用于电源系统管理的 Linduino

Michael Jones

引言

大多数电源系统管理设计都遵循一种「设定后便不需再过问」的模型。电源系统管理 (PSM) 器件的设置和调试利用 LTpowerPlay™ 是简单易行的，而且与一个批量编程解决方案组合时无需固件。不过，许多大型系统需要一个板级管理控制器 (BMC)，因而提出了这样的问题：「固件能够为 PSM 做什么呢？」

PSM 固件的基础是 PMBus；PMBus 的基础是 SMBus；而 SMBus 的基础则是 I2C。构建一个利用 PSM 固件增加价值的 BMC 需要对每种协议有一定程度的了解，或者一个预先存在的软件库以使编程人员摆脱细节的束缚。

Linduino™ 库负责处理每个协议层，并提供一个应用程序接口 (API)，从而使得 PSM 固件的编写十分容易。Linduino PSM 并不是 BMC 的一种替代品，而是一组可兼容典型 BMC 固件的软件库和示例。

另外，Linduino 还可作为一款学习工具与 LTC 演示电路一起使用。许多 BMC 设计已经具有一个 SMBus API，所有这些需要的是快速学习 PMBus 的工作原理。工程师们把 Linduino 代码片段复制 / 粘贴到现有的应用程序中并加以使用的现象是相当常见的。不过也可以实施 Linduino 层之一，然后重用整个软件库，包括：

- 器件和电源轨发现
- 命令 API
- 故障记录解码
- 系统内编程

本应用指南将陈述 Linduino 库、电源系统管理编程、具演示电路的 Linduino PSM 的设置和使用、以及 PSM 调试方法。如需了解有关协议和一般编程问题的详细信息，请查阅「应用指南 135」(Applicaton Note 135 - Implementing Robust PMBus System Software for the LTC3880) 以及针对 I^2C / SMBus / PMBus 的业界标准。

LINDUINO PSM 硬件

Linduino PSM 硬件包括一个 Linduino (DC2026) 和一个屏蔽 (DC2294)，以把 Linduino 的 I^2C 引脚连接到一块演示板或产品板的 PMBus / SMBus / I^2CBus。

为获得最佳的学习效果，可从一个 DC2026 (Linduino)、DC2294 (屏蔽)、DC1962 (Power Stick) 和一个 Total Phase Beagle (I^2C 嗅探器) 着手。这能提供控制器 (LTC388X) 和管理器 (LTC297X) 的编程、调试和学习。

图 1 (评估硬件) 示出了全部连接在一起的建议评估硬件。如欲使用该硬件，则利用两根 USB 电缆把 Linduino 和

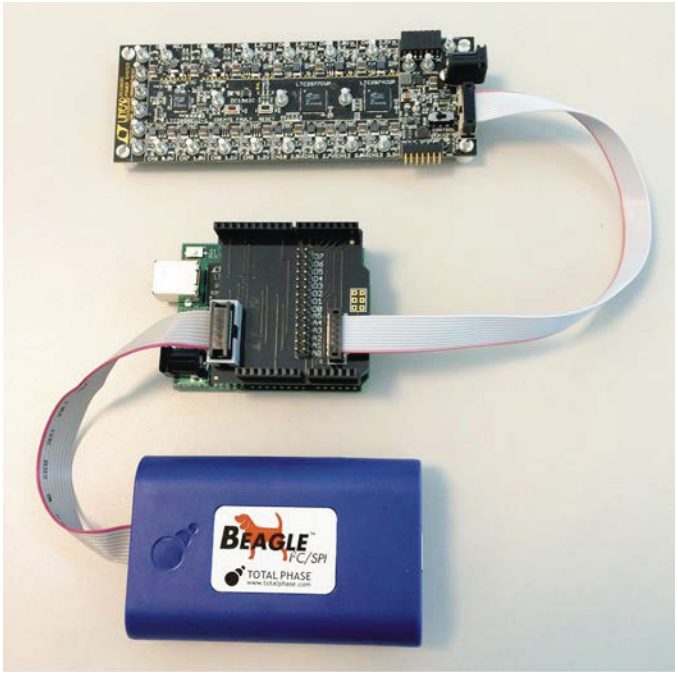

图 1：评估硬件

Beagle 连接至一台计算机。假如您没有用 USB 电缆连接 Beagle，那么就把 Beagle 带状电缆从 DC2294 断接，以避免干扰往来于它和 DC1962 之间的 PMBus 通信。

如果连接至一个系统板，则 DC2086 可在大多数情况下工作。

图 2：系统硬件

DC2086 将接受一个从 DC2294 引出的连接，并支持 12 针带状电缆、14 针带状电缆和 4 针电缆。另外，DC2086 还支持一个外部电源输入，该输入用于功率需求量高于 Linduino 所能提供之水平的系统板。

LINDUINO PSM SKETCH

在了解 PMBus 库的工作原理之前，快速浏览 DC1962 Sketch 将会弄清 Linduino PSM 的一般使用模型。此外，它还将演示编写代码程序有多么简单易行，即使对于非编程人员也不例外。

为进一步跟上，需要进行两项下载：Arduino 工具和 Linduino Sketchbook。Arduino 工具可从 www.arduino.cc 网站下载，而 Linduino Sketchbook 则可通过 www.linear.com.cn/linduino 下载。

Arduino 工具可在多种平台上的运行。本应用指南的构建和行文采用的是在 64 位 Ubuntu 14 TLS 上运行的 Arduino 1.6.4。

我们开始吧：

第一步：配置

当首次运行 Arduino 软件时，它将使用一个默认的 Sketchbook，而不是从 LTC 网站下载的 Linduino Sketchbook。

如欲变更 Linduino Sketchbook，则使用菜单栏上的 File | Preferences 选择，如图 3 (查找 Preferences 对话框) 所示。

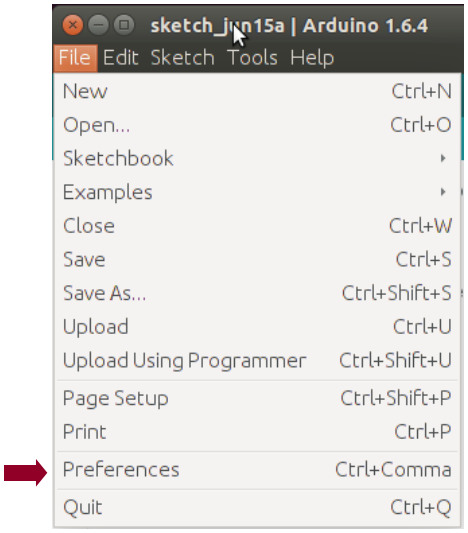

图 3：查找 Preferences对话框

图 4 (Preferences 对话框) 表明：Sketchbook Location 位于对话框的顶部。使用 Browse 按钮，导航至从 www.linear.com.cn/linduino 下载的 LTSketchbook。另

