AN2345

采用 MIC45116 电源模块实现基于 PIC16 的四通道电源定序器

作者： *Stan D'Souza*
Microchip Technology Inc.

简介

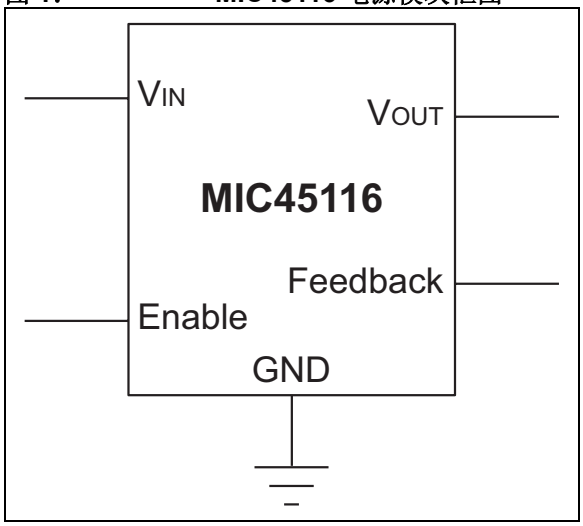
电源定序器通常用于系统级电路板设计，以顺序方式使能多个电源。通常，使用电源定序器的系统具有不同组件，这些组件要求不同的电源电压和功率水平。按序启动不同电压将确保组件上电之间没有冲突且所有部件均正确上电。关闭系统时，也会有相应的顺序。上电和掉电序列可基于时间进行编程。本应用笔记定义了一个四通道电源定序器。定义的四个电压是 5.0V、3.3V、2.5V 和 1.8V。每个电压都通过 Microchip MIC45116 电源模块（Power Module，PM）提供。这些 PM 有多个引脚；主要引脚是：输入 / 输出电压、地、片选（Enable）和反馈（Feedback）。用户可选择任意数量的 PM 用于其系统。软件以模块化格式编写，可容纳最多 10 个 PM。可添加 / 除去 PM 满足特定需求。选择增强型内核 PIC16F18344 器件作为本应用的 MCU。如果需要更大的程序存储器或数据存储器，可轻松将 PIC16F18344 替换为引脚兼容的 PIC16F18345，其程序存储器和数据存储器是 PIC16F18344 的两倍。

硬件设计注意事项

Microchip MIC45116 电源模块（PM）

PM 是电源模块，具有特定电压、电流和功率容量。我们在此设计中使用的 PM 是由 Microchip 制造的 MIC45116 系列（图 1）。每个 PM 有多个引脚，然而，本设计只需五个引脚：输入电压（VIN）、输出电压（VOUT）、地（GND）、Enable 输入和 Feedback。Enable 信号为高电平有效，且使能时，电压将出现在 PM 的 VOUT 引脚上。该电压与连接到 PM 的 Feedback 引脚的 RTOP 和 RBOTTOM 电阻直接相关。用户可使用图 2 中的公式选择 RTOP 和 RBOTTOM 电阻。此外，Feedback 引脚允许输出电压有 ±X mV 的裕量（其中，X mV 由用户定义）。Feedback 引脚需要直流电压并由 PIC® 单片机提供。在本设计中，直流电压由 PWM 控制的 DAC 输出经过基于 RC 的滤波器电路提供（图 3）。如果设计不需要裕量，则仅在 Feedback 引脚使用外部电阻。由于 Feedback 引脚不需要 DAC 电压，则省去了 PIC16 MCU 中的 PWM 及与其相关的 RC 电路和软件。请参见“*MIC45116 20V/6A DC/DC Power Module*”数据手册（DS20005571）了解关于 RTOP 和 RBOTTOM 电阻值的更多详细信息。RBOTTOM 电阻典型值以 10k RTOP 固定值计算得出，如表 1 所示。

