ANALOG DEVICES 超越一切可能™ | ADI AnalogDialogue

# 了解和使用No-OS及平台驱动程序

Mahesh Phalke，高级软件工程师

快速发展的技术需要软件支持（固件驱动程序和代码示例）来简化设计导入过程。本文介绍如何利用no-OS（无操作系统）驱动程序和平台驱动程序来构建ADI公司精密模数转换器和数模转换器的应用固件，这些器件在速度、功耗、尺寸和分辨率方面提供高水平的性能。

ADI公司提供基于no-OS驱动程序的嵌入式固件示例来支持精密变换器。No-OS驱动程序负责器件配置、转换器数据采集、执行校准等，而基于no-OS驱动程序的固件示例则便于将数据传输到主机进行显示、存储和进一步处理。

## No-OS和平台驱动程序简介

顾名思义，no-OS驱动程序设计用于通用（或无特定）操作系统。该名称还意味着这些驱动程序可以用在没有任何OS支持的裸机(BareMetal)系统上。No-OS驱动程序旨在为给定精密转换器的数字接口访问提供高级API。No-OS驱动程序使用器件的这些API接口访问、配置、读取、写入数据，而无需知道寄存器地址（存储器映射）及其内容。

No-OS驱动程序利用平台驱动程序层来支持跨多个硬件/软件平台复用相同的no-OS驱动程序，使固件高度可移植。平台驱动程序层的使用将no-OS驱动程序隔绝开来，后者无需知道平台特定接口（如SPI、I²C、GPIO等）的低级细节，因此no-OS驱动程序不需要修改就能跨多个平台复用。

