瞭解和使用 no-OS 及平台驅動程式

■作者：Mahesh Phalke / ADI 資深軟體工程師

快速發展的技術需要軟體支援 (硬體驅動程式和代碼示例) 來簡化設計導入過程。本文介紹如何利用 no-OS (無作業系統) 驅動程式和平台驅動程式來建構 ADI 精密類比數位轉換器和數位類比轉換器的應用硬體，這些元件在速度、功耗、尺寸和解析度方面提供高水準的性能。

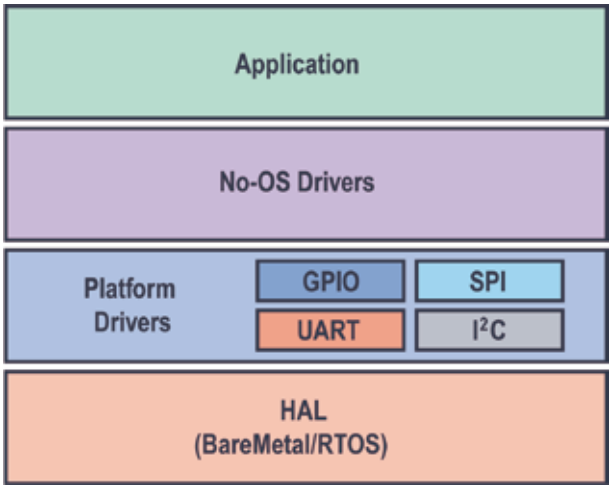
ADI 提供基於 no-OS 驅動程式的嵌入式硬體示例來支援精密變換器。no-OS 驅動程式負責元件配置、轉換器數據擷取、執行校準等，而基於 no-OS 驅動程式的硬體示例則便於將資料傳輸到主機進行顯示、儲存和進一步處理。

no-OS 和平台驅動程式簡介

顧名思義，no-OS 驅動程式設計用於通用 (或無特定) 作業系統。該名稱還表示這些驅動程式可以用在沒有任何 OS 支援的裸機 (BareMetal) 系統上。no-OS 驅動程式目的在為給定精密轉換器的數位介面瀏覽提供進階 API。no-OS 驅動程式使用元件的這些 API 介面瀏覽、配置、讀取、寫入資料，而無需知道暫存器位址 (記憶體映射) 及其內容。

no-OS 驅動程式利用平台驅動程式層來支援跨多個硬體 / 軟體平台複用相同的 no-OS 驅動程式，使硬體高度可移植。平台驅動程式層的使用將 no-OS 驅動程式隔絕開來，後者無需知道平台特定介面 (如 SPI、I²C、GPIO 等) 的低階細節，因此 no-OS 驅動程式不需要修改就能跨多個平台複用。