图 1：　MIC45116 电源模块框图

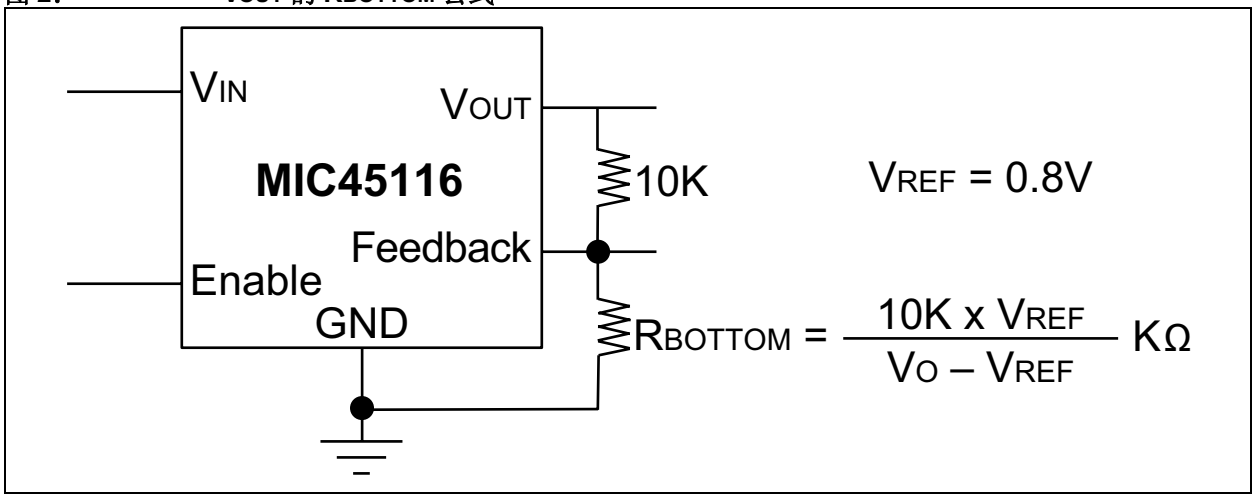


下表 1 列出了针对 1.8V、2.5V、3.3V 和 5.0V 电压的 MIC45116 RBOTTOM 典型值：

表 1：　典型 VOUT 电压的 RBOTTOM 值

VOUT	VIN	RTOP	RBOTTOM
1.8V	5-0V	10K	8.06K
2.5V	5-20V	10K	4.75K
3.3V	5-20V	10K	3.24K
5.0V	7-20V	10K	1.91K

图 2：　　VOUT 的 RBOTTOM 公式



上电序列

PIC16F18344 器件以 5.0V 和 4 MIPS 运行 （使用内部 RC 时钟），用于控制上电序列。上电序列通过以下两个步骤启动：

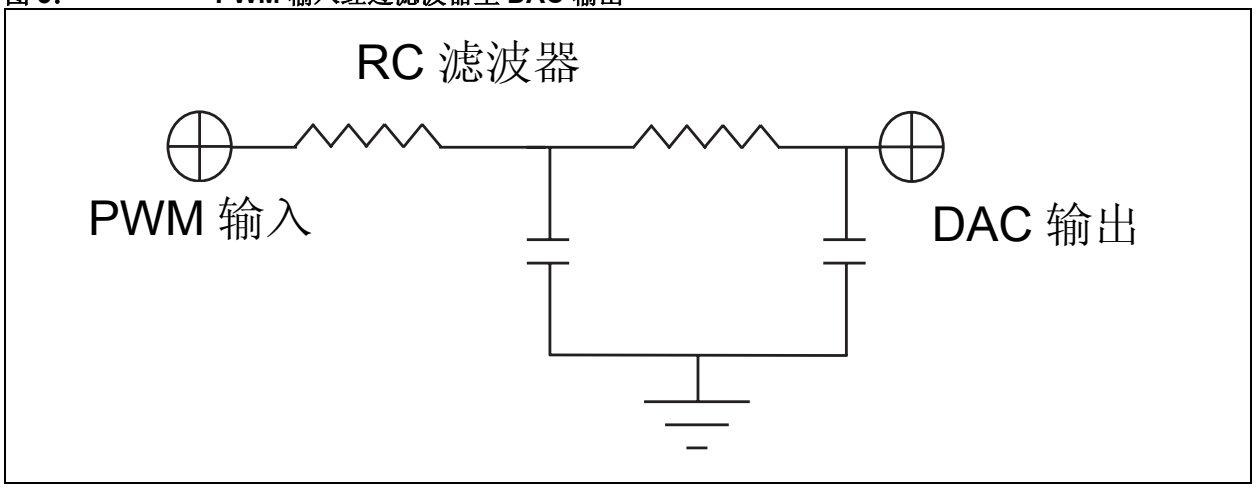
1. 使用 I^2C 接口的串行命令。
2. 按压按钮开关 S1。

每个PM按设定的时间间隔（从零到16,393 ms（16.4s））进行排序，精度为 1 ms。例如，PM1 可在启动命令之后 10 ms 启动，然后在 25 ms 启动 PM2，在 200 ms 启动 PM4，最后在 1000 ms 启动 PM3。每个 PM 都有相应的开启时间值，为固件中的一个 14 位无符号整数值。该值与每 ms 递增的定时器值进行比较。如果定时器值与 PM 的开启时间值发生匹配，则打开相应的 PM。开 / 关时序可由用户选择，并保存在 PIC16 器件的闪存中。也可使用串行 I^2C 图形用户界面 （Graphical User Interface，GUI）启动 / 停止开 / 关序列。

反馈

