用於電源系統管理的 Linduino（上）

大多數電源系統管理設計都遵循一種“設定後便不需再過問”的模型。電源系統管理 (PSM) 件的設定和調試利用 LTpowerPlay 是簡單易行的，而且當與一個量化程式設計解決方案組合時則無需韌體。不過，許多大型系統需要一個電路板管理控制器 (BMC)，因而提出了這樣的問題：“韌體能夠為 PSM 做什麼呢？”

■作者：Michael Jones/ 凌力爾特應用工程經理

PSM 韌體的基礎是 PMBus；PMBus 的基礎是 SMBus；而 SMBus 的基礎則是 I^2C。構建一個利用 PSM 韌體增加價值的 BMC 需要對每種協定有一定程度的瞭解，或者一個預先存在的軟體庫以使程式設計人員擺脫細節的束縛。

Linduino 庫負責處理每個協議層，並提供一個應用程式介面 (API)，從而使得 PSM 韌體的編寫十分容易。Linduino PSM 並不是 BMC 的一種替代品，而是一組可相容典型 BMC 韌體的軟體庫和示例。

另外，Linduino 還可作為一款學習工具與 LTC 展示電路一起使用。許多 BMC 設計已經具有一個 SMBus API，所有這些需要的是快速學習 PMBus 的工作原理。工程師們把 Linduino 代碼片段複製 / 粘貼到現有的應用程式中並加以使用的現象是相當常見的。不過也可以實施 Linduino 層之一，然後重用整個軟體庫，包括：

■元件和電源軌發現

■命令 API

■故障記錄解碼

■系統內程式設計

本應用指南將陳述 Linduino 庫、電源系統管理程式設計、具展示電路的 Linduino PSM 的設置和使用、以及 PSM 調試方法。如需瞭解有關協議和一般程式設計問題的詳細資訊，請查閱“應用指南 135”(Applicaton Note 135 - Implementing Robust PMBus System Software for the LTC3880) 以及針對 I^2C/SMBus/PMBus 的業界標準。

LINDUINO PSM 硬體

Linduino PSM 硬體包括一個 Linduino (DC2026) 和一個遮罩 (DC2294)，以把 Linduino 的 I^2C 引腳連接到一塊展示板或產品板的 PMBus/SMBus /I^2CBus。

為獲得最佳的學習效果，可從一個 DC2026 (Linduino)、DC2294(遮罩)、DC1962(Power Stick) 和一個 Total Phase Beagle(I^2C 嗅探器) 著手。

圖 1：評估硬體

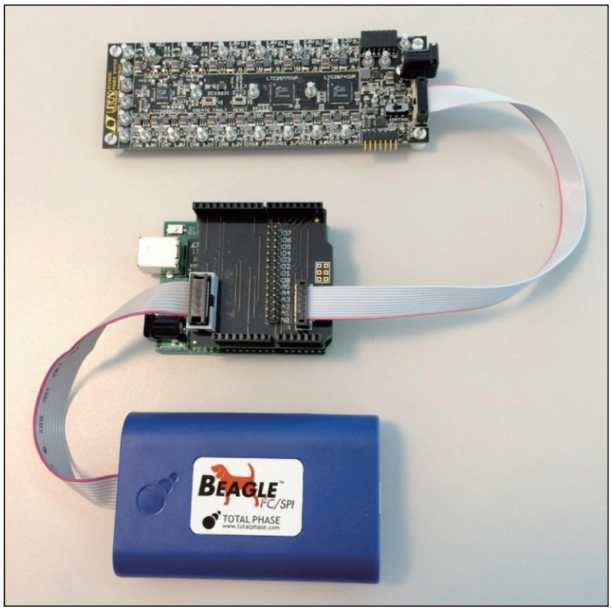


圖 2：系統硬體

這能提供控制器 (LTC388X) 和管理器 (LTC297X) 的程式設計、調試和學習。

圖 1(評估硬體) 所示為全部連接在一起的建議評估硬體。如欲使用該硬體，則利用兩根 USB 電纜把 Linduino 和 Beagle 連接至一台電腦。假如您沒有用 USB 電纜連線 Beagle，那麼就把 Beagle 帶狀電纜從 DC2294 斷接，以避免干擾往來於它和 DC1962 之間的 PMBus 通訊。