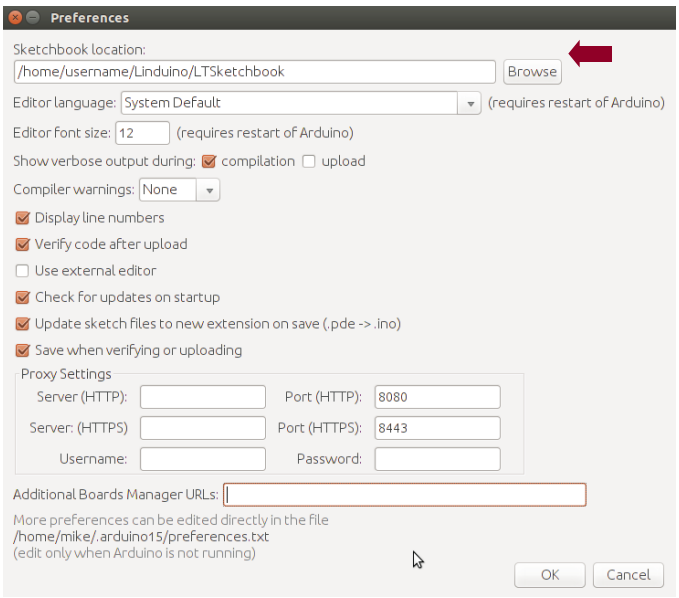

图 4：Preferences 对话框

外，在编译期间检查 ON Display 行号和 Show 详细输出也是有帮助的。后一个设置在命令行上滚动编译消息，它们在这里更容易看到。

在设定路径之后，必须关闭所有的 Arduino 窗口，而且必须重启 Arduino 软件。当重启 Arduino 时，它重新扫描 Sketchbook 目录并逐步建立 Arduino 菜单。如果未重启 Arduino 软件，则该菜单将不反映 LTSketchbook，而是指向前一个 Sketchbook。

第二步：加载您的第一个 Sketch：

通过仿效图 5 (加载 hello_world) 来加载 hello_world Sketch。

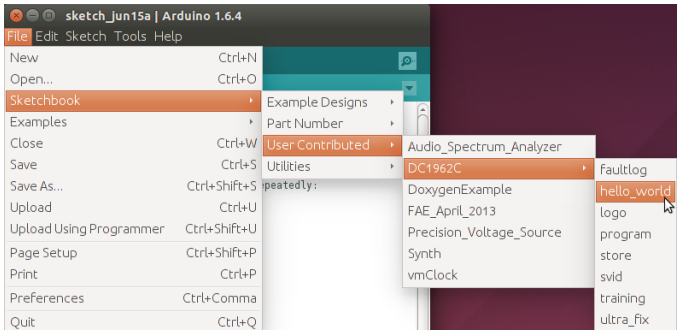

图 5：加载 hello_world

在加载了 Sketch 之后，弹出一个具有该 Sketch 的窗口，如图 6 (Sketch 窗口) 所示。

第三步：编译和运行

如图 7 (Arduino 工具栏) 所示，通过揿按工具栏上的钩形符号来编译 Sketch。

指向右的箭头将编译 Sketch 并把经过编译的 Sketch 装入 Linduino 硬件。屏幕放大镜显示 Sketch 的输出。把箭头看作发送代码至显示控制台，或编译代码并把代码发送至 Linduino 硬件，这样显示控制台就具有一些需要对话的东西。

注：Arduino 板类型应设定为 Arduino Uno，而且应选择端口。见工具菜单 (Tools Menu)。

在加载了 Sketch 之后，揿按位于工具栏右边的屏幕放大镜以打开控制台窗口。把行尾结束符号设定为 *Carriage return*，并将波特率设定为 115200，以与图 8 (Arduino 命令窗口) 相匹配。

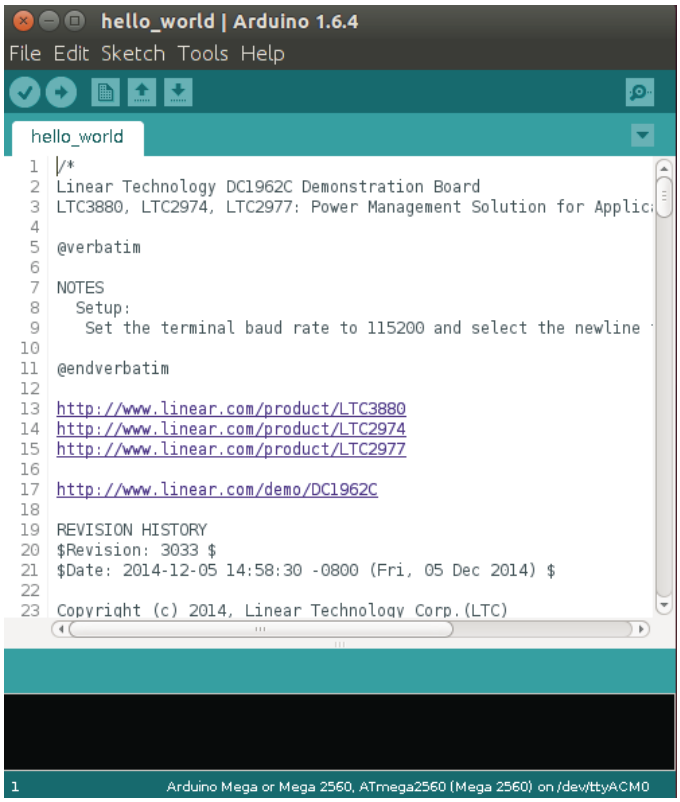

图 6：Sketch 窗口

图 7：Arduino 工具栏

如需与 Sketch 互动，则把光标置于顶部 (Send 按钮的左边) 的方框中，从菜单键入一个数字，然后揿按 Send 钮或 <CR>。Sketch 接著将执行命令并随后重新显示菜单。

第四步：考察菜单项

图 9 (Sketch 菜单) 示出了当揿按 1 以变更基本命令 (Basic Commands) 窗口、之后揿按 1 以读取所有电压 (Read All Voltages) 时会发生什么。DC1962 之所有电源轨的 VOUT 测量值被读出并打印。

