的级别如何。考虑现成抽象的优势，如IVI，这个决定的答案几乎是肯定的。 不采用抽象的唯一情况是100%已知项目的寿命，并且永远不需要进行任何更改，但这几乎是不可能的。

您是否需要HAL?

下一个决定是使用什么级别的硬件抽象。 这个决定较为复杂，因为这涉及诸多因素。 硬件抽象通常更容易理解，因此比MAL的实现成本更低。 如果可以合理地使用预抽象驱动程序（例如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，如果系统具有一些符合IVI标准的电源以及不符合IVI标准的电源，则可能需要开发适用于两种类型的抽象电源定义。 定义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。然后可以使用该信息来定义系统内每种仪器类型的常用方法和参数。

目标是覆盖约80%合理预期的功能。与团队成员谈话，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，团队可以规定所有电源的核心功能应该先初始化，然后设置电压/电流/启用状态、回读电压/电流/启用状态，最后关闭。虽然某些其他功能可能以后会被电源使用到，但这些功能并不一定值得成为系统标准化的一部分。如果您对某种特定仪器类型不够了解，或者不确定需要什么功能，请从最小的架构开始。您可以随时添加新功能到标准中，但是在多个驱动程序使用该标准后，很难更改功能的参数或详细信息。

下面的流程图可以帮助您决定什么级别的硬件抽象适合您。如果你不确定答案，可以假设一个更抽象的解决方案，或者比较不抽象的解决方案。更抽象的解决方案需要更多的前期设计，但可以节省长期的时间，而较不抽象的解决方案可以让您更快地运行，但未来可能会出现问题。 要注意的一点是，需要先问一下自己：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。