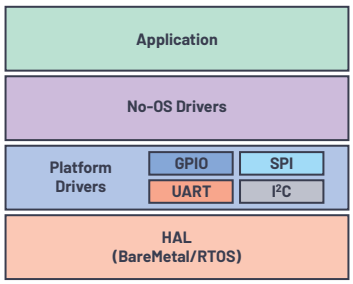


*图1. 精密转换器固件协议栈。*

## 使用No-OS驱动程序

图2显示了no-OS驱动程序的典型代码结构。

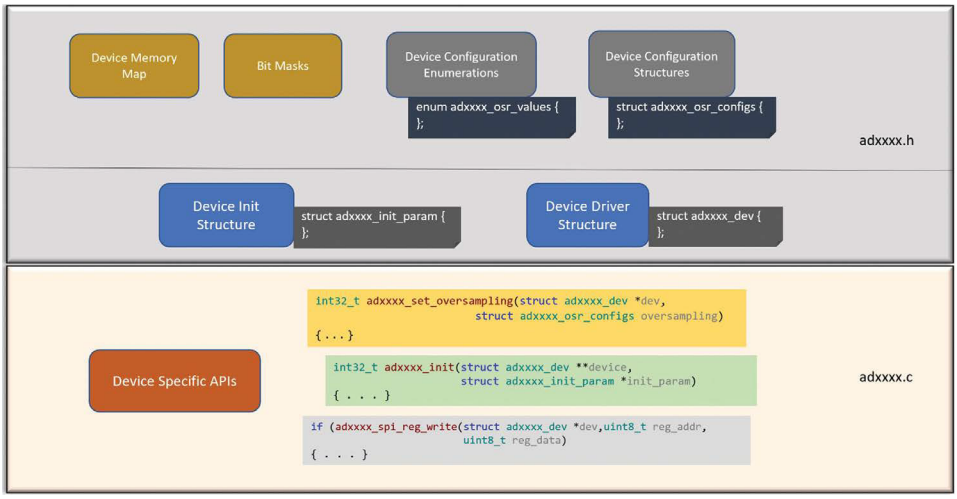


*图2. No-OS驱动程序代码结构。*

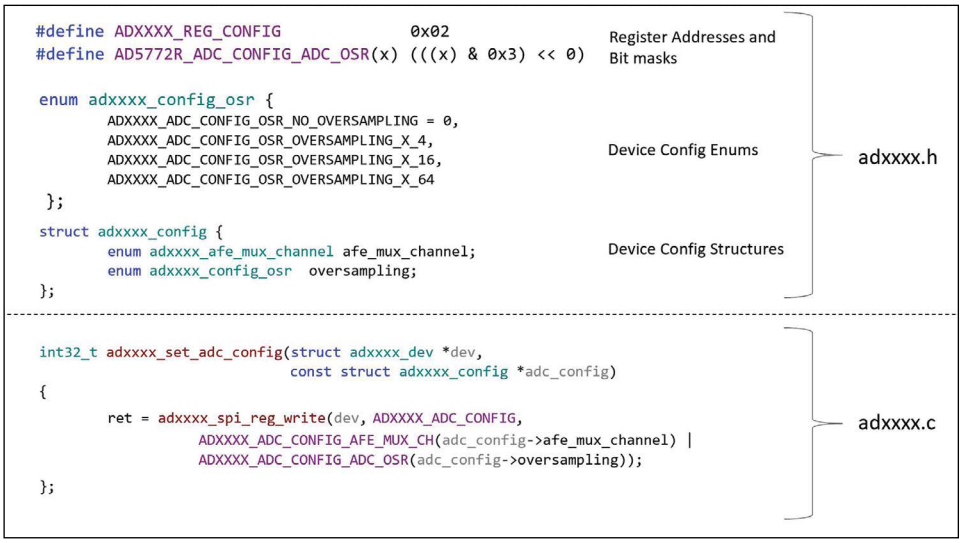


*图3. 器件配置枚举、结构和API。*

精密转换器的no-OS驱动程序代码通常包含在两个以C编程语言编写的源文件中：**adxxxx.c**和**adxxxx.h**，其中xxxx代表器件名称（例如AD7606、AD7124等）。器件头文件(**adxxxx.h**)包含器件特定结构、枚举、寄存器地址和位掩码的公共编程接口，将此文件包含到所需的源文件中便可使用这些公开访问接口。器件源文件(**adxxxx.c**)包含接口的实现，用于初始化和移除器件、读/写器件寄存器、从器件读取数据、获取/设置器件特定参数等。

典型的no-OS驱动程序围绕一组常见功能来构建：

- 器件特定寄存器地址、位掩码宏、器件配置枚举、读/写器件特定参数（如过采样、增益、基准电压等）的结构的声明。
- 通过no-OS驱动程序的器件初始化/移除函数以及器件特定的初始化和驱动程序结构与描述符初始化物理器件/解除器件初始化。
- 使用器件寄存器读/写函数访问器件存储器映射或寄存器详细信息，例如**adxxxx_read_register()**或**adxxxx_write_register()**。

## No-OS驱动程序代码使用

### 使用器件特定地址、位掩码、参数配置枚举和结构：

如前所述，**adxxxx.h**头文件包含所有器件特定枚举和结构的声明，这些枚举和结构被传递到器件特定的函数或API以配置或访问器件参数。具体情况如图3所示。

图3中显示的**adxxxx_config**结构允许用户选择多路复用器通道并为其设置过采样率。此结构的成员（**afe_mux_channel**和**oversampling**）是存在于同一头文件中的枚举，其包含这两个字段的所有可能值的数字常量，用户可以选择。

**adxxxx.c**文件中定义的**adxxxx_set_adc_config()**函数通过配置结构获取用户传递的配置/参数，并进一步调用**adxxxx_spi_reg_write()**函数，通过数字接口（在上例中是SPI）将数据写入**ADXXXX_REG_CONFIG**器件寄存器。