在通过揿按位于左上角的 X 将控制台关闭之前，Sketch 一直处于运行状态。如果重新打开控制台，则其将重启 Sketch。

现在您可以考察其他的菜单选项，假如您熟悉 Beagle，则可排布一些走线并检查总线事务。

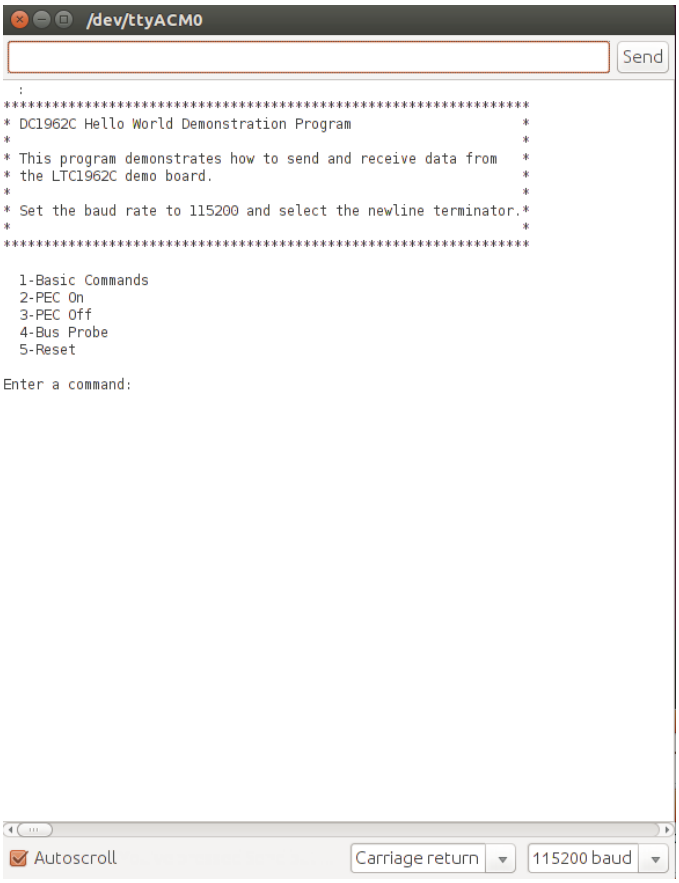

图 8：Arduino 命令窗口

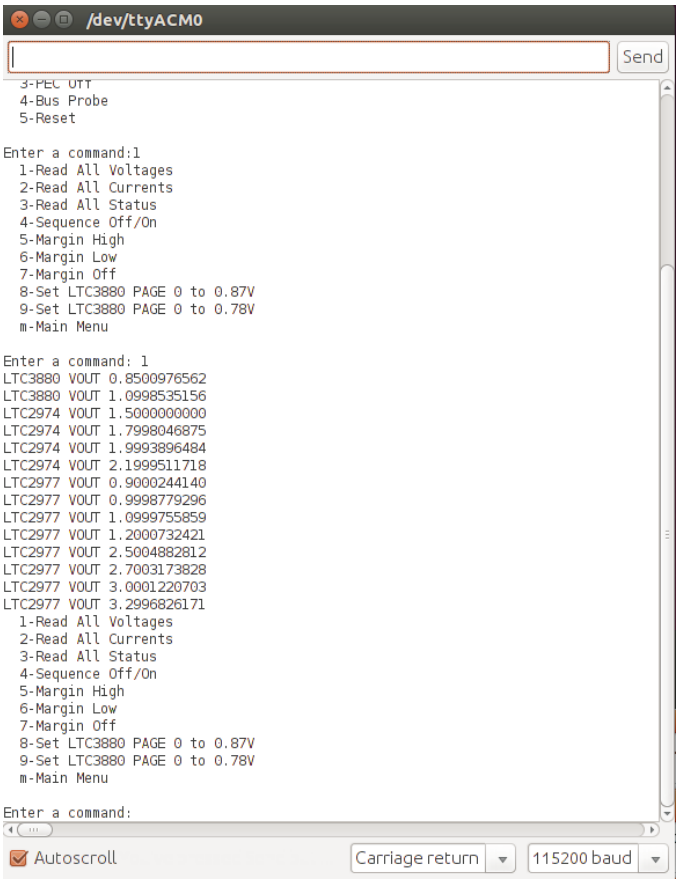

图 9：Sketch 菜单

第五步：代码的修改

Sketch 具有两个入口点。有一个setup() 函数 (其被调用一次)，和一个 loop() 函数 (其在一个循环中被永久地调用)。这些是 Arduino 编码环境的组成部分。如果您是一位有经验的 C 语言编程人员，则很有可能感到疑惑：main() 在哪里? Arduino 库具有一个调用 setup() 的预定义 main() 和一个无限循环调用 loop()。

菜单被编码为 Sketch 内部的帮助函数，而且 loop() 调用主菜单。每个菜单利用一个情况语句提供支持，这里每种情况处理一个菜单号。

好了，有关编程人员的内容说的足够了。修改应用程序就是简单地使用提供的 API 来变更 Sketch 中的情况语句。Sketch 中发出 PMBus 命令的函数 (API) 来自一个单独的库，并且具有听起来像您希望代码程序执行之任务的简单名称。例如：

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

意味著：使用 pmbus API，在地址 0x30 读取输出电压 (无轮询)，并将它转换成一个可变名称电压。

现在您应该做出少许变更了。例如，增添一个菜单项以读取和打印输出功率。如果您还没有准备好，就继续阅读以了解有关编写代码程序的更多内容。假如您已准备就绪，那么这里是一个提示：

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

在 LTC3880 上试用它 (在 Power Stick 上的地址 0x30)。为证明它是有用的，可把电阻或一个电流负载添加至 Power Stick 上的通道 0，并验证其与 Sketch 打印的内容相匹配。

LINDUINO PSM PMBus 库

PMBus 库存在于LTSketchbook/libraries/LT_PMBUS 目录中的 LTSketchbook 树之中。该软件库是分层的：从 TWI (两线式接口) 开始，然后是 I^2C、SMBus，最后是

PMBus。有一个用于在 L11/L16 (PMBus 格式) 和浮点之间来回转换数值的数字转换 API。最后，具有群组命令协议 (Group Command Protocol) 辅助、器件和电源轨发现、故障记录解码、乃至系统内编程功能。

每层是一个简单的 C++ 类，类似于 Arduino 把一个类别用于串行 (Serial) 和其他 I/O 函数的方式。如果您的最终环境是 C 语言，请不要担心。简单的意思就是您可以使用 C++ 类 (没有大量的内存开销)，或者也可以去除类包装并非常容易地将其用作纯 C 语言。C++ 包装器就是简化了应用程序代码并使其对于非编程人员而言更加容易。

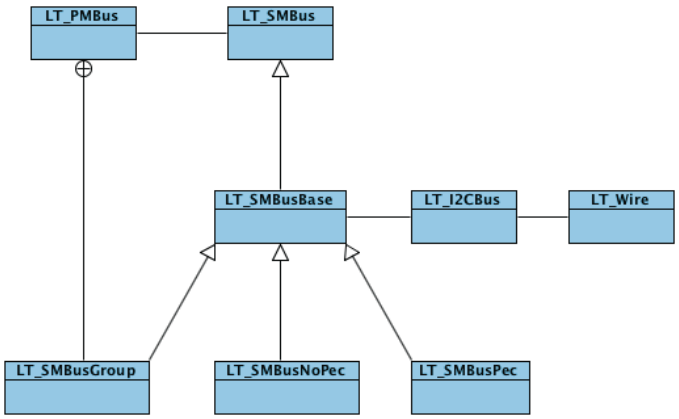

图 10：LT_PMBus 类示意图

对于那些仅必须了解内部结构的编程人员来说，SMBus 位于一个层次体系中，这样一来应用代码就与接通和关断 PEC 无关，并可帮助移植。LT_I2C.h、LT_SMBus.h 和 LT_PMBus.h 形成了 API 的层。如欲移植 Linduino PSM 库，则可选择任何 API 之一并在您的平台上实施 (使用您自己的软件库)。最常用的端口重新实施 LT_SMBusBase 类，然后 PMBus 类正常工作，数学转换正常运行，而且所有其他的函数和示例正常运作。