圖 1：精密轉換器硬體協定棧



使用 no-OS 驅動程式

圖 2 顯示了 no-OS 驅動程式的典型代碼結構。

圖 2：no-OS 驅動程式代碼結構

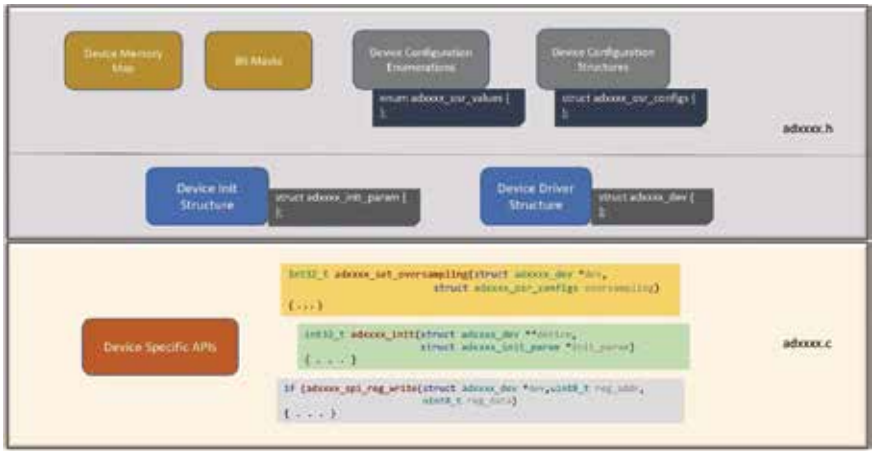
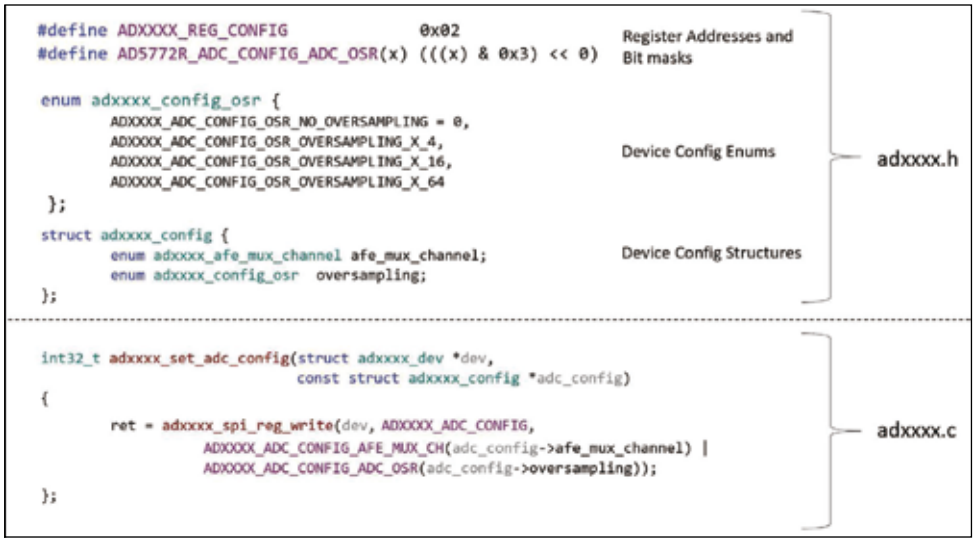


圖 3：元件配置列舉、結構和 API



精密轉換器的 no-OS 驅動程式代碼通常包含在兩個以 C 程式設計語言編寫的原始檔案中：adxxxx.c 和 adxxxx.h，其中 xxxx 代表元件名稱 (例如 AD7606、AD7124 等)。元件標頭檔 (adxxxx.h) 包含元件特定結構、列列舉、暫存器位址和位元遮罩的公共程式設計介面，將此檔包含到所需的原始檔案中便可使用這些公開瀏覽介面。元件原始檔案 (adxxxx.c) 包含介面的實現，用於初始化和移除元件、讀 / 寫元件暫存器、從元件讀取數據、擷取 / 設定元件特定參數等。

典型的 no-OS 驅動程式圍繞一組常見功能來建構：

■元件特定暫存器位址、位元遮罩巨集、元件配置列舉、讀 / 寫元件特定參數 (如過採樣、增益、基準電壓等) 的結構的聲明。

■透過 no-OS 驅動程式的元件初始化 / 移除函數以及元件特定的初始化和驅動程式結構與描述符初始化物理元件 / 解除元件初始化。

■使用元件暫存器讀 / 寫函數瀏覽元件記憶體映射或暫存器詳細資訊，例如 adxxxx_read_register() 或 adxxxx_write_register()。

no-OS 驅動程式代碼使用

使用元件特定位址、位元遮罩、參數配置列舉和結構：

如前所述，adxxxx.h 標頭檔包含所有元件特定列舉和結構的聲明，這些列舉和結構被傳遞到元件特定的函數或 API 以配置或訪問元件參數。具體情況如圖 3 所示。