### 使用no-OS驱动程序结构和初始化函数初始化器件：

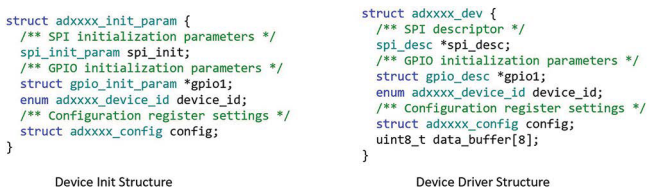


*图4. 器件初始化和驱动程序结构的声明。*

除了器件配置枚举和结构之外，no-OS驱动程序还提供以下两个结构：

- 器件初始化结构。
- 设备驱动程序结构。

器件初始化结构允许用户在用户应用程序代码中定义器件特定的参数和配置。初始化结构包含其他器件特定的参数结构和枚举的成员。图5显示了器件初始化结构的定义。

器件驱动程序结构通过器件初始化函数**adxxxx_init()**加载器件初始化参数。器件驱动程序结构是在运行时（动态）从堆空间中分配内存。器件驱动程序结构和器件初始化结构中声明的参数几乎完全相同。器件驱动程序结构是器件初始化结构的运行时版本。

以下步骤说明典型的器件初始化函数和初始化流程：

- 第1步：在应用程序中创建器件初始化结构的定义（或实例）（例如**struct adxxxx_init_params**），以初始化用户特定的器件参数和平台相关的驱动程序参数。参数在编译期间定义。

  注意：初始化结构中定义的参数因器件而异。

```
struct adxxxx_init_param adxxxx_init_params = { … }
```

- 第2步：在应用程序代码中创建器件驱动程序结构的指针实例（变量）。

  用户应用程序需要创建器件驱动程序结构的单个指针实例。将此实例传递给所有no-OS驱动程序API/函数以访问器件特定参数。应用程序代码中定义的此指针实例指向堆中动态分配的内存，这是通过no-OS驱动程序中定义的器件初始化函数（如**adxxxx_init()**）完成的。

```
static struct adxxx_dev *p_adxxxx_dev = NULL;
```

- 第3步：调用器件初始化函数以初始化器件和其他平台特定的外设。

```
/* Initialize ADxxxx device and peripheral interface */
init_status = adxxxx_init(&p_adxxxx_dev, &adxxxx_init_str);
if (init_status != SUCCESS) {
        return init_status;
}
```

  No-OS驱动程序中定义的**adxxxx_init()**函数用**adxxx_init_param**结构传递的用户特定参数初始化器件。器件驱动程序结构的指针实例和器件初始化结构的实例作为两个参数传递给此初始化函数。用户应用程序代码可以多次调用**adxxxx_init()**函数，只要调用初始化函数之后再调用器件移除函数来平衡。

## 通过器件寄存器读/写函数访问存储器映射（寄存器内容）如图6所示

用户可以通过no-OS驱动程序器件特定的**adxxx_read/write()**函数访问器件寄存器内容（例如产品ID、暂存区值、OSR等）。

大多数情况下，用户不会直接使用寄存器访问函数。器件特定的函数通过这些寄存器访问函数（如**adxxxx_spi_reg_read/write()**）来调用。如果可能，建议使用器件配置和状态API来访问器件存储器映射，而不要使用直接寄存器访问函数，因为这样能确保器件驱动程序结构与器件中的配置保持同步。