当打开 PM 时，根据预先指定的电阻值自动生成反馈电压。通过使用 PIC16 单片机的 PWM 输出创建额外的反馈电压。PWM 输出然后通过 RC 滤波器发送。滤波器的输出产生一个 DAC 电压，该电压被发送到 PM 的 Feedback 引脚 （图 3）。使用 PIC16 单片机上的 10 位 A/D 转换器 （ADC）监视 PM 的输出是否在设定电压的 ±1 最低有效位（LSb）内。每个 PM 电压通过对 16 个读数取平均值得出，是一个 14 位值。仅该值的最高 8 位用于表示每个 PM 的 VOUT 电压值。ADC 的参考电压是 VDD 或 5.0V。例如，如果 PM 输出电压为 2.5V，则测量精度将为（2.5/5.0）/256 = 2 mV。监视输出电压是否有故障（即，输出电压低于每个 VOUT 的最小输出限值或高于最大输出限值）。如果发生故障，PM 按照掉电序列自动关闭。

图 3：　　PWM 输入经过滤波器至 DAC 输出



掉电序列

PIC16 MCU 还控制四个电源的可编程掉电序列。掉电序列通过以下步骤启动：

1. 来自 I²C 总线的串行命令。
2. 按压按钮开关 S1。
3. PM 或输入电压出现任何故障情况。

每个 PM 按设定的时间间隔（从零到 16,393 ms（16.4s））顺序掉电，精度为 1 ms。例如，PM4 可在停止命令之后 20 ms 关闭，然后在 25 ms 关闭 PM2，在 200 ms 关闭 PM3，最后在 1000 ms 关闭 PM1。每个 PM 都有一个相应的关闭时间值，是一个 14 位无符号整数。该值与开启时间值无关。该值与每 ms 递增的 16 计时器值进行比较。如果两个值相等，那么关闭相应的 PM。关闭时间由用户选择并烧写在闪存中。在出现故障情况掉电时，将根据用户指定的重试次数自动启动新的上电序列。通常，用户可指定重试两次或三次。当所有重试均返回故障时，系统将关闭并发出故障情况信号。用户可使用 I²C GUI 确定哪个 PM 或输入电压导致故障。用户必须采取适当的纠正措施来消除故障情况，并使用 GUI 复位系统，然后重试成功的上电序列。