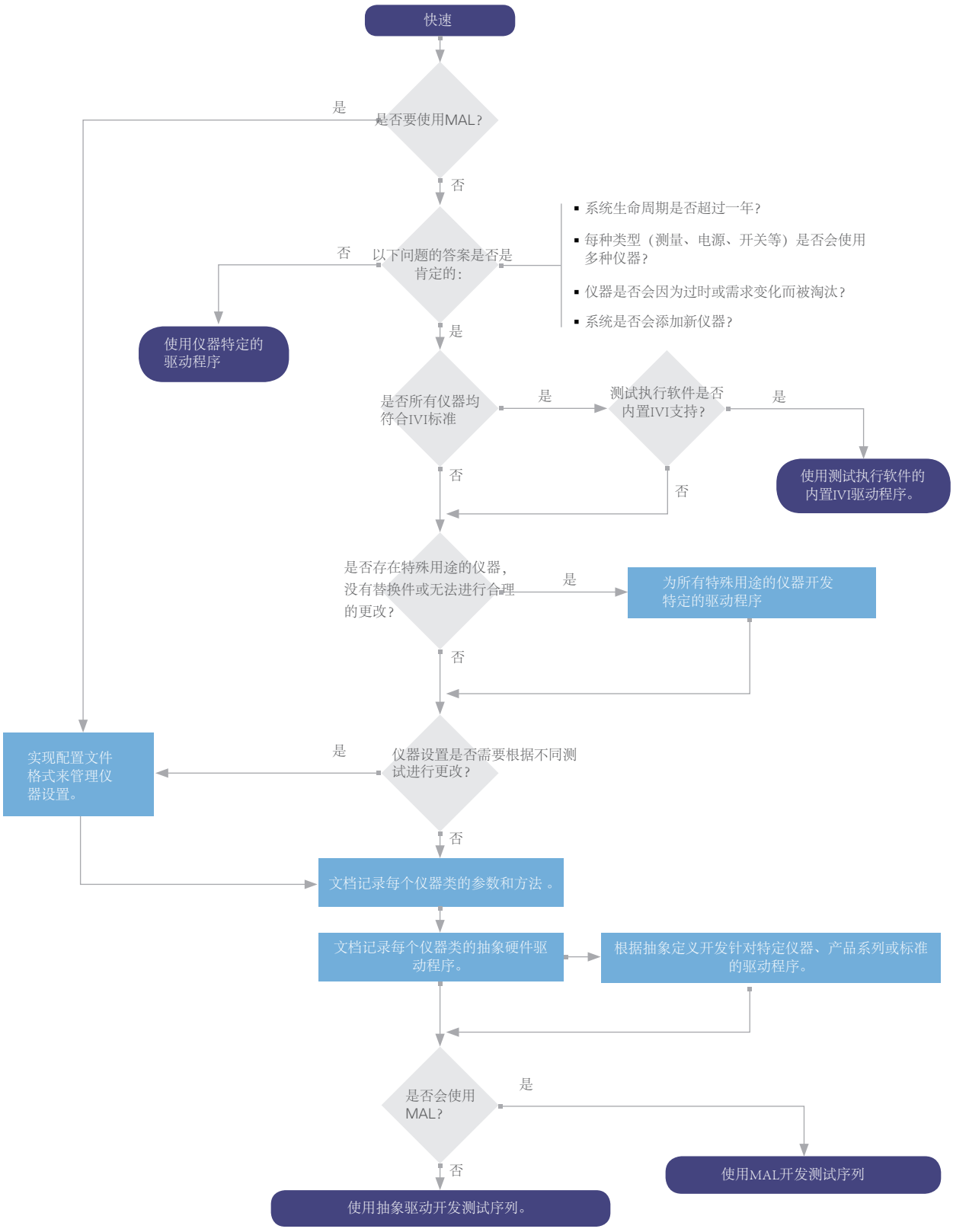


图19.决定要实现的抽象级别的流程图

您是否需要MAL?

HAL的第一个决定是是否需要MAL。 这是因为在不依赖于硬件抽象的情况下，MAL几乎是不可能实现的。 因此，这个问题实质上是在问你是否需要集成的抽象框架。 当团队中有多名测试开发者没有底层软件开发经验时，HAL/MAL是理想的选择。 以下一些主要问题可以帮助您决定是否开发MAL:

- 是否有一个软件架构师可以计划和支持该框架？ HAL/MAL难以在没有架构师/所有者的情况下获得有效的支持。
- 是否有多个测试开发人员只有很少的软件开发经验吗？ 抽象框架的一个最大好处是它缩短了测试开发的学习曲线。
- 系统是否具有长使用寿命，需要支持多种产品？ 这可能需要较大的前期投资，但后期收益将越大，使用频率也越高。
- 您是否愿意开发和支持MAL？ 没有抽象比定义不明确和执行不佳的抽象更好。 简单易用的HAL/MAL可以节省大量的时间；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。

如果你对大多数这些问题回答“是”，那么开发一个集成的抽象框架将会得到长期的回报。

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（其中单位通常是时间）。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。

成本是多少钱?

这是一个难以回答的问题，因为它主要取决于开发人员的经验、编程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。

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常用接口类型	24	48
	每种设备类型的HAL定义	记录每种类型设备的输入和输出和方法	8（每个设备）	16（每个设备）
	每个动作的MAL定义	记录每种类型的测量/动作的输入和输出和方法	8（每个操作）	16（每个操作）
	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 - 大多数底层框架在此处开发	60	120
	抽象设备驱动开发	为每个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（每个设备）	24（每个设备）
	仪器驱动开发	为使用HAL的每个仪器特定驱动程序进行软件开发 -填写到每个特定确定的模板中	4（每个仪器）	24（每个仪器）
	动作开发	为MAL定义的每个动作进行软件开发 - 实现前端和后端API来与测试执行程序和映射框架连接	4（每个动作）	24（每个动作）
总计		开发框架（不包括单个仪器驱动程序）的总时间 - 假设五种设备类型和五个动作，每个设备一个仪器特定的驱动程序	248	776

表3.抽象框架任务和成本

这表明完全集成的HAL/MAL抽象层所需的开发时间可以低至250小时，也可以超过750小时。 取决于抽象的级别，该时间甚至可以超过1000小时。

如何降低成本？

当涉及到软件开发时，成本与复杂性密切相关。 复杂性有好也有坏，取决于其性质。 目标是增加良好的复杂性，同时避免不良的复杂性。 可以增加功能的复杂性是好的。 每个特性通常会增加功能。 使用可扩展、灵活的模块化代码来实现这些目标可能较为复杂。 但是当以简练的方式实现时，这种复杂性是有益的。 由于规划不当、冗余功能和繁琐的意大利面条式代码而产生的复杂性是不好的，因为它增加了开发成本，但没有增加功能。

