利用與硬體無關的方法簡化嵌入式系統設計：基本知識

本文將展示一種加速嵌入式系統設計原型階段的方法，說明如何將與硬體無關的驅動程式和感測器結合使用，簡化整個嵌入式系統的元件選擇。同時並將介紹嵌入式系統的元件、典型軟體結構以及驅動程式的實現。

■作者：Giacomo Paterniani / ADI 現場應用工程師

簡介

透過使用與硬體無關的驅動程式，設計人員可以自由選擇微控制器或處理器的類型來管理感測器，而不受硬體的限制。此種方法的優勢，除了供應商提供的基本軟體層外，還可以增加額外的軟體層，同時簡化感測器的整合。本文將以慣性測量單元 (IMU) 感測器為例，說明如何實現與硬體無關的驅動程式，不過，此種方法同樣適用於其他類型的感測器和元件。驅動程式採用 C 語言編寫，並在一款通用微控制器上進行了測試。

元件選擇

IMU 感測器主要用於運動檢測，以及透過加速度和角速度來測量運動強度。本示例選擇使用 ADIS16500 IMU 感測器 (圖 1)，因為相較於複雜且昂貴的分立設計方案，該感測器能夠為精準的多軸慣性感測與工業系統的整合提供簡單且經濟高效的方法。

主要應用包括：

■導航、穩定性和儀器儀錶

■無人機和自動駕駛車輛

■智慧農業和施工機械設備

■工廠 / 工業自動化、機器人

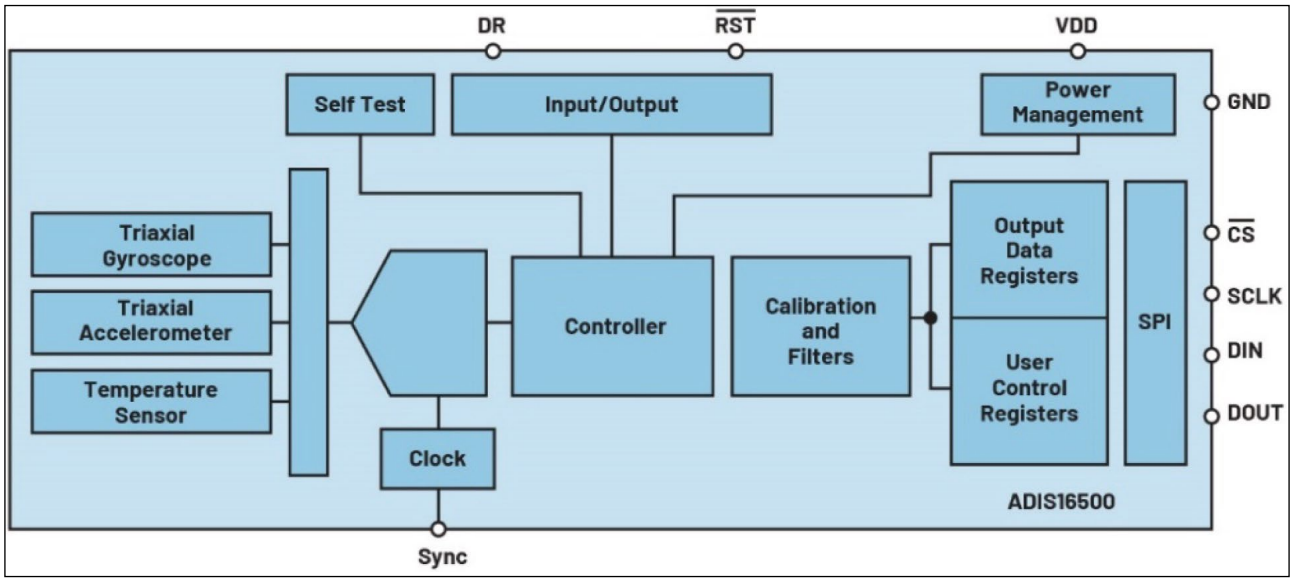
圖 1:ADIS16500 評估板。

■虛擬 / 擴增實境

■運動物聯網

ADIS16500 為一款精密微型機電系統 (MEMS) IMU，內建一個三軸陀螺儀、一個三軸加速度計和一個溫度感測器。參見圖 2。該 IMU 的靈敏度、偏置、對準、線性加速度 (陀螺儀偏置) 和坐標軸原點 (加速度計位置) 已在工廠校準。表示在各種條件下都能提供精準的感測器測量。