圖 3 中顯示的 adxxxx_config 結構允許使用者選擇多工器通道並為其設定過取樣速率。此結構的成員 (afe_mux_channel 和 oversampling) 是存在於同一頭檔中的列舉，其包含這兩個欄位的所有可能值的數位常量，使用者可以選擇。

adxxxx.c 檔中定義的 adxxxx_set_adc_config() 函數透過配置結構獲取使用者傳遞的配置 / 參數，並進一步調用 adxxxx_spi_reg_write() 函數，透過數位介面 (在上例中是 SPI) 將資料寫入 ADXXXX_REG_CONFIG 元件暫存器。

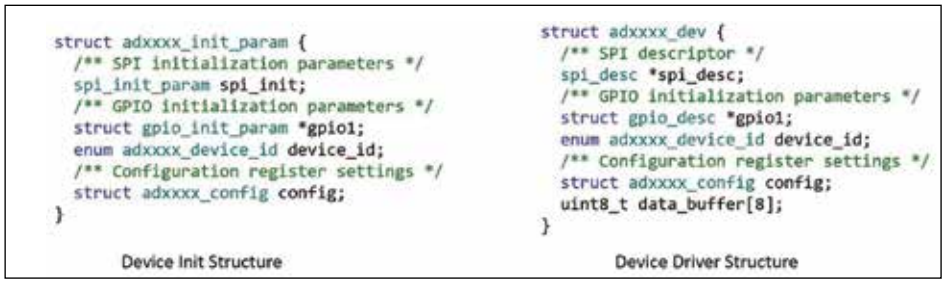
使用 no-OS 驅動程式結構和初始化函數初始化元件：

除了元件配置列舉和結構之外，no-OS 驅動程式還提供以下兩個結構：

■元件初始化結構。

■設備驅動程式結構。

圖 4：元件初始化和驅動程式結構的聲明



元件初始化結構允許使用者在使用者應用程式碼中定義元件特定的參數和配置。初始化結構包含其他元件特定的參數結構和列舉的成員。圖 5 顯示了元件初始化結構的定義。

元件驅動程式結構透過元件初始化函數 adxxxx_init() 載入元件初始化參數。元件驅動程式結構是在運行時 (動態) 從堆空間中分配記憶體。元件驅動程式結構和元件初始化結構中聲明的參數幾乎完全相同。元件驅動程式結構是元件初始化結構的執行階段版本。

以下步驟說明典型的元件初始化函數和初始化流程：

■第 1 步：在應用程式中創建元件初始化結構的定義 (或實例) (例如 struct adxxxx_init_params)，以初始化用戶特定的元件參數和平台相關的驅動程式參數。參數在編譯期間定義。