## 平台驱动程序

平台驱动程序是包装平台特定API的硬件抽象层(HAL)之一。它们由no-OS器件驱动程序或用户应用程序代码调用，使后者可以独立

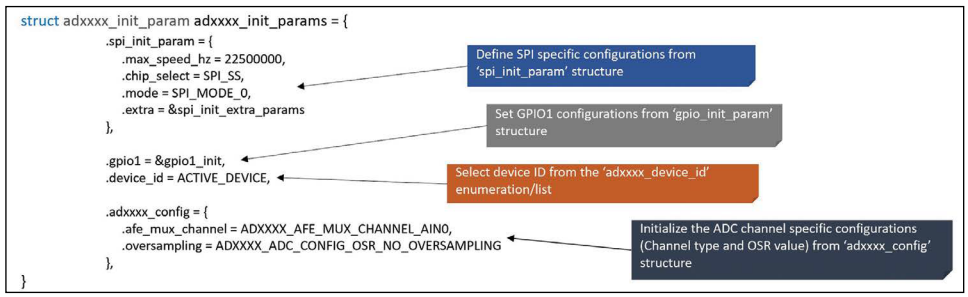


*图5. 用户应用程序中的器件初始化结构定义。*

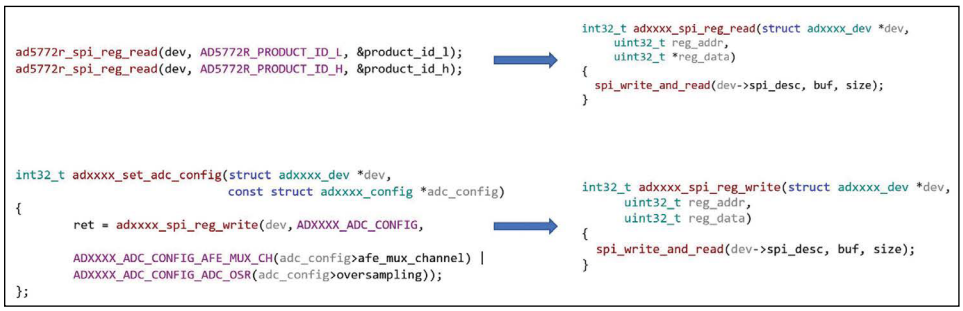


*图6. 访问寄存器内容。*

于底层硬件和软件平台。平台驱动程序包装了平台特定的低级硬件功能，例如SPI/$I^2C$初始化和读/写、GPIO初始化和读/写、UART初始化和接收/发送、用户特定的延迟、中断等。

SPI平台驱动程序模块的典型文件结构如图7所示。

## 使用平台驱动程序

平台驱动程序代码通常包含在以C/C++编程语言编写的三个源文件中。

1) **spi.h**：这是一个与平台无关的文件，包含SPI功能所需的器件结构和枚举。此头文件中定义的C编程接口没有平台依赖性。

   初始化和器件结构中声明的所有参数对任何平台上的SPI接口都是通用的。

   器件初始化结构中使用的**void *extra**参数允许用户传递额外的参数，这些参数可以是所用平台特定的。

   SPI驱动程序结构和SPI初始化结构中声明的参数几乎完全相同。SPI驱动程序结构是SPI初始化结构的运行时版本。

2) **spi.cpp/.c**：此文件包含**spi.h**文件中声明的函数的实现，这些函数用于初始化特定平台的SPI外设以及读/写数据。广义的“平台”是指硬件微控制器（目标器件）和软件（如RTOS或Mbed-OS）的组合。此文件依赖于平台，移植到其他平台时需要修改。

图9详细说明了Mbed平台的SPI接口，并显示了如何使用这些接口和器件初始化/驱动程序结构来初始化SPI和读/写数据。

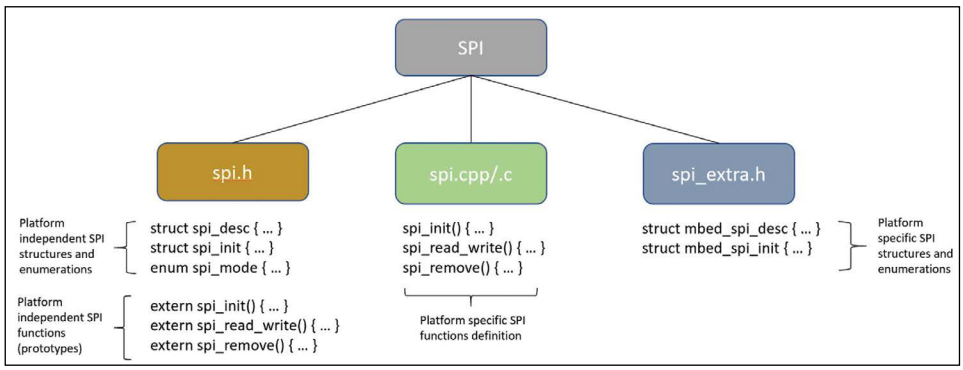


*图7. SPI平台驱动程序代码结构。*

```
typedef struct spi_init_param {
   uint32_t max_speed_hz;
   uint8_t chip_select;
   enum spi_mode mode;
   void *extra;
} spi_init_param;
```