圖 2：ADIS16500 框圖。



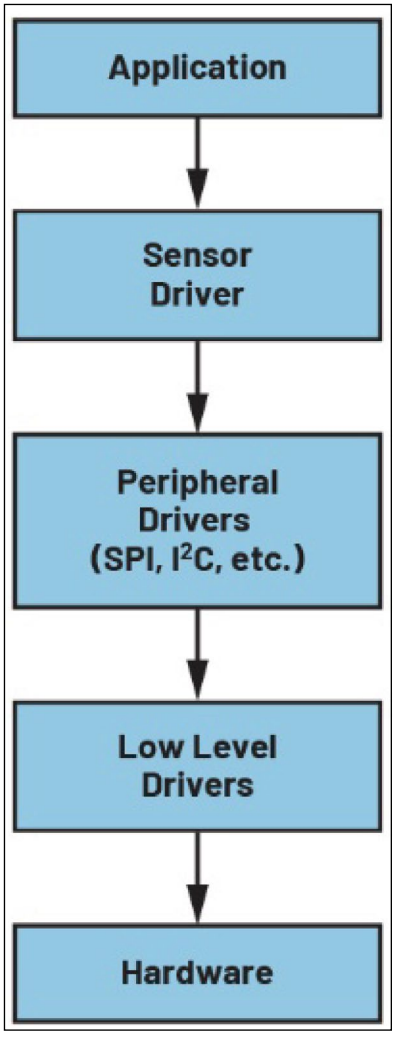
透過該介面，微控制器可以寫入和讀取用戶控制暫存器，並讀取輸出資料暫存器，進而獲得加速度計、陀螺儀或溫度感測器資料。為此，管理該介面所需的全部軟體和韌體均已完成開發。圖 2 所示為數據就緒 (DR) 接腳。該接腳是一個數位訊號，指示何時可從感測器讀取新資料。DR 接腳可被視為通過通用輸入 / 輸出 (GPIO) 埠的輸入，因此可透過微控制器輕鬆管理。

從硬體的角度來看，IMU 感測器和微控制器將使用 SPI 介面連接，該介面是由 nCS、SCLK、D_{IN} 和 D_{OUT} 接腳組成的 4 線介面。DR 接腳應連接到微控制器的其中一個 GPIO。此外，IMU 感測器需要 3 V 至 3.6 V 的電源電壓，因此 3.3 V 就足夠了。

瞭解嵌入式系統的典型軟體結構

瞭解嵌入式系統的通用軟體和韌體結構對於與感測器驅動程式連接十分重要。這將協助設計人員建構一個彈性且易於整合到任何專案的軟體模組。此外，驅動程式必須以模組化的方式實現，以使設計人員能夠依賴於現有函數增加更高階的函數。

圖 3：嵌入式系統的軟體 / 韌體結構。



嵌入式系統的軟體結構如圖 3 所示。在圖 3 中，層次結構從應用層開始，應用代碼就是在這一層編寫的。應用層包括 main 檔、依賴於感測器的應用模組，以及依賴於管理處理器配置的周邊驅動程式的模組。此外，在應用層中，還有與微控制器必須處理的任務相關的所

有模組。例如，透過中斷或輪詢、狀態機等管理任務的所有軟體。應用層級別根據專案的類型而有所不同，因此不同項目中實現的程式碼也不同。在應用層，系統的所有感測器根據其產品手冊進行初始化和配置。感測器驅動程式提供的所有公共函數均可調用。例如，讀取負責輸出資料的暫存器，或者寫入一個暫存器以更改設置 / 校準的程式。

應用層下面是感測器的驅動層，這一層有兩種類型的介面。可從應用層調用的所有函數都在這一層實現。此外，函數的原型插入到驅動程式標頭檔 (.h) 中。因此，透過查看感測器驅動程式的標頭檔，您可以瞭解驅動程式的介面以及可從較高層級調用的函數。較低級別的層將與特定周邊驅動程式連接，這些周邊驅動程式依賴於管理感測器的微控制器。周邊驅動程式包括管理微控制器周邊的所有模組，例如 SPI、I^2C、UART、USB、CAN、SPORT 等，或管理處理器內部模組的模組，例如計時器、記憶體、ADC 等。由於它們與硬體緊密相關，因此稱為低級函數。例如，由於微控制器不同，因此每個 SPI 驅動程式都是不同的。我們以 ADIS16500 為例。介面是 SPI，因此其驅動程式將與微控制器的 SPI 驅動程式封裝在一起。對於不同的感測器和不同的介面也是如此。例如，如果另一個感測器具有 I^2C 介面，那麼同樣地，將在感測器的初始化過程中與微控制器的 I^2C 驅動程式封裝在一起。