使用 PMBus 库

这个库的使用可以无需了解所有这些类成员；仅仅需要几个导入和静态变量，Sketch 就做好了动作准备。

一般来说 PMBus 库是利用 Sketch 菜单添加的，如图 11 (包括库 [Include Library]) 中所示。但是最重要包括的则示于图 12 (基础包括 [Base Includes])。

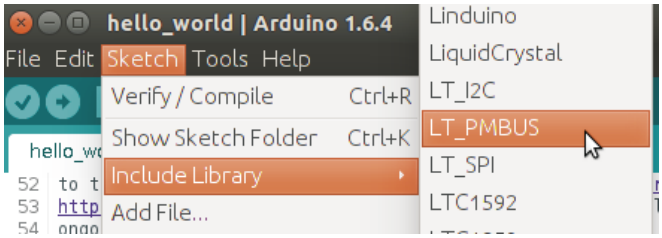

图 11：包括库 (Include Library)

```
#include <LT_SMBusPec.h>
#include <LT_SMBusNoPec.h>
#include <LT_SMBus.h>
#include <LT_SMBusMath.h>
```

图 12：基础包括 (Base Includes)

一个 Sketch 具有至少两个静态变量，一个用于系统管理总线 (smbus)，另一个用于电源管理总线 (pmbus)，如图 13 (静态变量) 所示。Smbus 变量是 Pec 或 NoPec 版本。乾净层 (clean layers) 的一个优良的特点是编程人员可根据其项目的需要把应用代码编写为 SMBus 或 PMBus 代码。利用 PMBus API 来编写应用隐藏了命令代码和数据格式化的细节，而利用 SMBus API 编写应用则可实现对所有可能命令代码的访问以及对原始值的直接访问。

```
static LT_SMBus *smbus = new LT_SMBusPec();
static LT_SMBus *smbus = new LT_PBusPec(smbus);
```

图 13：静态变量

一旦两个变量进行了初始化，则可通过 smbus-> 使用整个 SMBus API，以及通过 pmbus-> 使用整个 PMBus API。可以在同一个应用中兼用这两种 API。

LT_PMBusMath

LT_PMBusMath 类是一种高度优化的数字转换库，用于在 L11/L16 和浮点之间进行数值的来回转换。浮点对于 PMBus 代码是不需要的，而且有些终端用户应用是仅利用整数编写的，特别是在提前知道了电压和电流值、或者依靠一个非常小微控制器进行浮点转换过于缓慢的情况下。倘若固件不需要进行此类转换，则它们仍然可被某个离线应用所使用，以产生用于该应用的整数。然而，当函数采用浮点时编写代码就容易得多了。

LT_I2C 库

LT_I2C 库不同于 LT_PMBus 库中存在的 I^2C 类。LT_PMBus 中的版本除了支持大块操作之外还针对 PMBus 进行了字节次序优化。另外，LT_PMBus 库中的 I^2C 类还基于 Wire 库，而且至其他 Arduino 板的可移植性更高。例如，它可在 Arduino Mega 2560 上工作。

所有的非 PSM Sketch 均采用 LT_I2C 库。最好不要把 LT_I2C 库用于 PSM/PMBus 器件，而且没有必要这样做。

编写一个简单的 Sketch

最佳的学习方法是从头编写代码，本节所涉及的内容即在于此。如果您自己执行这些步骤 (一次一个) 并在逐步执行的过程中验证结果，那么下面的示例将是最有帮助的。

该 Sketch 示例采用一个 DC1962 和在第一节中提到的另一个硬件。此示例接受 5 个简单的命令：

1. 打印电压
2. 裕度调节
3. 接通 / 关断
4. 总线探测
5. 复位

第一步：创建一个空白的 Sketch

如图 14 (新的 Sketch) 所示，通过选择 File|New 以创建一个新的 Sketch。然后，您将看到一个如图 15 (空的 Sketch) 所示的空 Sketch。

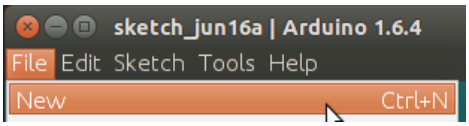

图 14：新的 Sketch (New Sketch)

使用 File|Save As... 菜单，并为新的 Sketch 选择一条路径。要确保文件夹名称和 Sketch 名称匹配，如图 16 (Save As...) 所示。该路径必须在 LTSketchbook 的下方 (File|Preferences 对话框中的相同路径) 以便其在 Sketchbook 菜单中显现出来。

图 15：空的 Sketch (Empty Sketch)

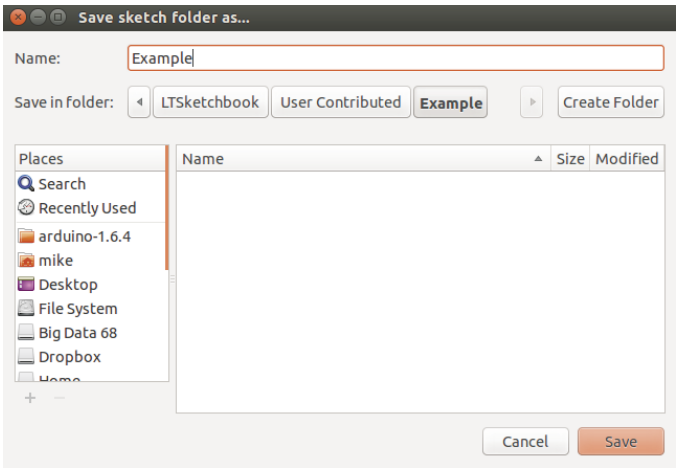

图 16：Save As … (另存为 …)

第二步：添加包括

使用 Sketch|Include|Include Library 菜单，选择下面的库 (一次一个库)：

- UserInterface
- Linduino
- LT_PMBUS

图 17 (Includes) 示出了怎样选择 LT_PMBUS 库。所有的库都在同一个 Include Library 菜单中。

在文件的头一行上添加该包含语句：

```
#include <Arduino.h>
```

现在，添加用于地址和 SMBus/PMBus 对象的静态变量。增添设置 (Setup) 代码以预置变量和串行总线对象。利用 File|Save 保存该代码。最后，掀按工具栏上的检查按钮以编译代码。