SPI Init Structure

```
typedef struct spi_desc {
   uint32_t max_speed_hz;
   uint8_t chip_select;
   enum spi_mode mode;
   void *extra;
} spi_desc;
```

SPI Driver Structure

*图8. SPI初始化和驱动程序结构。*

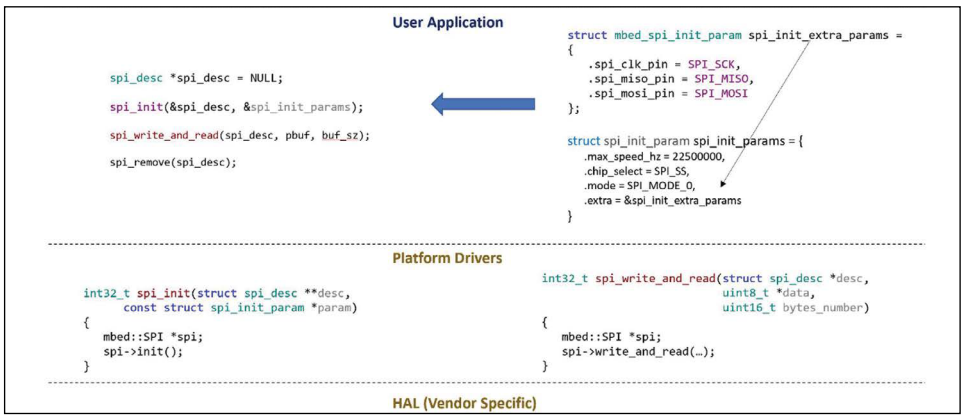


*图9. SPI API或函数注意：增加的spi_init()和spi_write_and_read()代码是节略代码，为清楚起见而省略了细节。*

```
typedef struct mbed_spi_init_param
{
  uint8_t spi_miso_pin;
  uint8_t spi_mosi_pin;
  uint8_t spi_clk_pin;
} mbed_spi_init_param;
```

SPI Extra Init Structure

```
typedef struct mbed_spi_desc {
  void *spi_port;
  void *slave_select;
} mbed_spi_desc;
```

SPI Extra Driver Structure

*图10. SPI额外的初始化和驱动程序结构。*

3) **spi_extra.h**：此文件包含其他器件结构或枚举，其特定于给定平台。它允许用户应用程序代码提供通用**spi.h**文件中未涉及的配置。例如，SPI引脚可能随平台而异，因此可以作为这些平台特定的额外结构的一部分添加。

## 移植平台驱动程序

平台驱动程序可以从一个平台（微控制器）移植到另一个平台；若要移植，通常需要创建平台特定的**.cpp/.c**和**_extra.h**文件。平台驱动程序驻留在微控制器单元供应商提供的器件特定硬件抽象层(HAL)之上的一层。因此，为将平台驱动程序从一个平台移植到另一个平台，与调用供应商提供的HAL中存在的函数或API相关的平台驱动程序代码需要做一些细微改动。

图12区分了基于Mbed的SPI平台驱动程序和ADuCM410 SPI平台驱动程序。

ADI no-OS存储库和平台驱动程序的GitHub源代码链接可在ADI公司Wiki和GitHub页面上找到。

## 为No-OS驱动程序做贡献

ADI no-OS驱动程序已开源并托管在GitHub上。驱动程序不仅支持精密转换器，也支持许多其他ADI产品，如加速度计、收发器、光电器件等。任何熟悉源代码的人都可以为这些驱动程序做贡献，方式是提交变更和创建拉取请求来审核这些变更。

有许多示例项目可以在Linux和/或Windows环境中运行。许多示例项目是用硬件描述性语言(HDL)开发的，以便在Xilinx®、Intel®等公司开发的FPGA以及由不同供应商开发的目标处理器上运行。