如果連接至一個系統板，則 DC2086 可在大多數情況下工作。

DC2086 將接受一個從 DC2294 引出的連接，並支援 12 針帶狀電纜、14 針帶狀電纜和 4 針電纜。另外，DC2086 還支援　一個外部電源輸入，該輸入用於功率需求量高於 Linduino 所能提供之水準的系統板。

LINDUINO PSM SKETCH

在瞭解 PMBus 庫的工作原理之前，快速瀏覽 DC1962 Sketch 將會釐清 Linduino PSM 的一般使用模型。此外，它還將展示編寫代碼程式有多麼簡單易行，即使對於非程式設計人員也不例外。

為進一步跟上，需要進行兩項下載：Arduino 工具和 Linduino Sketchbook。Arduino 工具可從 www.arduino.cc 網站下載，而 Linduino Sketchbook 則可透過 www.linear.com.cn/linduino 下載。

Arduino 工具可在多種平台上的運行。本應用指南的構建和行文採用的是在 64 位 Ubuntu 14 TLS 上運行的 Arduino 1.6.4。

我們開始吧：

第一步：配置

當首次運行 Arduino 軟體時，它將使用一個 default Sketchbook，而不是從 LTC 網站下載的 Linduino Sketchbook。

如欲變更 Linduino Sketchbook，則使用功能表列上的 File | Preferences 選擇，如圖 3(查找 Preferences 對話方塊) 所示。

圖 3：查找 Preferences 對話方塊

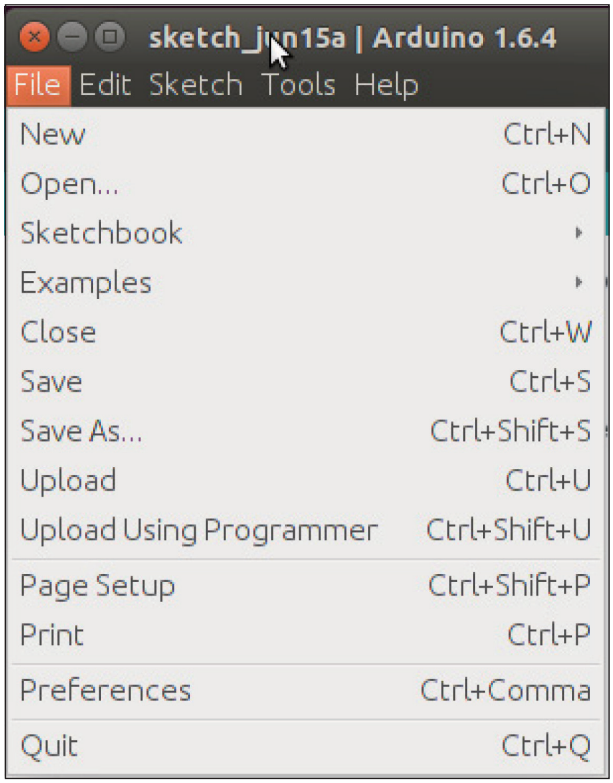
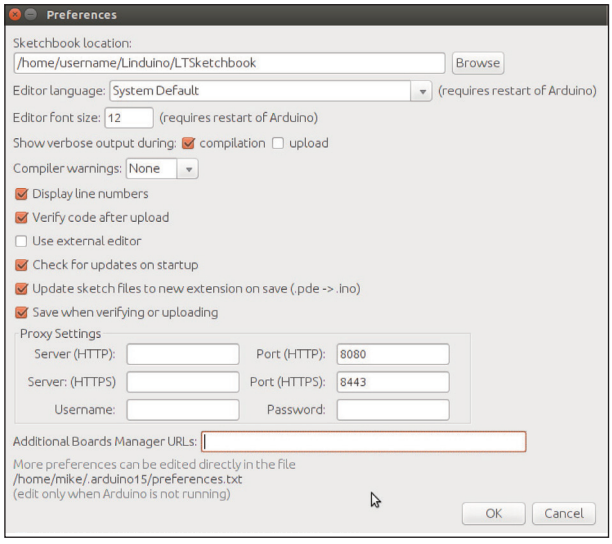