注意：初始化結構中定義的參數因元件而異。

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

■第 2 步：在應用程式碼中創建元件驅動程式結構的指標實例 (變數)。

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

圖 5: 使用者應用程式中的元件初始化結構定義

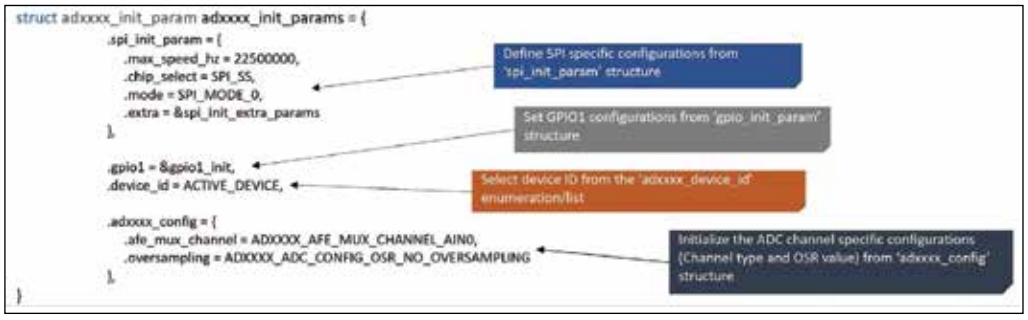
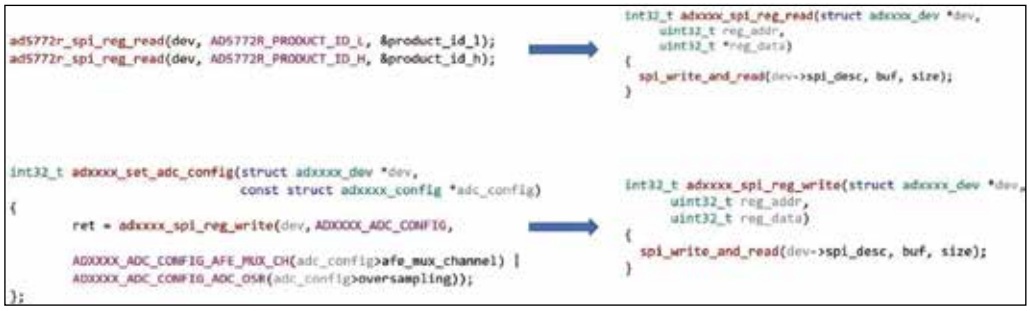


圖 6: 存取暫存器內容



使用者應用程式需要創建元件驅動程式結構的單一指標實例。將此實例傳遞給所有 no-OS 驅動程式 API/ 函數以存取元件特定參數。應用程式碼中定義的此指標實例指向堆中動態分配的記憶體，這是透過 no-OS 驅動程式中定義的元件初始化函數 (如 adxxxx_init()) 完成的。