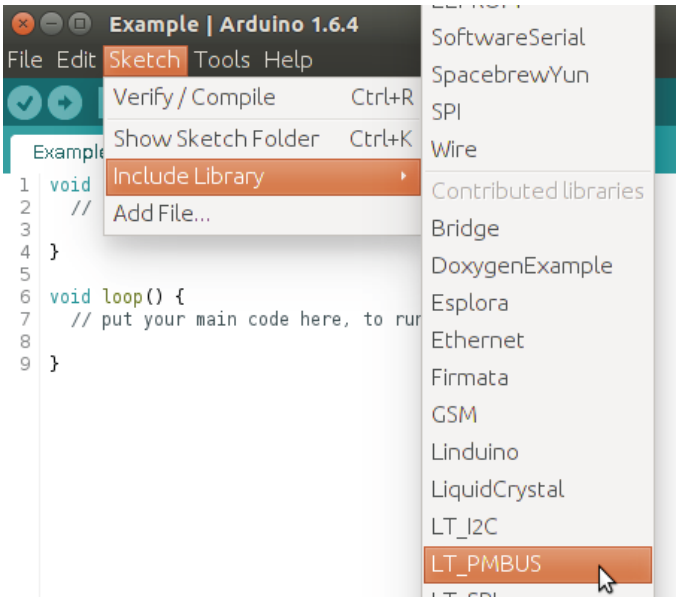

图 17：包括 (Includes)

您的代码应看似图 18 (初始化代码) 的样子。

```
Example
#include <Arduino.h>
#include <UserInterface.h>
#include <Linduino.h>
#include <LT_Wire.h>
#include <LT_twi.h>
#include <LT_SMBusPec.h>
#include <LT_PMBusMath.h>
#include <LT_SMBus.h>
#include <LT_I2CBus.h>
#include <LT_SMBusGroup.h>
#include <LT_PMBus.h>
#include <LT_FaultLog.h>
#include <LT_SMBusNoPec.h>
#include <LT_SMBusBase.h>

#define LTC3880_I2C_ADDRESS 0x30
#define LTC2974_I2C_ADDRESS 0x32
#define LTC2977_I2C_ADDRESS 0x33

static uint8_t ltc3880_i2c_address;
static uint8_t ltc2974_i2c_address;
static uint8_t ltc2977_i2c_address;
static LT_SMBus *smbus = new LT_SMBusNoPec();
static LT_PMBus *pmbus = new LT_PMBus(smbus);

void setup() {
  Serial.begin(115200);
  ltc3880_i2c_address = LTC3880_I2C_ADDRESS;
  ltc2974_i2c_address = LTC2974_I2C_ADDRESS;
  ltc2977_i2c_address = LTC2977_I2C_ADDRESS;
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

图 18：初始化代码

第三步：设置菜单

一个菜单需要打印选择，以及对用户选择的响应。

添加一个 print_prompt() 函数以打印一个提示，并从设置函数调用它以在 Sketch 运行时打印菜单提示。代码应看起来像图 19 (Prompt)。

保存和编译以确保代码没有错误。

```
void print_prompt()
{
  Serial.print(F("\n  1-Print Voltages\n"));
  Serial.print(F("  2-Margin High\n"));
  Serial.print(F("  3-Margin Low\n"));
  Serial.print(F("  4-No Margin\n"));
  Serial.print(F("  5-Go Off\n"));
  Serial.print(F("  6-Go On\n"));
  Serial.print(F("  7-Probe Bus\n"));
  Serial.print(F("  8-Reset\n"));
  Serial.print(F("\nEnter a command:"));
}

void setup() {
  Serial.begin(115200);
  ltc3880_i2c_address = LTC3880_I2C_ADDRESS;
  ltc2974_i2c_address = LTC2974_I2C_ADDRESS;
  ltc2977_i2c_address = LTC2977_I2C_ADDRESS;
  print_prompt();
}
```

图 19：Prompt

第四步：添加菜单响应

当一个菜单选项被键入控制台时，代码程序必须读取它并做出响应。

环路函数将处理用户输入。首先，它必须检查串行总线是可用的。然后，它必须把输入作为一个整数读取并将之传递至一个开关语句。在开关语句的内部，它必须执行某些函数并随后调用提示函数。用于每个命令的代码将在开关语句每种情况的内部编写，而提示函数将在之后调用。您的代码框架应看上去与图 20 (用户输入) 相似。

对它进行保存和编译以确保其没有错误。

第五步：读取电压

现在是编写可做些有用工作的实际 PMBus 代码的时候了。

图 21 (读取电压) 示出了负责读取所有电压的代码。代码在情况 1 的内部示出。两个变量保持电压和页面：一个用于电压的 float，和一个用于页面的 unit8_t，显示在第 57~58 行。打印使用标准的 Arduino Serial.print··· 函数。

```
void loop() {
  uint8_t user_command;

  if (Serial.available())
  {
    user_command = read_int();

    switch (user_command)
    {
      case 1:
        break;
      case 2:
        break;
      case 3:
        break;
      case 4:
        break;
      case 5:
        break;
      case 6:
        break;
      case 7:
        break;
      case 8:
        break;
    }
    print_prompt();
  }
}
```

图 20：用户输入

```
    switch (user_command)
    {
      case 1:
        float   voltage;
        uint8_t page;

        Serial.println(F(""));
        for (page = 0; page < 2; page++)
        {
          pmbus->setPage(ltc3880_i2c_address, page);
          voltage = pmbus->readVout(ltc3880_i2c_address, false);
          Serial.print(F("LTC3880 VOUT "));
          Serial.println(voltage, DEC);
        }

        for (page = 0; page < 4; page++)
        {
          pmbus->setPage(ltc2974_i2c_address, page);
          voltage = pmbus->readVout(ltc2974_i2c_address, false);
          Serial.print(F("LTC2974 VOUT "));
          Serial.println(voltage, DEC);
        }

        for (page = 0; page < 8; page++)
        {
          pmbus->setPage(ltc2977_i2c_address, page);
          voltage = pmbus->readVout(ltc2977_i2c_address, false);
          Serial.print(F("LTC2977 VOUT "));
          Serial.println(voltage, DEC);
        }
        break;
      case 2:
        break;
```

图 21：读取电压

字符串周围的 F() 把它们放入闪存中，这样它们就不会使用宝贵的 RAM。对于每个器件，一个 for loop 通过调用 pmbus->setPage(...) 的页面进行检索，随后利用 pmbus->readVout(...) 以读取电压。接著，代码以十进制打印电压 (采用 DEC 类型)。

您可以找到自己在 PMBus.h 文件中的 LT_PMBus 库或 Doxigen 文档里使用的所有 API 函数声明。

第六步：裕度调节和接通 / 关断

裕度调节代码比电压代码简单，因为操作是全局的，这意味著所有的器件能够响应一个命令，而且不需要页面寄存器。此外，没有任何东西要打印。

图 22 (裕度调节和接通 / 关断) 示出了代码。情况 4 是 No Margin 菜单选择。采用 sequenceOnGlobal() 来结束裕度调节也许看起来很奇怪。在内部结构中用于这些操作的 PMMBus 命令是 OPERATION (0x01) 命令。