圖 4(Preferences 對話方塊) 表明：Sketchbook Location 位於對話方塊的頂部。使用 Browse 按鈕，導航至從 www.linear.com.cn/linduino 下載的 LTSketchbook。另外，在編譯期間檢查 ON Display 行號和 Show 詳細輸出也是有幫助的。後一個設置在命令列上滾動編譯消息，它們在這裡更容易看到。

在設定路徑之後，必須關閉所有的 Arduino 視窗，而且必須重啓 Arduino 軟體。當重啓 Arduino

圖 4：Preferences 對話方塊



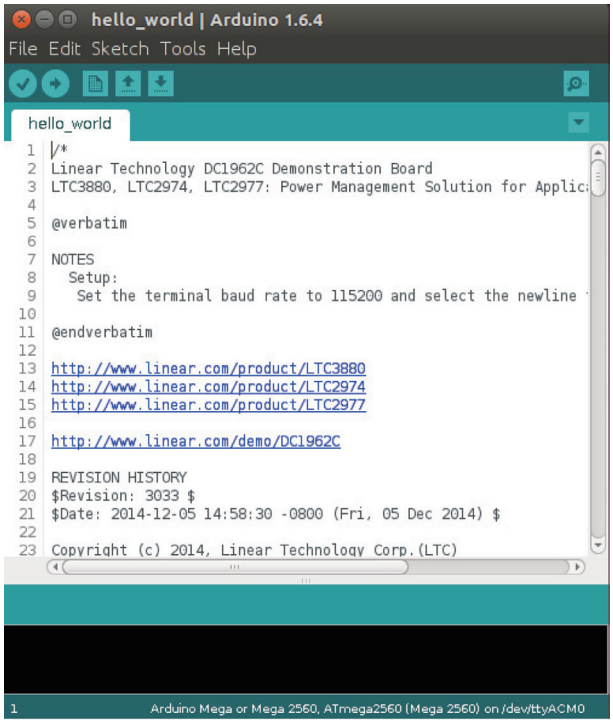
時，它會重新掃描 Sketchbook 目錄並逐步建立 Arduino 功能表。如果未重啓 Arduino 軟體，則該功能表將不反映 LTSketchbook，而是指向前一個 Sketchbook。