感測器驅動程式的下層是周邊驅動程式，各類微控制器的周邊驅動程式各不相同。如圖 3 所示，周邊驅動程式和低階驅動程式是分開的。本質上，周邊驅動程式透過可用的通訊協定提供讀寫函數。由於低階驅動程式將管理訊號的實體層，因此其非常依賴於設計人員所使用的硬體。周邊和低階驅動層往往透過視覺化工具從微控制器的整合式開發環境 (IDE) 產生，具體取決於安裝微控制器的評估板。

驅動程式實現

與硬體無關的方法支援在不同應用、不同微控制器或不同處理器中使用相同的驅動程式。如此方法取決於驅動程式的實現方式。要瞭解驅動程式的實現方式，首先要看介面，或圖 4 中的感測器標頭檔 (adis16500.h)。

標頭檔包含有用的公共巨集。其中包括暫存器的位址、SPI 最大速度、預設輸出資料速率 (ODR)、位元遮罩，以及加速度計、陀螺儀和溫度感測器的輸出靈敏度，這些巨集與用於表示資料的位元數 (16 或 32) 有關。圖 4 顯示了這些巨集，其中僅顯示了幾個暫存器的位址作為示例。

圖 4：ADIS16500 標頭檔 (adis16500.h) 中顯示的巨集。

```
//------------------------------------------------------------//
// ADIS16500 registers
#define ADIS16500_PROD_ID          0x4074

#define ADIS16500_REG_DIAG_STAT    0x02
#define ADIS16500_REG_X_GYRO_L     0x04
#define ADIS16500_REG_X_GYRO_OUT   0x06
#define ADIS16500_REG_Y_GYRO_L     0x08
#define ADIS16500_REG_Y_GYRO_OUT   0x0A
#define ADIS16500_REG_Z_GYRO_L     0x0C
#define ADIS16500_REG_Z_GYRO_OUT   0x0E
#define ADIS16500_REG_X_ACCEL_L    0x10
#define ADIS16500_REG_X_ACCEL_OUT  0x12
#define ADIS16500_REG_Y_ACCEL_L    0x14
#define ADIS16500_REG_Y_ACCEL_OUT  0x16
#define ADIS16500_REG_Z_ACCEL_L    0x18
#define ADIS16500_REG_Z_ACCEL_OUT  0x1A
#define ADIS16500_REG_TEMP_OUT     0x1C
#define ADIS16500_REG_TIME_STAMP   0x1E
// spi max speed in brust mode
#define ADIS16500_BURST_MAX_SPEED  1000000

// default f_odr
#define ADIS16500_DEFAULT_F_ODR    2000 // Sample per second --> [hz]

// masks
#define ADIS16500_MASK_SYNC_MODE   0x000C
#define ADIS16500_MASK_DR_POL      0x0001

// output sensitivities (acceleration and gyroscope)
#define ADIS16500_ACC_SENSITIVITY_32b  5351254.0f   // [LSB/(m/sec^2)]
#define ADIS16500_GYRO_SENSITIVITY_16b 10.0f        // [LSB/(°/sec)]
#define ADIS16500_GYRO_SENSITIVITY_32b 655360.0f    // [LSB/(°/sec)]
#define ADIS16500_TEMP_SCALE_FACT_16b  0.1f         // [°C/LSB]
#define ADIS16500_TS_SCALE_FACT_16b    49.02f       // [usec/LSB]
```

附錄中的圖 3 顯示了包括 adis16500.h 在內的每個模組均可使用的所有公開變數和公共型別宣告，其中定義了新的類型，以便更有效地管理資料。例如，ADIS16500_XL_OUT 類型被定義為包含三個浮點的結構，每個軸 (x、y 和 z) 一個浮點。此外並透過列舉來支援不同的

感測器配置，使設計人員能夠靈活地選擇符合自身需求的配置。最值得關注的是使驅動程式與硬體無關的部分。在公開變數部分的開頭 (附錄中的圖 3)，有三個關鍵的類型定義：指向三個基本函數的指標，或者 SPI 發送和接收函數，以及為產生正確的停轉時間，兩次 SPI 存取之間所需的延遲函數。這些代碼還顯示了可指向的函數的原型。SPI 發送函數將指向待發送值的指標作為輸入，然後返回可供檢查的內容，以確定發送是否成功。SPI 接收函數也是如此，該函數將指向變數的指標作為輸入，這個指標將儲存接收時讀取的值。延遲函數以浮點數作為輸入，表示設計人員想要等待的微秒數，不返回任何內容 (void)。透過如此方式，設計人員可以在應用層 (例如在 main 檔中) 利用這些特定的原型來聲明這三個函數。聲明後，他們可以將這三個函數賦值給 ADIS16500_INIT 私有結構的欄位。附錄中的圖 2 列舉了一個示例，以協助更好理解最後一步。