```
      case 2:
        pmbus->marginHighGlobal();
        break;
      case 3:
        pmbus->marginLowGlobal();
        break;
      case 4:
        pmbus->sequenceOnGlobal();
        break;
      case 5:
        pmbus->sequenceOffGlobal();
        break;
      case 6:
        pmbus->sequenceOnGlobal();
        break;
```

图 22：裕度调节和接通 / 关断

取自 LTC3880 产品手册的图 23 (OPERATION 命令) 表明：没有用于停止裕度调节的专用命令。裕度调节利用数值 0x80 (它意味著接通) 来关断。这就是采用 pmbus->sequenceOnGlobal() 来关断裕度调节操作的原因。

表 4：当 On-Off_Config_Use_PMBus 使能 Operation_Control 时的 OPERATION 命令细节寄存器 OPERATION 数据内容

符号	动作	数值
位		
函数	立即关断	0x00
	接通	0x80
	裕度调节低	0x98
	裕度调节高	0xA8
	序列关闭	0x40

图 23：OPERATION 命令

第七步：总线探测和复位

探测总线是 SMBus API 的一部分，毕竟并非所有的器件均为 PMBus。图 24 (探测和复位) 示出：探测是对 smbus->probe(0) 的调用。0 是它用以实施探测的命令，

其为 PAGE (0x00) 命令。探测将测试所有的有效地址并回送一个所发现之器件的清单。它将找到所有能够确定收到一个读命令 0x00 的器件。

复位命令不太明显。LTC388X 和 LTC297X 系列的复位方法不同。LTC388X 器件支持一个 MFR_RESET (0xFD) 命令，但是 LTC297X 器件则不支持。例如，在 LTC2977 上 0xFD 命令为 MFR_TEMPERATURE_MIN，而不是 MFR_RESET。使一个管理器复位的正确方法是从非易失性存储器 (NVM) 恢复 RAM，因为在该变换之后器件复位。

然而，如欲使所有的器件同时复位，则使用群组命令协议。这把所有的操作编组为单个事务，在这里所有的命令均由 PSM 器件在 STOP 执行。

图 24 (探测和复位) 中的 case 8 示出了怎样设置一个群组协议事务。该事务受pmbus->startGroupprotocol() 和 pmbus->executeGroupProtocol() 调用的束缚。

```
100       case 7:
101         uint8_t *addresses;
102         addresses = smbus->probe(0);
103         while(*addresses != 0)
104         {
105           Serial.print(F("ADDR 0x"));
106           Serial.println(*addresses++, HEX);
107         }
108         break;
109       case 8:
110         pmbus->startGroupProtocol();
111         pmbus->reset(ltc3880_i2c_address);
112         pmbus->restoreFromNvm(ltc2974_i2c_address);
113         pmbus->restoreFromNvm(ltc2977_i2c_address);
114         pmbus->executeGroupProtocol();
115         break;
```

图 24：探测和复位

第八步：测试

这是编译和运行应用程序并确信其工作状况一切正常的好时机。

假如应用程序运行，但并不打印合理的数据，那么您或许犯了某种错误。您可以使用下面的调试方法对其进行调试。或者，如果您没有耐心，则可只进行复查：

- 地址
- 页面
- 中断语句

调试

有几种调试一个 Linduino PSM 应用程序、或在这方面的任何固件应用程序的方法：

- 打印
- Spy 工具
- 调试器

本应用指南将不追求第三种选项。对于简单的 Sketch 它通常不是必需的。如果您希望了解有关调试器的更多信息，则转至 Arduino 网站论坛，看看其他人使用的是什么工具。

您已经看见了在上面的实例中使用的打印。您可通过添加更多的打印语句以进行调试。不过，应始终把字符串放在 F() 宏指令的内部，这样 RAM 就不会被耗尽。当打印文本和数字时将其分成两个调用，于是文本部分便在闪存中了。

PSM 库采用这种方法。误差 (比如：NACK 和 PEC 误差) 在命令窗口中打印。因此，增添调试打印通常被限制在应用程序代码。

您已经看见了采用 DEC 的打印。您也可以使用 HEX 和其他格式。查阅 Arduino 文档以获得有关格式化的更多帮助。

用于 PMBus 的最好调试器是 Spy 工具。Spy 工具之所以很好，原因是您能看见总线上的通信量，而且在需要帮助的时候您可以把一个跟踪记录与您的代码一起发送给 LTC 应用技术工程师。

本应用指南将重点关注由与 Total Phase Beagle 对话的 Total Phase Data Center 应用程序产生的数据。在 Total Phase 的网站上提供了帮助您安装 Data Center Application 的信息 (www.totalphase.com)。

启动开发工作最简单的方法是使用您刚刚创建的 Sketch 来跟踪总线。将采用菜单选择 3 (读取电压)。

图 25 (Beagle跟踪) 示出了数据。让我们直接投入并对某些事务进行解密，并采用索引以记录我们所在的位置。

在 Index #1 (I1) 和 index #6 (I6)，有两项写字节事务。在 SMBus 中，这是写字节协议 (Write Byte Protocol)。地址为 0x30，这是 LTC3880，如在代码中可以看到的那样。第一个字节是命令，其为 0x00，它是页面 (PAGE) 命令。

Index	m:s.ms.us	Dur	Len	Err	S/P	Addr	Record	Data
0	0:00.000.000						Capture start...	[Tue 16 Jun
1	0:08.432.357	305 us	2 B		SP	30	Write Transac...	00 00
2	0:08.432.702	209 us	1 B		S	30	Write Transac...	8B