■第 3 步：調用元件初始化函數以初始化元件和其他平台特定的周邊。

```
/* 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 驅動程式中定義的 adxxxx_init() 函數用 adxxx_init_param 結構傳遞的使用者特定參數初始化元件。元件驅動程式結構的指標實例和元件初始化結構的實例作為兩個參數傳遞給此初始化函數。使用者應用程式碼可以多次調用 adxxxx_init() 函數，只要調用初始化函數之後再調用元件移除函數來平衡。

透過元件暫存器讀 / 寫函數存取記憶體映射 (暫存器內容) 如圖 6 所示。

使用者可以透過 no-OS 驅動程式元件特定的 adxxx_read/write() 函數存取元件暫存器內容 (例如產品 ID、暫存區值、OSR 等)。

大多數情況下，使用者不會直接使用暫存器存取函數。元件特定的函數透過這些暫存器存取函數 (如 adxxxx_spi_reg_read/write()) 來調用。如果可能，建議使用元件配置和狀態 API 來存取元件記憶體映射，而不要使用直接暫存器存取函數，因為這樣能確保元件驅動程式結構與元件中的配置保持同步。

平台驅動程式

平台驅動程式是包裝平台特定 API 的硬體抽象層 (HAL) 之一。它們由 no-OS 元件驅動程式或使用者應用程式碼調用，使後者可以獨立於底層硬體和軟體平台。平台驅動程式包裝了平台特定的低階硬體功能，例如 SPI/I²C 初始化和讀 / 寫、GPIO 初始化和讀 / 寫、UART 初始化和接收 / 發送、用戶特定的延遲、中斷等。

SPI 平台驅動程式模組的典型檔結構如圖 7 所示。

圖 7：SPI 平台驅動程式代碼結構

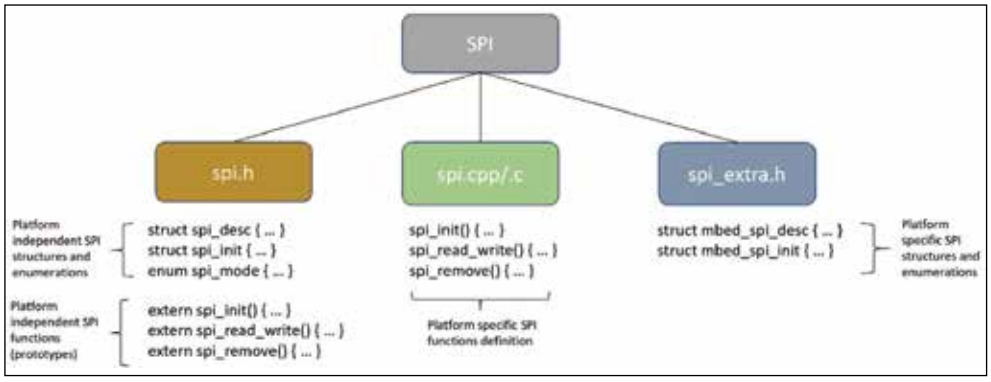


圖 8：SPI 初始化和驅動程式結構

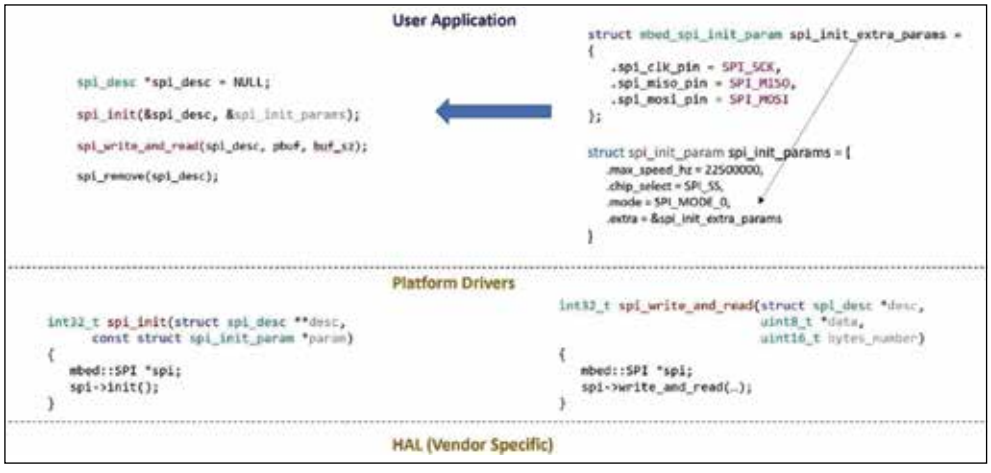
```
typedef struct spi_init_param {
    uint32_t max_speed_hz;
    uint8_t chip_select;
    enum spi_mode mode;
    void *extra;
} spi_init_param;
```

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

圖 9：SPI API 或函數注意：增加的 spi_init() 和 spi_write_and_read() 代碼是節略代碼，為清楚起見而省略了細節。



使用平台驅動程式

平台驅動程式代碼通常包含在以 C/C++ 程式設計語言編寫的三個原始檔案中。

1) spi.h：這是一個與平台無關的檔，包含 SPI 功能所需的元件結構和列舉。此標頭檔中定義的 C 程式設計介面沒有平台依賴性。

初始化和元件結構中聲明的所有參數對任何平台上的 SPI 介面都是通用的。

元件初始化結構中使用的 void *extra 參數允許用戶傳遞額外的參數，這些參數可以是所用平台特定的。

SPI 驅動程式結構和 SPI 初始化結構中聲明的參數幾乎完全相同。SPI 驅動程式結構是 SPI 初始化結構的執行階段版本。

2) spi.cpp/.c：此檔包含 spi.h 檔中聲明的函數的實現，這些函數用於初始化特定平台的 SPI 周邊以及讀 / 寫資料。廣義的「平台」是指硬體微控制器 (目標元件) 和軟體 (如 RTOS 或 Mbed-OS) 的組合。此檔依賴於平台，移植到其他平台時需要修改。

圖 9 詳細說明了 Mbed 平台的 SPI 介面，並顯示了如何使用這些介面和元件初始化 / 驅動程式結構來初始化 SPI 和讀 / 寫資料。

3) spi_extra.h: 此檔包含其他元件結構或列舉，其特定於給定平台。它

圖 10：SPI 額外的初始化和驅動程式結構