SPI 發送器、接收器函數和延遲函數在 main 檔中聲明為靜態函數，因此屬於應用層。這些函數依賴於周邊驅動程式函數，因此感測器驅動程式本身與硬體無關。這三個函數被分配給一個變數的欄位，而這些欄位是指向函數的指標。如此一來，設計人員可以封裝感測器和微控制器，而無需修改感測器驅動程式的程式碼。如果設計人員更換微控制器，他們只需將三個靜態函數內的低階函數替換為新微控制器的相應函數，進而調整 main 檔。透過此種方法，驅動程式變得與硬體無關，因為設計人員不需要更改感測器的驅動程式的程式碼。微控制器的 IDE 中通常包含 spiSelect、spiReceive、spiUnselect、chThdSleepMicroseconds 等低階函數。在本例中，所用的微控制器評估板是 SDP-K1，其嵌入了 STM32F469NIH6 Cortex-M4 微控制器。該 IDE 是 ChibiOS，這是一個免費的 Arm 開發環境。

附錄中的圖 4 顯示了應用級別的可調用函數原型。這些原型以及附錄中圖 2 和圖 3 討論的所有其他軟體和韌體都可在感測器驅動程式的標頭檔 (adis16500.h) 中找到。首先，初始化函數 (adis16500_init) 將指向 ADIS16500_INIT 結構的指標作為輸入，並返回狀態碼，以指示初始化是否成功。初始化函數的實現在感測器驅動程式的原始檔案 (adis16500.c) 中完成。附錄中的圖 5 所示為 adis16500_init 函數的程式碼。首先，定義名為 ADIS16500_PRIV 的類型，其中至少包含 ADIS16500_INIT 結構的所有欄位，然後聲明一個屬於該類型的私有變數 _adis16500_priv。在初始化函數中，應用層傳遞的 ADIS16500_INIT 結構的所有欄位將賦值給私有變數 _adis16500_priv 的欄位。表示對感測器驅動程式的任何後續調用都將使用由應用層傳入的 SPI 讀寫函數和處理器延遲函數。這一點很關鍵，正因如此，感測器驅動程式才能與硬體無關。如果設計人員想要更改微控制器，只需更改傳遞給 adis16500_init 函數的函數即可，不需要修改感測器驅動程式程式的程式碼本身。在初始化函數開頭，_adis16500_priv 變數的已初始化欄位設定為 false，因為初始化過程尚未完成。在該函數結束時，該欄位將設定為 true，然後返回。設計人員每次調用另一個公共函數 (附錄中的圖 4) 時，都會執行以下檢查：如果 _adis16500_priv.initialized 為 true，可以繼續；如果為 false，將立即返回 ADIS16500_RET_VALERROR 錯誤。這是為了防止使用者在沒有先初始化感測器驅動程式的情況下調用函數。繼續討論初始化函數，執行以下步驟：

1. 透過讀取 ADIS16500_REG_ PROD_ID 暫存器，檢查預先已知的產品 ID。
2. 將應用層 (main.c) 傳遞的值寫入 ADIS16500_REG_MSC_CTRL 暫存器的相應位元欄位，設定資料就緒 (DR) 接腳極性。
3. 將應用層 (main.c) 傳遞的值寫入 ADIS16500_REG_MSC_CTRL 暫存器的相應位欄位，
4. 設定同步模式。
5. 將應用層 (main.c) 傳遞的值寫入 ADIS16500_REG_DEC_RATE 暫存器，設定抽取率。

初始化函數取決於讀寫暫存器函數 (附錄中的圖 6)。因此，為 _adis16500_priv 變數賦值之後，需要完成上述四個常式。否則，在調用讀取或寫入暫存器函數時，它們不知道該使用哪個 SPI 發送器、接收器和處理器延遲函數。

參考附錄中的圖 4，在初始化函數之後，還可以調用其他公共函數。以下是已實現常式的功能描述，所示為低級別常式。本文的第二部分將詳細介紹驅動程式的其他已實現函數。以下所有函數只能在初始化函數之後調用。為此，將在每個函數開頭仔細檢查，以確定感測器是否已初始化。如果未初始化，程式會立即返回錯誤。