第二步：載入您的第一個 Sketch：

透過仿效圖 5(載入 hello_world) 來載入 hello_world Sketch。

在載入了 Sketch 之後，彈出一個具有該 Sketch 的視窗，如圖 6 (Sketch 窗口) 所示。

圖 6：Sketch 窗口



第三步：編譯和運行

如圖 7 (Arduino 工具列) 所示，透過工具列上的鉤形符號來編譯 Sketch。

圖 5：載入 hello_world

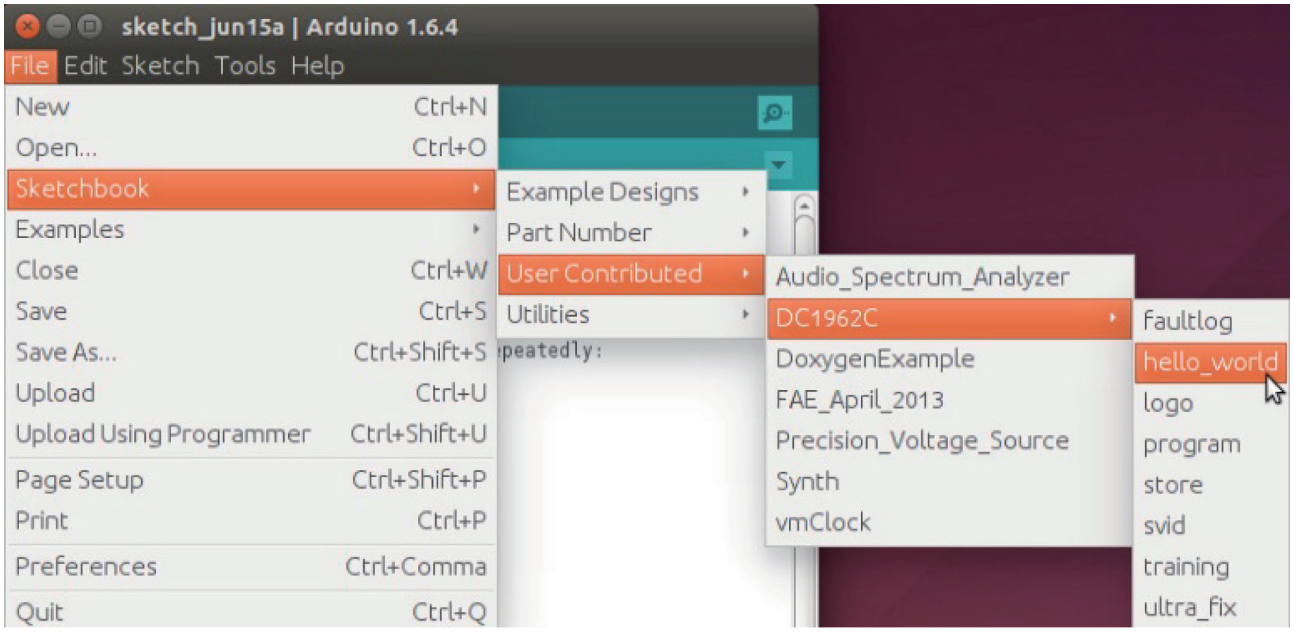


圖 7：Arduino 工具列

指向右的箭頭將編譯 Sketch 並把經過編譯的 Sketch 裝入 Linduino 硬體。螢幕放大鏡顯示 Sketch 的輸出。把箭頭看作發送代碼至顯示控制台，或編譯代碼並把代碼發送至 Linduino 硬體，這樣顯示控制台就具有一些需要對話的東西。

注：Arduino 板類型應設定為 Arduino Uno，而且應選擇埠。見工具功能表 (Tools Menu)。

在載入了 Sketch 之後，按位於工具列右邊的螢幕放大鏡以打開控制台視窗。把行尾結束符號設定為 Carriage return，並將串列傳輸速率設定為 115200，以與圖 8(Arduino 命令窗口) 相匹配。

如需與 Sketch 互動，則把游標置於頂部 (Send 按鈕的左邊) 的方框中，從功能表鍵入一個數位，

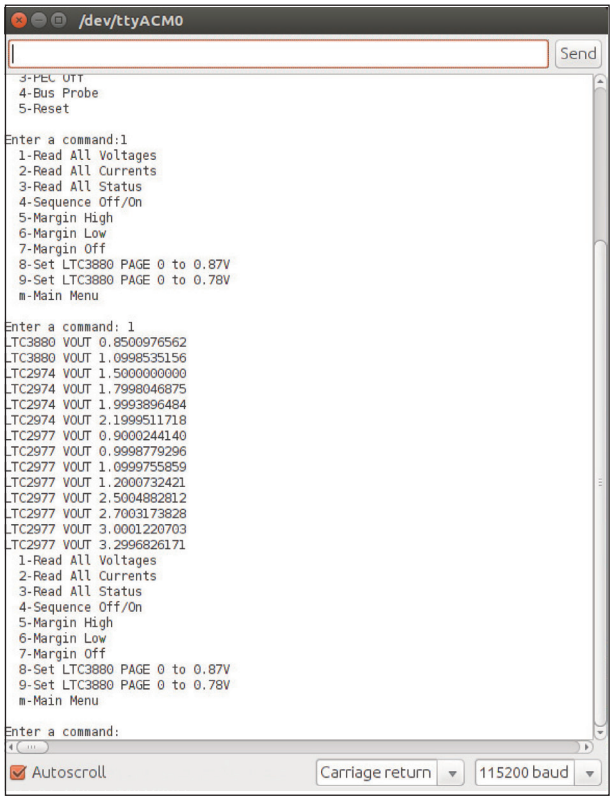
圖 8：Arduino 命令窗口

然後按 Send 鈕或 <CR>。Sketch 接著將執行命令並隨後重新顯示功能表。

第四步：考察功能表項目

圖 9(Sketch 選單) 顯示當按 1 以變更基本命令 (Basic Commands) 視窗、之後按 1 以讀取所有電壓 (Read All Voltages) 時會發生什麼。DC1962 之所有電源軌的 VOUT 測量值被讀出並列印。

圖 9：Sketch 選單



在透過按壓位於左上角的 X 將控制台關閉之前，Sketch 一直處於運行狀態。如果重新打開控制台，則其將重啓 Sketch。

現在您可以考察其他的功能表選項，假如您熟悉 Beagle，則可排佈一些佈線並檢查匯流排事務。

第五步：代碼的修改

Sketch 具有兩個入口點。有一個 setup() 函數 (其被調用一次)，和一個 loop() 函數 (其在一個迴圈中被永久地調用)。這些是 Arduino 編碼環境的組