您可以通过四种方式降低ATE抽象框架的复杂性：

- **预先规划您的架构。**与大多数开发过程一样，前期规划和文档可以节省大量的时间和避免开发过程中的麻烦。 事先规划和文档记述API和代码模块有助于减少交叉功能和不必要的相互依赖，这将使得代码更加强大，并减少不必要的复杂性。 您不必计划每个API和代码模块的每个细微差别，而是定义软件的主要交互、参数和基本功能。
- **不要想得太超前**。 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情况。 虽然前瞻性的方法可能是好的，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。 这只有20%的概率，但却需要80%的时间。 你将最终花费更多的时间来尝试处理假定的边缘情况，而不是专注于将被大多数时间使用的软件。
- **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，可考虑将自定义硬件交互作为框架的一部分，来解决通用接口无法实现的情况。 为您的系统制定切实可行的规则，以减少抽象层。 例如，将配置文件限制为一种格式（ini、xls、database）以减少映射框架的复杂性，或将操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。
- **保持灵活、可扩展和模块化。**虽然灵活性、可扩展性和模块化会增加复杂性，但它们是开发大型架构的最佳工具。 此时非常适合采用插件架构，因为插件架构可定义低级框架，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，而不会干扰已经存在的功能。 精心规划的插件架构不仅可满足已知情况的需求，还可在未来根据新挑战进行扩展。

值得吗？

即使实现得很好，抽象框架的开发可能非常耗时，但因为其收益往往大于付出而得到广泛应用。 几个关键因素可以提高回报率，使您的框架更成功。 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和使用HAL/MAL抽象框架的系统之间的差异。

任务	估计的时间（标准）	估计的时间（抽象）	WHY THE PAYOFF?
为新测试工程师提供测试软件平台培训	60小时 每个工程师	40小时 每个工程师	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。在任一情况下，开发人员必须了解如何与测试系统硬件交互。在使用仪器特定驱动程序时，工程师必须知道每个驱动程序的详细信息以及如何使用它们。而在学习抽象框架时，工程师只需要理解要执行的高级动作，因为仪器细节留给框架。通常，这些高级动作比各种仪器特定驱动程序更直观、更容易实现。
基本功能测试序列的开发和调试（由有经验的工程师进行）	每个序列 80 小时	每个序列 40 小时	测试序列开发变得更快，因为硬件的细节被保存到序列中的一个固定位置，而不是每个驱动程序调用。测试从UUT的角度与硬件交互，可允许序列更直观、更好地匹配测试步骤。一般来说，直观易懂的框架可以将开发和调试时间缩短一半。
由新工程师进行测试开发和调试	每个序列 120 小时	每个序列 60 小时	如果是新的或经验不足的工程师开发测试序列，开发时间的回报将会放大。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存在的步骤来开发序列。此外，直观的框架可让产品测试工程师无需掌握底层软件语言即可开发序列。
由于故障/过时仪器或新仪器需求而需要更新测试序列	驱动程序开发 8 小时 加上每个序列 4 ~ 20 小时	驱动程序开发 8 小时 8 小时 加上每个序列 <1小时	当系统中的仪器需要更换时，测试也必须随之进行更改。对于非抽象平台，这意味着必须为新仪器更新驱动程序调用的每个实例。仪器引用得越多，更新的时间越长。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ATE硬件平台	每个序列 40 ~ 80 小时	每个序列 <8 小时	有时，整个系统都会升级，所有测试都必须迁移到新系统。通常这些新系统会采用完全不同的仪器。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，但是这些驱动程序开发出来之后，必须更新测试序列才能使用它们。如果使用非抽象序列，这一过程非常麻烦，有时还不如重新编写序列来得简单。而抽象序列通常可以在不到一天的时间内更新，所有这一切都通过配置文件来完成，而不必触及测试序列软件。