如需无操作系统的系统的no-OS软件驱动程序（用C编写），请访问ADI公司no-OS GitHub存储库。

ADI公司Wiki页面提供了使用Mbed和ADuCMxxx平台为精密转换器开发的示例。

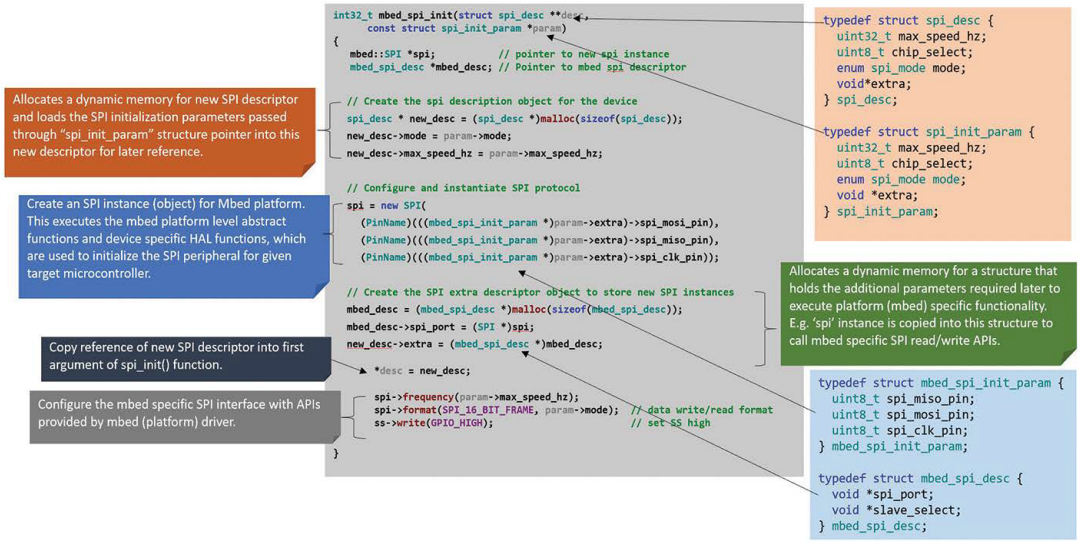


*图11. Mbed平台特定的SPI初始化实现。*

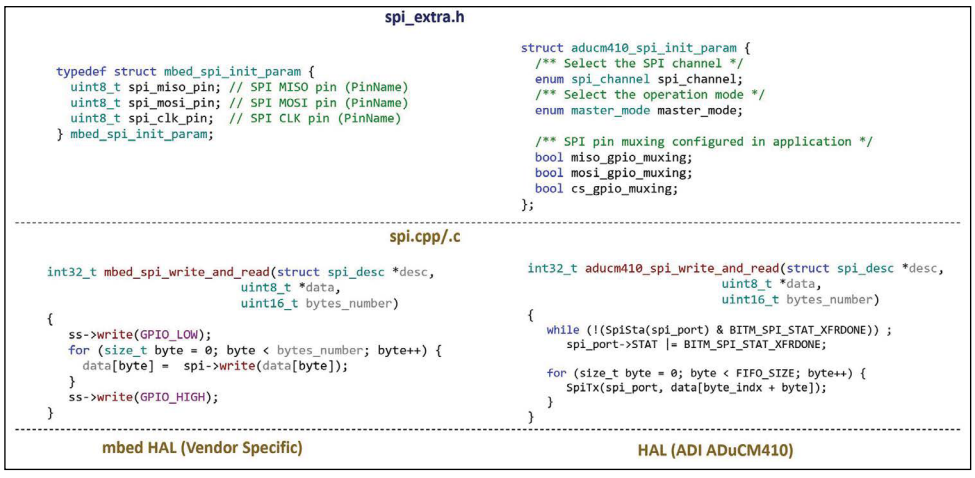


*图12. 平台驱动程序差异。*

## 作者简介

Mahesh Phalke是ADI公司位于印度班加罗尔的精密转换器技术软件部门的高级软件工程师。2011年毕业于瀑内大学，获电子工程学士学位。联系方式：mahesh.phalke@analog.com。