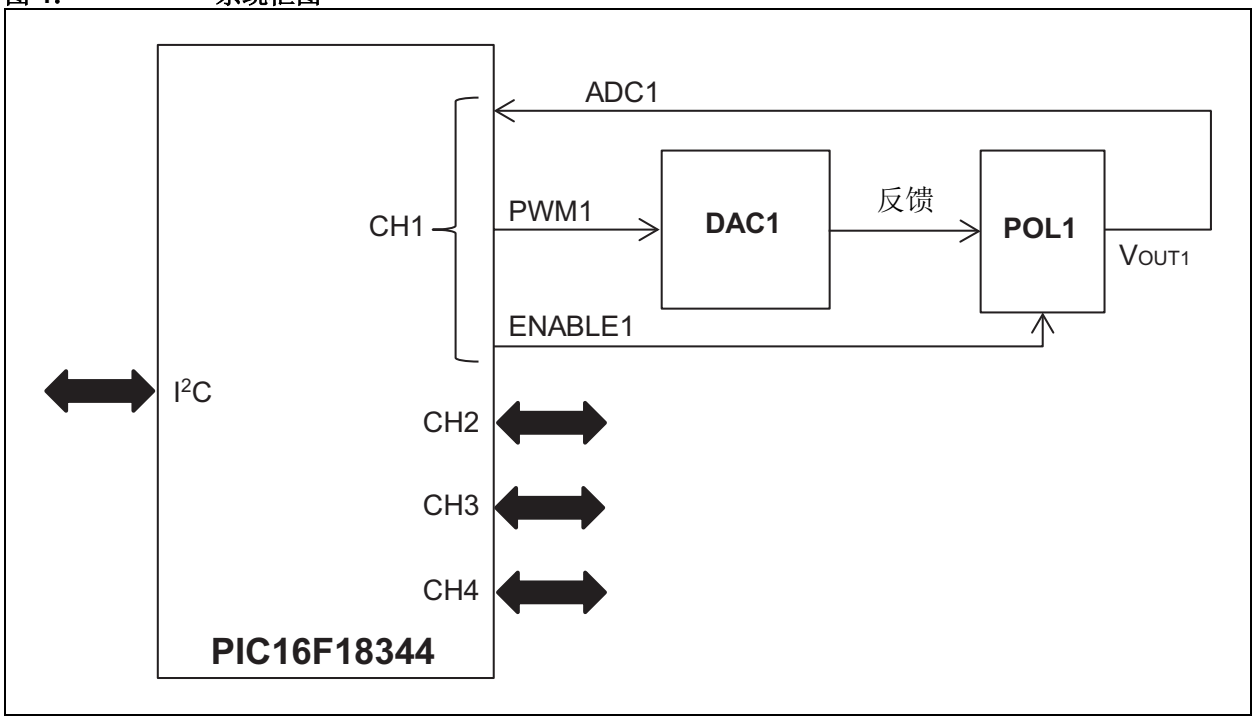
MCU 需求

MCU 需求分为以下四类：

1. I/O 引脚用于使能 / 禁止 PM。
2. ADC 输入用于采样 PM VOUT 电压和输入电压。
3. PWM 输出用于产生 DAC 输出作为反馈电压。
4. 使用 I²C 进行通信。

由于在此设计中使用 4 通道 PM，因此需要至少 4 个 I/O 线实现使能 / 禁止功能。另外，还需要四个 ADC 通道、加上一个通道用于输入电压、4 个 PWM 输出、两条线用于 I²C、MCLR、SW1（KEY 输入）、VDD、VSS 和编程引脚。总共需要 20 个引脚。因此，本设计选择使用 PIC16F18344 MCU。图 4 所示为系统框图。

图 4：　　系统框图



MCU 由 5.0V 供电，内部 16 MHz RC 时钟用于以 4 MIPS 运行 CPU。可对硬件 / 固件进行修改以适应最多 10 个 PM。如果需要更多 PM， I/O 数将增加且必须选择较大的 PIC16F1XXX 器件。如果需要较少 PM，则应用可选择较小的 PIC16F1XXX 器件。