▶ *adis16500_rd_reg_16*

該函數用於讀取 16 位元暫存器。該函數實現可參見附錄中的圖 6。輸入包括 ad，這是一個 uint8_t 變數，表示要讀取的暫存器的位址，以及 *p_reg_val，這是指向 uint16_t 類型變數的指標，表示讀取值將賦值的目標。要透過 SPI 協定讀取暫存器，需要存取兩次 SPI；第一次存取是為了發送位址，第二次是為了讀回被定址暫存器的值。兩次存取之間需要有停轉時間，因此需要延遲函數。在第一次存取過程中，我們發送讀 / 寫位，在本例中為 1 (R = 1，W = 0)，暫存器位址移位 8 位，再補充 8 位 0，因此序列如下：

R/W | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

其中 AD 代表地址，R/W 代表讀 / 寫位。

經過延遲後，函數透過 SPI 讀取值，並將該值傳遞給輸入指針。ADIS16500 的暫存器具有一個包含高位值 (8 個最高有效位) 的高位址和一個包含低位值 (8 個低有效位) 的低位址。為了獲得 16 位的完整值 (低位和高位)，使用低位址作為 ad 已經足夠，因為低位址和高地址是連續的。

▶ *adis16500_wr_reg_16*

該函數用於寫入 16 位元暫存器。該函數實現可參見附錄中的圖 6。輸入包括 ad，這是一個 uint8_t 類型變數，表示要寫入的暫存器的位址，以及 reg_val，這是 uint16_t 類型變數，表示要寫入暫存器的值。對於讀取函數，需要考慮低位址和高位址以及低位元值和高位值。因此，根據產品手冊，要想寫入 ADIS16500 的暫存器，需要在發送時存取兩次 SPI。第一次存取將發送等於 0 的 R/W 位，接著是低暫存器位址，然後是低位值，因此序列如下：

R/W | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |，其中 D 代表資料。

第二次 SPI 發送器存取將發送等於 0 的 R/W 位，接著是高暫存器位址，然後是高位值，因此序列如下：

R/W | AD14 | AD13 | AD12 | AD11 | AD10 | AD9 | AD8 | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |。

寫入和讀取暫存器函數實際上也可以定義為私有，因此從驅動程式軟體模組外部不可見，也不可調用。將它們定義為公共是為了能夠調變。如此一來，設計人員能夠快速存取感測器中的任何暫存器以進行讀取或寫入，進而協助解決問題。

▶ *adis16500_rd_acc*

該函數從輸出資料暫存器讀取 x、y、z 加速度數據，並返回它們的值，單位為 [m/sec2]。該函數實現可參見附錄中的圖 7。輸入是指向 ADIS16500_XL_OUT 結構的指標，其只嵌入三個欄位：以浮點類型表示的 x、y、z 加速度。在這三個軸上，讀取加速度的方式是相同的，唯一的區別在於要讀取的暫存器。每個軸有其各自要讀取的暫存器：x 軸必須在 x 加速度輸出資料暫存器上讀取，y 和 z 軸也在相應暫存器上讀取。加速度值將用 32 位值來表示，因此要讀取的暫存器有兩個。一個讀取高 16 位，一個讀取低 16 位。因此，透過查看代碼可知，將進行兩次暫存器讀取瀏覽，再加上適當的移位和 OR 位元運算，得到整個二進位值並儲存在名為 _temp 的私有 int32_t 變數中。然後，資料將經過二進位轉二進位補數的轉換。轉換後，用二進位補數值除以靈敏度 (單位為 [LSB/(m/sec2)])，如此最終將獲得以 [m/sec2] 為單位的加速度值。此值將記錄到指標的 x、y 或 z 欄位，該指標指向已作為輸入傳遞的結構。

▶ *adis16500_rd_gyro*

陀螺儀讀取函數與加速度讀取函數的實現方法完全相同。毫無疑問，該函數將讀取 x、y、z 陀螺儀資料，單位為 [°/sec]。其實現方法可參見附錄中的圖 8。與加速度函數類似，函數的輸入是指向 ADIS16500_GYRO_OUT 結構的指標，該結構嵌入以浮點類型表示的 x、y 和 z 陀螺儀資料。讀取的暫存器是陀螺儀輸出資料暫存器。二進位值將用 32 位元表示，要獲得二進位補數值，需要完成與加速度函數相同的步驟。完成二進位到二進位補數轉換後，用得到的值除以靈敏度 (單位為 [LSB/(°/sec)])，最終得到以 [°/sec] 為單位的值，然後該值將記錄到指標的 x、y 或 z 欄位，該指標指向已作為輸入傳遞的結構。

結論

本文闡述了嵌入式系統的典型軟體 / 韌體堆疊，並介紹了 IMU 感測器的驅動程式實現。與硬體無關的方法為各種感測器或元件提供了可重複使用的方法，即使介面 (SPI、I_2C、UART 等) 不同也不會影響。 CTA