3	0:08.432.912	302 us	2 B		SP	30	Read Transac...	9A 0D*
4	0:08.433.249	209 us	1 B		S	30	Write Transac...	20
5	0:08.433.459	207 us	1 B		SP	30	Read Transac...	14*
6	0:08.435.261	305 us	2 B		SP	30	Write Transac...	00 01
7	0:08.435.616	209 us	1 B		S	30	Write Transac...	8B
8	0:08.435.825	309 us	2 B		SP	30	Read Transac...	9A 11*
9	0:08.436.170	209 us	1 B		S	30	Write Transac...	20
10	0:08.436.380	207 us	1 B		SP	30	Read Transac...	14*
11	0:08.438.191	305 us	2 B		SP	32	Write Transac...	00 00
12	0:08.438.541	210 us	1 B		S	32	Write Transac...	8B
13	0:08.438.752	310 us	2 B		SP	32	Read Transac...	FC 2F*
14	0:08.439.096	209 us	1 B		S	32	Write Transac...	20
15	0:08.439.306	207 us	1 B		SP	32	Read Transac...	13*
16	0:08.441.132	305 us	2 B		SP	32	Write Transac...	00 01
17	0:08.441.478	215 us	1 B		S	32	Write Transac...	8B
18	0:08.441.693	302 us	2 B		SP	32	Read Transac...	9B 39*
19	0:08.442.031	209 us	1 B		S	32	Write Transac...	20
20	0:08.442.240	207 us	1 B		SP	32	Read Transac...	13*
21	0:08.444.047	305 us	2 B		SP	32	Write Transac...	00 02
22	0:08.444.397	209 us	1 B		S	32	Write Transac...	8B
23	0:08.444.606	310 us	2 B		SP	32	Read Transac...	02 40*
24	0:08.444.951	209 us	1 B		S	32	Write Transac...	20
25	0:08.445.161	207 us	1 B		SP	32	Read Transac...	13*
26	0:08.446.967	305 us	2 B		SP	32	Write Transac...	00 03
27	0:08.447.322	209 us	1 B		S	32	Write Transac...	8B
28	0:08.447.532	310 us	2 B		SP	32	Read Transac...	65 46*
29	0:08.447.877	209 us	1 B		S	32	Write Transac...	20
30	0:08.448.086	207 us	1 B		SP	32	Read Transac...	13*
31	0:08.449.903	305 us	2 B		SP	33	Write Transac...	00 00
32	0:08.450.258	214 us	1 B		S	33	Write Transac...	8B
33	0:08.450.473	302 us	2 B		SP	33	Read Transac...	CD 1C*
34	0:08.450.810	209 us	1 B		S	33	Write Transac...	20
35	0:08.451.020	212 us	1 B		SP	33	Read Transac...	13*
36	0:08.452.816	305 us	2 B		SP	33	Write Transac...	00 01
37	0:08.453.171	209 us	1 B		S	33	Write Transac...	8B
38	0:08.453.381	302 us	2 B		SP	33	Read Transac...	01 20*
39	0:08.453.718	209 us	1 B		S	33	Write Transac...	20
40	0:08.453.928	207 us	1 B		SP	33	Read Transac...	13*
41	0:08.455.725	305 us	2 B		SP	33	Write Transac...	00 02
42	0:08.456.075	209 us	1 B		S	33	Write Transac...	8B

图 25： Beagle 跟踪

LTC3880 产品手册中的表 2 示出了 PAGE 命令。这张表是对 Beagle 数据进行解码的快速方法。请注意 Type 列显示的是 R/W Byte。这意味著寄存器是读 / 写字节协议，因此两个方向均得到支持。

Table 2. Summary (Note: The Data Format abbreviations are detailed at the end of this table.)

COMMAND NAME	CMD CODE	DESCRIPTION	TYPE	PAGED	DATA FORMAT	UNITS	NVM	DEFAULT VALUE	PAGE
PAGE	0x00	Channel or page currently selected for any command that supports paging.	R/W Byte	N	Reg			0x00	63

图 26：PAGE 命令

回过头去看一下 I1 和 I6，第二个字节是一个 0x00 和 0x01。代码将把 PAGE 寄存器设定为 0 和 1。重新提及图 21 (读取电压) 中的第 63 行，这是设定页面的地方。我们应查看在紧接此之后的 I2 至 I5 上处于读取之中的电压。

我们看到：

写 0x8B，读 0x9A 0x0D

写 0x20，读 0x14

0x8B 是一个 READ_VOUT 命令。产品手册中的表 2 显示它是一个 R 字协议，而且采用的是 L16 格式。

0x20 是一个 VOUT_MODE 命令。产品手册中的表 2 显示它是一个 R 字节命令。

导致这些发生的 Linduino 调用示于图 21 (读取电压) 中的第 64 行。

为什么单个函数调用发布了两项事务? 为弄清原因，我们必须了解 API 背后的代码，示于图 27 (读取 VOUT 代码)。