反馈电压要求也可以调整。如果用户只想要使用外部电阻控制 PM，则不需要 DAC 电压。反馈电压使用外部电阻值预先设置，并消除了 PWM 需求以及用于 DAC 的滤波器。用于驱动 PWM 和 DAC 的软件也将免去。

电压限值

每个PM都有自己的正常过压/欠压限值。对于通过DAC提供的反馈电压， PM 还具有容限过压 / 欠压限值。

正常过压 / 欠压限值

按照规定，这些限值是每个 PM 的绝对限值。实时监视每个 PM 的输出电压，如果输出电压在任何时候低于或高于这些限值，则发出故障信号并顺序关闭电源。发生电源故障后会重试。在自动重试用户指定的一定次数后，系统将关闭，直到用户通过 I^2C GUI 使用串行命令清零系统并修复硬件 / 软件错误。例如，对于 5V PM，用户可选择4.5V作为欠压限值，而5.5V作为过压限值。

容限过压 / 欠压限值

每个 PM 都有一个用户定义的容限过压 / 欠压限值。这些限值是保存在固件中的 8 位值，并对应于相应 PM 的 PWM 占空比值。用户可使用 GUI 修改这些值。DAC 值可在容限过压 / 欠压限值的范围内变化。

注：	微调电压不受PIC16单片机监视。只有PWM占空比值与用户设定的限值进行比较。该值定义为一个 8 位值。

I^2C 通信协议和命令

在 MCU 上实现 I^2C 从器件接口以与外部 I^2C GUI 进行串行通信。I^2C 接口的命令结构在**附录 A：“串行命令定义和结构”**中指定。演示板上提供了一个 MCP2221 I^2C 转迷你 USB 接口芯片。用户可在其设计中实现该接口或其他 I^2C 接口。

I^2C 寻址

由于只有一个器件连接到该接口，因此为定序器选择固定从器件地址 `0x14`。可根据需要在固件中更改该地址。目前不支持 PMBus™ 协议。

固件实现

使用 Microchip 代码配置器 （MCC）生成外设代码

所有用于 I/O、Timer1、ADC、PWM、闪存和 I²C 外设的固件均已使用免费的 MPLAB® 代码配置器（MPLAB® Code Configurator，MCC）软件创建和初始化。系统设置也使用 MCC 软件。使用 MPLAB® X IDE 开发整个固件。强烈建议使用 MCC 软件，因为所创建的固件已经过测试并且有详细的文档记录。每个外设的初始化和用户程序都是模块化的，如果用户想要使用另一个更大或更小的 PIC16FXXXX 器件，重新创建外设程序将非常简单，只需要很少的时间。如果使用 MCC，所有外设的代码将自动生成并放置在 MPLAB X IDE 项目中。作为例子，**附录 B：“使用 MPLAB® 代码配置器 （MCC）将 Timer1 功能添加到应用程序”**演示了使用 MCC 生成 Timer1 代码的分步过程。

I²C 从器件中断固件

I²C 从器件中断固件已使用 MCC 创建。**附录 C：“修改 MCC 创建的 I²C 从器件中断文件实现应用串行接口”**详细说明了使用 MCC 的设置以及需要进行的修改。MCC 固件有一个串行 EEPROM 的内置示例。修改该固件以包含电源定序器程序中使用的从器件 I²C。该方法很快速而且非常容易使用，无需任何知识储备或了解关于 I²C 协议或功能的设置细节。

用户应用程序

所有代码均以状态机格式编写。PM 模块的代码位于文件 `app.c` 中。ADC、闪存、KEY 和系统的代码位于相应的 `appAdc.c`、`appFlash.c`、`appKey.c` 和 `appSys.c` 文件中。定义和所使用的变量在 `app.h` 中定义。整个应用程序代码基于状态机，即将每个任务定义为一系列状态。例如，ADC、每个 PM 模块、闪存和整个系统都有一个状态机。 `*main.c` 程序非常简单，如例 1 所示。