```
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
```

允許用戶應用程式碼提供通用 spi.h 檔中未涉及的配置。例如，SPI 接腳可能隨平台而異，因此可以作為這些平台特定的額外結構的一部分增加。

移植平台驅動程式

平台驅動程式可以從一個平台 (微控制器) 移植到另一個平台；若要移植，通常需要創建平台特定的 .cpp/.c 和 _extra.h 文件。平台驅動程式駐留在微控制器單元供應商提供的元件特定硬體抽象層 (HAL) 之上的一層。因此，為將平台驅動程式從一個平台移植到另一個平台，與調用供應商提供的 HAL 中存在的函數或 API 相關的平台驅動程式代碼需要做一些細微改動。

圖 12 區分了基於 Mbed 的 SPI 平台驅動程式和 ADuCM410 SPI 平台驅動程式。

ADI no-OS 儲存庫和平台驅動程式的 GitHub 原始程式碼連結可參考：Analog Devices Wiki 和 GitHub 頁面。

爲 no-OS 驅動程式做出貢獻

ADI no-OS 驅動程式已開源並託管在 GitHub 上。驅動程式不僅支援精密轉換器，也支援許多其他 ADI 產品，如加速度計、收發器、光電元件等。任何熟悉原始程式碼的人都可以為這些驅動程式做貢獻，方式是提交變更和創建拉取請求來審核這些變更。

有許多示例項目可以在 Linux 和 / 或 Windows 環境中運行。許多示例專案是用硬體描述性語言 (HDL) 開發的，以便在 Xilinx、Intel 等公司開發的 FPGA 以及由不同供應商開發的目標處理器上運行。

如需無作業系統的系統的 no-OS 軟體驅動程式 (用 C 編寫)，請瀏覽 ADI no-OS GitHub 儲存庫。

ADI Wiki 頁面提供了使用 Mbed 和 ADuCMxxx 平台為精密轉換器開發的示例。CTA

圖 11：Mbed 平台特定的 SPI 初始化實現

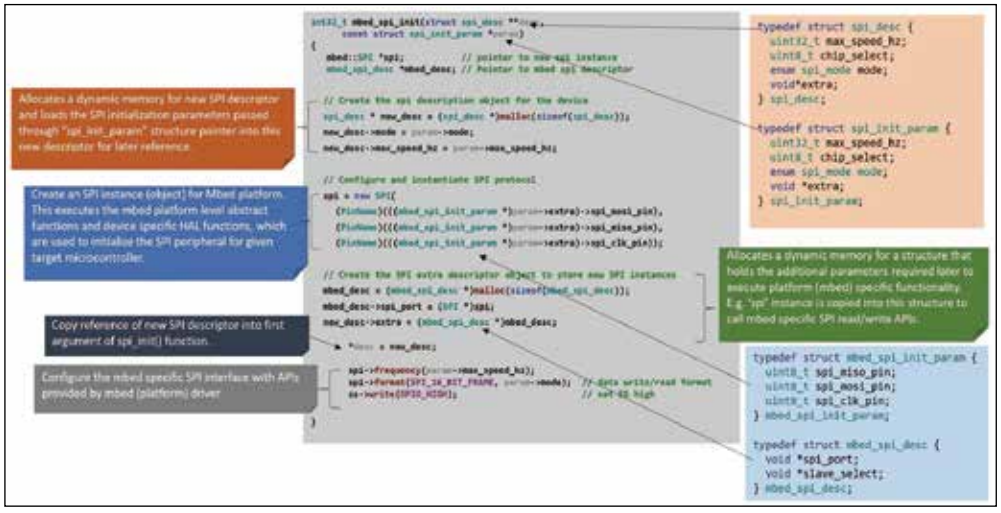


圖 12：平台驅動程式差異