```
vout_L16 = smbus_.readWord(address, READ_VOUT);
exp = (int8_t)

  (smbus_.readByte(address, VOUT_MODE) & 0x1F);
return math_.lin16_to_float(vout_L16, exp);
```

图 27：读取 VOUT 代码

该代码示出了一个后随 smbus_.readByte(address, VOUT_MODE) 的 smbus_.readWord(address, READ_VOUT)，而且 5 个位从模式抽取并被置入变量 exp。数学转换然后采用指数 exp 实现从 L16 至浮点的转换。

大致说来，用于读取电压的代码是负责读取用于把 L16 转换为浮点之指数的通用代码。LTC388X 和 LTC297X 器件采用了一个不同的指数。这就是存在两项事务的原因。

注：代码可利用指数的某种先验知识来编写，而且它的运行速度快一点。不过，通用代码将具有较少的漏洞，且对于应用程序编写者更加容易。当编写自己的代码时这是您必须做出的权衡。

在结束之前，让我们看一个更加有趣的事务：复位。

看一下图 28 (复位跟踪)，并需注意到有三个写事务。在 S/P 列中，有两个 S 和一个 SP。这意味著 I1 是一个启动 (Start)，I2 是一个重复启动 (Repeated Start)，I3 是一个重复启动 (Repeated Start)，之后是一个停止 (Stop)。此外，它们各自的地址是不同的：0x30、0x32 和 0x33。

Index	m:s.ms.us	Dur	Len	Err	S/P	Addr	Record	Data
0	0:00.000.000						Capture start...	[Tue 1
1	0:04.867.488	209 us	1 B		S	30	Write Transac...	FD
2	0:04.867.698	244 us	1 B		S	32	Write Transac...	16
3	0:04.867.942	250 us	1 B		SP	33	Write Transac...	16
4	0:08.899.681						Capture stop...	[Tue 1

图 28：复位跟踪 (Reset Trace)

这是群组命令协议。所有的命令都将在 I3 的末端 (STOP) 处理。这与图 24 (探测和复位) 中所示代码的第 110~114 行相关联。

如果您在自己的 Beagle 轨迹解码上花些时间，就将对 PSM 器件的 PMBus 命令获得更加完整的了解。另一方面，假如您的目标是使代码正常运行，那么 PSM 库非常愿意为您完成繁重的任务。

采用 LTpowerPlay 的高级调试

回到引言部分，那里说大多数系统是「设定后便不需再过问」的，而且有些系统具有一个 BMC。事实是那些具有一个 BMC 的系统把「设定后便不需再过问」与固件组合在一起。为什么让一个 BMC 承担了全部的设置责任呢? 这是因为把某种基本设置写入 PSM 器件、然后仅将 BMC 固件用于添加的价值功能要容易得多。另外，这还造就了更加可靠的系统，因为大多数固件负责读取遥测数据、裕度、并实施轻微的电压变更，不需要让它控制排序或始终是静态的 PWM 频率等关键功能。

由于 LTpowerPlay 是一款用于设计、调试和调通 PSM 系统的通用型工具，因此调试固件必须与物理时钟和数据线上的另一个 PMBus 主控器进行竞争。

在进入两个主控器的实际意义之前，最好是仔细研究一下当一根 PMBus 具有两个主控器时会发生什么。PMBus 基于 SMBus，这包括多主控。

时钟和数据线是开路漏极。这意味著任何器件 (主控器或受控器) 都能够下拉一根线的电平，但不能上拉其电平。有这样一条规则：当一个主控器未下拉数据线电平、并检测到数据线为低电平时，则其认为有另一个主控器在把数据线拉至低电平，于是中止其自身的事务，从而允许另一个主控器继续完成其事务。

该方法有时被称为「位优势仲裁」(bit dominance arbitration)，这是一种指明「在数据中将一个零置为有效的主控器总是获胜」的奇特说法。

Linduino 和 LTpowerPlay (DC1613) 多主控器，而且您也许认为一切都很完美了。不过，还有一项更为关键的考虑因素。

PMBus 定义了一个 PAGE 命令 (0x00)，它像一个进入数据的地址。页面像是通道一样。例如，一个 LTC2977 能够管理 8 个电源：它拥有 8 个通道，各由 PAGE 寄存器 / 命令进行寻址。

实际上这意味著：为读取某种数值 (如电压)，它需要两项事务：一项用于 PAGE，一项用于 READ_VOUT。如果两个主控器同时试图从同一个受控器读取遥测数据，而且倘若一个主控器在另一个主控器的页面命令和遥测命令之间插入了一个页面命令，则它将读取错误的页面。

当 LTpowerPlay 启动并运行时，它所做的首要之事是读取遥测数据，这将保持其状态显示为最新，于是您就能查看输出的曲线图、故障和其他重要信息。猜猜固件通常做什么? 它读取遥测数据！

更加糟糕的是，假设固件在启动的时候执行一项 VID (电压识别) 功能。假使固件由于 LTpowerPlay 修改了 PAGE 寄存器而把一个电压值写至错误的页面，将会怎么样呢? 系统有可能发生故障关断，甚至更糟地造成某些东西受损。(幸运的是，VOUT_MAX 寄存器通常可防止系统损坏)

PAGE 命令的基本问题是 PMBus 规格中固有的。它并不是 LTC 电源系统管理器件所特有的，而且必须对其进行处置。

有两种允许 LTpowerPlay 与固件共存并避免发生 PAGE 问题的基本方法。第一种方法就是简单地不让两个主控器同时谈话。第二种方法是在一个或另一个主控器上使用 PAGE PLUS 协议和其他的技巧。

我们避开 PAGE PLUS，因为它不是常用的。PAGE PLUS 提供了一种原子事务，其在一项事务中包括 PAGE 和 COMMAND。由于并非所有的器件都支持它，所以它通常仅在特殊场合中使用，因此本应用指南将不把重点放在 PAGE PLUS 和其他诀窍上。如果您没有解决此问题的其他方法，则可阅读 LTC PSM 产品手册，或致电当地的应用技术工程师以寻求帮助。

更常见的是防止 LTpowerPlay 和固件同时与受控器对话。LTpowerPlay 具有一种非常简单控制其运行的方法。图 29 (LTpowerPlay 启动 / 停止) 示出了遥测曲线图，它在其工具栏上具有一个红色方形按钮。

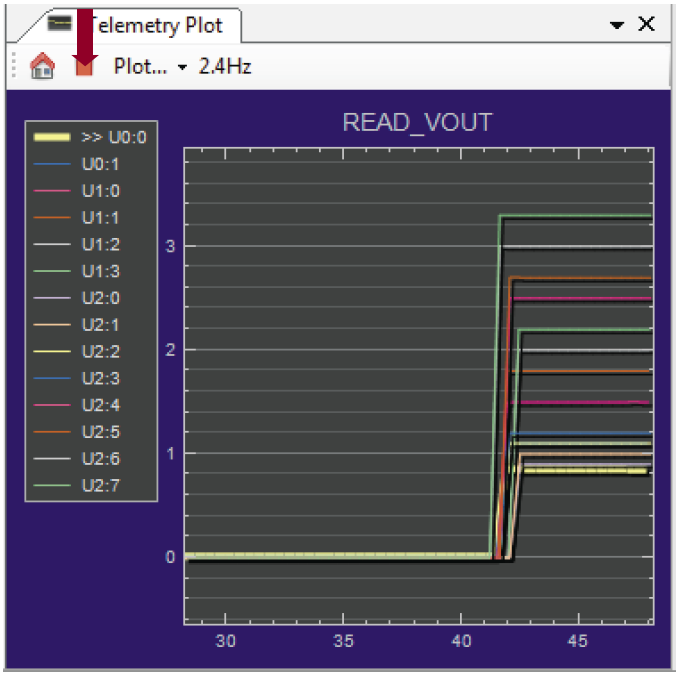

图 29：LTpowerPlay 启动 / 停止

当按下此按钮时，它会停止总线上的所有遥测和所有的 LTpowerPlay 动作。然后，它变为一个绿色箭头，如图 30 (停止的 LTpowerPlay) 所示。

图 30：停止的 LTpowerPlay

不幸的是，固件并非总是具有一种用于实现总线静噪的内置机制。LTC 简单地建议利用一种内置静噪机制、或者通过提供一个硬件总线开关 / 多工器 (MUX) 或跳线来设计新型固件。

一旦有了一种使总线主控器、LTpowerPlay 或固件静噪的方法，那么调试只是一件根据需要在工具之间进行交替的简单事。

总之，有两种选择：

1. 一次只允许一个主控器在总线上进行对话
2. 在 LTC 应用技术工程师的帮助之下粗略地了解 PAGE PLUS 及其他先进方法的细节

对于一款新设计而言，选项 1 始终是最佳的选择。

概要

利用 Linduino PSM Sketchbook、一块 Linduino 开发板和任选的 DC2294 屏蔽，可使针对 PSM 器件的程序代码编写变得简单。软件库具有一个用于 SMBus 和 PMBus 的简单 API。LTpowerPlay 仍可用于调试，而且可附接一个 Total Phase Beagle 或其他的 Spy Tool 以观察总线上的通信量。

不管您是希望移植代码还是仅仅希望了解 PMBus 的工作原理，Linduino 都是启动开发工作的一种绝佳的方法。一旦您加快了学习曲线，并了解了 PSM 编程的基础知识，就能提升自己的工作进度并充满信心地使用其他工具。

现在完成了本应用指南的阅读，您或许希望察看与 LTSketchbook 配套提供的其他 Sketch。试用一些更高级的 Sketch，比如：故障记录解码 (Fault Log Decoding) 或系统内 / 运行中更新 (In System/Flight Update)。