成部分。如果您是一位有經驗的 C 語言程式設計人員，則很有可能感到疑惑：main() 在哪裡 ?Arduino 庫具有一個調用 setup() 的預定義 main() 和一個無限迴圈調用 loop()。

功能表被編碼為 Sketch 內部的說明函數，而且 loop() 調用主功能表。每個功能表利用一個情況語句提供支援，這裡每種情況處理一個功能表號。

現在，對於程式設計人員的內容已經說明足夠。修改應用程式就是簡單地使用提供的 API 來變更 Sketch 中的情況語句。Sketch 中發出 PMBus 命令的函數 (API) 來自一個單獨的庫，並且具有聽起來像您希望代碼程式執行之任務的簡單名稱。例如：

```
voltage = pmbus->readVout (0x30, false);
```

意味著：使用 pmbus API，在位址 0x30 讀取輸出電壓 (無輪詢)，並將它轉換成一個可變名稱電壓。

現在您應該做出少許變更了。例如，增添一個功能表項目以讀取和列印輸出功率。如果您還沒有準備好，就繼續閱讀以瞭解有關編寫代碼程式的更多內容。假如您已準備就緒，那麼這裡是一個提示：

```
float readPout(uint8_t address, bool polling);
```

在 LTC3880 上試用它 (在 Power Stick 上的地址 0x30)。為證明它是有用的，可把電阻或一個電流負載添加至 Power Stick 上的通道 0，並驗證其與 Sketch 列印的內容相匹配。

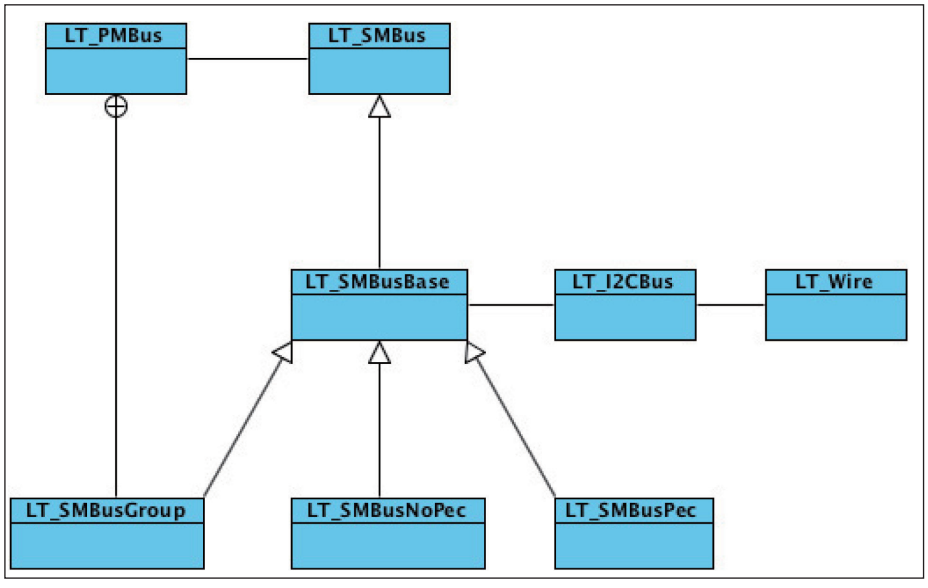
LINDUINO PSM PMBus 庫

PMBus 庫存在於 LTSketchbook/libraries/LT_PMBUS 目錄中的 LTSketchbook 樹之中。該軟體庫是分層的：從 TWI(兩線式介面) 開始，然後是 I^2C、SMBus，最後是 PMBus。有一個用於在 L11/L16 (PMBus 格式) 和浮點之間來回轉換數值的數位轉換 API。最後，具有群組命令協議 (Group Command Protocol) 輔助、元件和電源軌發現、故障記錄解碼、乃至系統內程式設計功能。