例 1： 代码片段

```
void  main(void) {

      // 一开始初始化所有状态机
    SYSTEM_Initialize();
    APP_FLASH_Initialize();
    APP_M0_Initialize();
    APP_M1_Initialize();
    APP_M2_Initialize();
    APP_M3_Initialize();
    APP_M4_Initialize();
    APP_KEY_Initialize();
    APP_ADC_Initialize();
    APP_SYS_Initialize();
    // 在无限循环中，每个状态机一个接一个运行
      while (1)
      {
            APP_M0_Tasks();
            for (MI=1;MI < AllModules;MI++)
                APP_MX_Tasks();
            APP_KEY_Tasks();
            APP_ADC_Tasks();
            APP_SYS_Tasks();
            APP_FLASH_Tasks();
      }
}
```

每个初始化程序初始化模块、系统或外设。另一方面，每个任务是一个状态机，负责运行应用程序 PM 模块 0 到 4、 ADC、 KEY、闪存和整个系统的任务。

注： 请注意，模块 1 至 4 对应于 PM 1 至 4。模块 0 指定为运行监视输入电压的状态机。

每个模块的参数结构

每个电源模块都分配了一组参数，在 app.h 中定义。

参数包括：

- State——它定义每个模块在特定时刻的状态。

  8 个状态是：

  - Init——模块的初始状态
  - On——模块处于开启状态
  - Off——模块处于关闭状态
  - Start——模块开启
  - Starting——开启时会考虑模块上升时间
  - Stop——模块关闭
  - CheckADC——更新相应 PM 模块的 ADC 值
  - CheckError——检查输出电压是否出现过压 / 欠压故障

注： 附录 E 中的图 E-1 给出了模块状态机图。

- NormalUVL、NormalOVL、MarginUVL 和 MarginOVL——OverVoltageLimit 和 UnderVoltageLimit——针对正常电压和容限电压。使用和保存这些变量进行错误检查。它们是 8 位无符号值，可由用户修改。
- ADC——16 位无符号值，用于保存 10 位 ADC 值的 16 个计数。
- PWMValue——特定于模块的 PWM 中使用的 10 位 PWM 占空比值。
- ADCValue——取 10 位 ADC 值的 16 个计数的 8 位平均值。该值与限制电压值作比较并检查是否出错。
- anChannel——分配给模块的 ADC 通道编号。
- OnTime 和 OffTime——每个模块的 14 位启动和停止时间值，单位为 ms。
- TurnOn——用于指示模块是否开启的位值。

所有参数都以类型定义结构进行定义，这个结构的数组定义如下：appmData[5]。appmData[1] 用于模块 1，而 appmData[4] 用于模块 4。输入电压 VIN 的参数分配在 appmData[0] 或数组的第一个元素中。请注意，可能只有一些参数与 VIN 相关。例 2 给出了模块 3 的 PWM 参数。

例 2： PWM 占空比参数

```
appmData[3].PWMValue——对应于模块 3 使用的 PWM 的 10 位占空比值。
```

除了创建这个结构，每个模块都有一个初始化程序。任务程序对每个 PM 模块 1 至 4 通用，因此通用的 APP_MX_Tasks() 程序与对应于相应 PM 模块的模块索引（Module Index，MI）参数一起使用。VIN 分配有自己的程序：APP_M0_Tasks();。

通过为每个模块创建该结构和相关的功能，用户可以很容易地增加或减少应用中使用的模块的数量。如果需要三个模块，那么元素数量可从四个减少一个到三个。也可删除一个初始化和一个任务功能。但是，如果模块数量需要增加一个到五个，那么数组中的元素数也需要增加（从四个增加到五个）。需要复制 / 粘贴一个额外的初始化程序，并调整相应的参数以反映新的模块 5。**附录 D：“将额外的模块添加到现有代码”**举例说明如何将一个额外的模块 4 添加到应用程序中。