表4.非抽象和抽象系统中不同任务的相关成本

对于前面假设的场景，您可以使用这些数字来看看是否或何时开发集成的抽象框架。

首先，假设您自己开发所有四个测试序列。您必须从开发框架开始。在标准的非抽象情况下，您必须开发针对特定仪器的驱动程序。第二个场景可使用现成的抽象来开发驱动程序。对于第三个场景，您需要开发集成式HAL/MAL。

任务	开发时间（标准）	开发时间（现成抽象）	开发时间（集成的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 x 4 = 320 小时	80 x 4 = 320 小时	40 x 4 = 160 小时
总计	400 小时	420 小时	660 小时

表5.集成的HAL/MAL需要最多的前期开发工作。

在完成初始开发时，集成的HAL/MAL方法比标准方法多出大约240小时，而现成抽象仅多出20个小时。 然而，初始开发结束并不意味着测试程序的结束。

六个月后，研发部门发现需要进行更多的测量，系统中的32通道多路复用器已经不够用了，需要使用4 x 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而不是多路复用器。如果您使用现成的抽象驱动程序，则不需要进行任何驱动程序开发来处理新矩阵，并且序列中的函数调用不需要更改 - 只有详细信息需要更改。而如果使用集成的HAL/MAL，序列更新将仅需要在通道配置文件中完成。

任务	开发时间（标准）	开发时间（现成抽象）	开发时间（集成的HAL/MAL）
新驱动程序开发	4 小时	0 小时	0 小时
新矩阵需更新2个简单的测试序列	2 x 4 = 8 小时	2 x 2 = 4 小时	2 x 1 = 1 hour
新矩阵需更新2个复杂的测试序列	2 x 16 = 32 小时	2 x 12 = 24 小时	2 x 2 = 2 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.集成的HAL/MAL方法更易于更新，但仍需要更多的开发工作。

即使到现在，集成抽象层的投入还没有开始得到回报，而现成即用的硬件抽象几乎持平了。现在想象一下，一个新的测试程序出现，需要测试四个程序集。但很遗憾，你太忙了，不能自己开发这些序列，招聘了两名新测试工程师。 你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（现成抽象）	开发时间（集成的HAL/MAL）
培训/学习时间	60 x 2 = 120 小时	50 x 2 = 100 小时	40 x 2 = 80 小时
测试开发（4个测试）	120 x 4 = 480 小时	100 x 4 = 400 小时	60 x 4 = 160 小时
额外时间	600 小时	500 小时	240 小时
总计	1,044 小时	948 小时	907 小时

表7.当开发更多测试或需要重大更改时，集成的HAL/MAL方法将得到长远的回报。

此时，最初开发框架的500小时投入已经开始得到回报，比标准开发方法少了100小时。 随着新测试的开发、产品的改变以及产品的生命周期持续，初始投资将持续得到回报。

使用抽象还有许多更主观的回报，无法仅通过一个数字来呈现。 开发测试的时间也大大减少，因为HAL/MAL使得硬件尚未完全定义之前的软件工作更易于展开。 通过维护标准框架，您可以确保一个统一的存储库，并添加新的驱动程序和测量、管理错误以及减少工程师之间的代码分歧。 标准化有助于使每个人（测试工程师、制造工程师和技术人员）保持一致，从而更好地支持系统。 虽然本文档中还详细描述了无数其他优势，但还是让您的能力和ROI计算来帮助您了解什么级别的抽象适合您。

下一步

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，可支持多种测试代码语言、灵活的报表生成和并行/多线程测试。 尽管TestStand包含了许多现成即用的功能，但是它也具有高度可扩展性。 因此，全球数以万计的用户均选择TestStand来构建和部署自动化测试系统。 NI提供了培训和认证计划，每年培养和认证了一千多名TestStand用户。

了解有关TestStand的更多信息

关于Bloomy

Bloomy致力于电子功能测试；航空电子设备、电池和BS硬件在环(HIL)测试；航天系统集成实验室(SIL)数据系统提供产品和服务，同时提供一流的labVIEW、TestStand和VeriStand应用程序开发服务。 Bloomy是NI 24年的联盟合作伙伴，自联盟合作伙伴成立以来就加入，是NI最重要的白金级和精选级合作伙伴。

了解更多关于Bloomy UTS软件套件（套件包含了集成的HAL/MAL)