每層是一個簡單的 C++ 類，類似於 Arduino 把一個類別用於串列 (Serial) 和其他 I/O 函數的方式。如果您的最終環境是 C 語言，請不要擔心。簡單的意思就是您可以使用 C++ 類 (沒有大量的記憶體開銷)，或者也可以去除類包裝並非常容易地將其用作純 C 語言。C++ 包裝器就是簡化了應用程式碼並使其對於非程式設計人員而言更加容易。

對於那些僅必須瞭解內部結構的程式設計人員來說，SMBus 位於一個層次體系中，這樣一來應用代碼就與接通和關斷 PEC 無關，並可幫助移植。LT_I2C.h、LT_SMBus.h 和 LT_PMBus.h 形成了 API 的層。如欲移植 Linduino PSM 庫，則可選擇任何 API 之一並在您的平台上實施 (使用您自己的軟體庫)。最常用的埠重新實施 LT_SMBusBase 類，然後 PMBus 類正常工作，數學轉換正常運行，而且所有其他的函數和示例正常運作。

圖 10：LT_PMBus 類示意圖



使用 PMBus 庫

這個庫的使用可以無需瞭解所有這些類成員；僅僅需要幾個導入和靜態變數，Sketch 就做好了動作準備。

圖 11：包括庫 (Include Library)

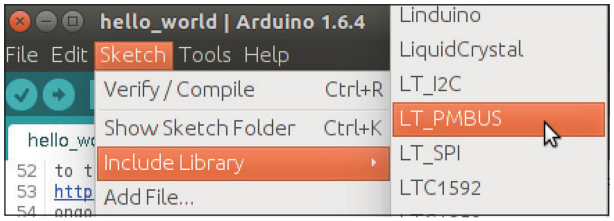


圖 12：基礎包括 (Base Includes)

```
#INCLUDE <LT_SMBUSPEC.H>
#INCLUDE <LT_SMBUSNOPEC.H>
#INCLUDE <LT_SMBUS.H>
#INCLUDE <LT_SMBUSMATH.H>
```

一般來說 PMBus 庫是利用 Sketch 功能表添加的，如圖 11(包括庫 [Include Library]) 中所示。但是最重要包括的則示於圖 12(基礎包括 [Base Includes])。

一個 Sketch 具有至少兩個靜態變數，一個用於系統管理匯流排 (smbus)，另一個用於電源管理匯流排 (pmbus)，如圖 13(靜態變數) 所示。Smbus 變數是 Pec 或 NoPec 版本。乾淨層 (clean layers) 的一個優良的特點是程式設計人員可根據其專案的需要把應用代碼編寫為 SMBus 或 PMBus 代碼。利用 PMBus API 來編寫應用隱藏了命令代碼和資料格式化的細節，而利用 SMBus API 編寫應用則可實現對所有可能命令代碼的訪問以及對原始值的直接訪問。

圖 13：靜態變數

```
STATIC LT_SMBUS *SMBUS = NEW LT_SMBUSPEC();
STATIC LT_SMBUS *SMBUS = NEW LT_PBUSPEC(SMBUS);
```

一旦兩個變數進行了初始化，則可透過 smbus-> 使用整個 SMBus API，以及透過 pmbus-> 使用整個 PMBus API。可以在同一個應用中兼用這兩種 API。

LT_PMBusMath

LT_PMBusMath 類是一種高度優化的數位轉換庫，用於在 L11/L16 和浮點之間進行數值的來回轉換。浮點對於 PMBus 代碼是不需要的，而且有些終端使用者應用是僅利用整數編寫的，特別是在提前知道了電壓和電流值、或者依靠一個非常小微控制器進行浮點轉換過於緩慢的情況下。倘若韌體不需要進行此類轉換，則它們仍然可被某個離線應用所使用，以產生用於該應用的整數。然而，當函數採用浮點時編寫代碼就容易得多了。

LT_I2C 庫

LT_I²C 庫不同於 LT_PMBus 庫中存在的 I²C 類。LT_PMBus 中的版本除了支援　大塊操作之外還針對 PMBus 進行了位元組次序優化。另外，LT_PMBus 庫中的 I²C 類還基於 Wire 庫，而且至其他 Arduino 板的可攜性更高。例如，它可在 Arduino Mega 2560 上工作。

所有的非 PSM Sketch 均採用 LT_I²C 庫。最好不要把 LT_I²C 庫用於 PSM/PMBus 元件，而且沒有必要這樣做。 CTA

(未完，下期待續！)