ADC 程序和电压测量

ADC 程序基本上贯彻始终并采样每个模块 0 至 4 的电压。如前面所述，模块 0 对应于输入电压，始终对其进行故障监视。输入电压故障将导致发生关闭事件。不会重新尝试。10 位 ADC 程序对每个电压采样 16 次，然后使用 8 位平均值与相应的过压 / 欠压限值作比较，检查是否存在错误。在使用的硬件中，参考电压是 5.0V 或系统的 VDD。采样和转换 1.8V、2.5V 和 3.3V 时，5.0V 参考电压工作良好。但是，对于 5.0V 模块和输入电压，需要一个电阻分压器将整个电压范围调整在 5.0V 参考电压范围内。5.0V 模块的电阻分压器系数为 0.55，输入电压分压系数为 0.239。用户在计算过压和欠压限值时必须使用该值并在头文件中进行相应定义。如果用户决定使用本应用笔记之外的值，则尤其需要。

ADC 状态机状态

- Init——初始化 ADC 转换器
- Sample——在该状态下，在开始转换之前 ADC 通道被分配 1 ms 的采样时间。
- Convert——发出转换命令并启动 ADC 转换
- Done——检查转换是否完成；如果是，则加上转换后的值，并且如果所有 16 个采样均已完成，则分配相应的模块状态进行电压更新和电压检查。
- Next——在该状态下，将下一个模块分配给 ADC 并重复整个过程

ADC 状态机始终运行。仅在模块开启或处于打开状态时才检查指定的模块电压。否则，只检查输入电压。模块开启后，则始终检查。

KEY 状态机状态

KEY 状态机具有以下状态：

- Init——初始化状态机
- High——当按键处于默认高电平状态时，执行按键按压检查 （0V）。
- Low——当按键处于按下状态时，执行按钮释放检查 （5V）。
- Debounce——在此状态下，执行 20 ms 延时，然后检查是高电平状态还是低电平状态

KEY 状态机确保按键开关 S1 正常工作。如果系统关闭，按压按键将打开系统。如果系统打开，按压按键将关闭系统。

SYS 状态机

SYS 状态机管理系统的整体功能。特别是在启动期间管理故障模式。如果发生启动故障，那么执行自动重启，重启次数由用户在重试参数中定义。`APP_SYS_Tasks()` 程序确保执行所有重试尝试。如果所有重试尝试都已执行，系统将关闭并产生故障条件。然后，用户必须使用 I^2C 接口上的串行命令或 GUI 上的按钮复位系统。任何故障机制必须修复以防止启动期间发生另一个故障。直到复位命令执行或掉电复位完成后，系统才会重新启动。

SYS 状态有：

- Init——初始化系统状态
- ON——系统打开时的状态
- OFF——系统关闭时的状态
- Starting ——系统正在启动时的状态
- Stopping——系统正在关闭时的状态
- Fault——发生系统故障时的状态

闪存状态机

默认参数保存在 PIC16F18344 的闪存中。闪存 PM 存储单元起始于 `0x1f80` 且顺序保存每个模块的参数。每个模块的默认参数是：

- 启动时间
- 停止时间
- 正常欠压限值
- 正常过压限值
- 容限欠压限值
- 容限过压限值
- PWM 占空比

除了这 28 个参数 （4 x 7）外，还保存系统参数。

- V_{IN} 欠压限值
- V_{IN} 过压限值

重试次数

用户可使用 GUI 或串行命令修改这些参数的值。用户必须使用 GUI 上的 **Burn Flash**（烧写闪存）按钮或串行命令：`0x10`, `0xAB` 将这些值烧写到闪存中。烧写后，这些值变成新的默认值。文件 `flash.as` 定义闪存 PM 中的所有默认值和存储单元。

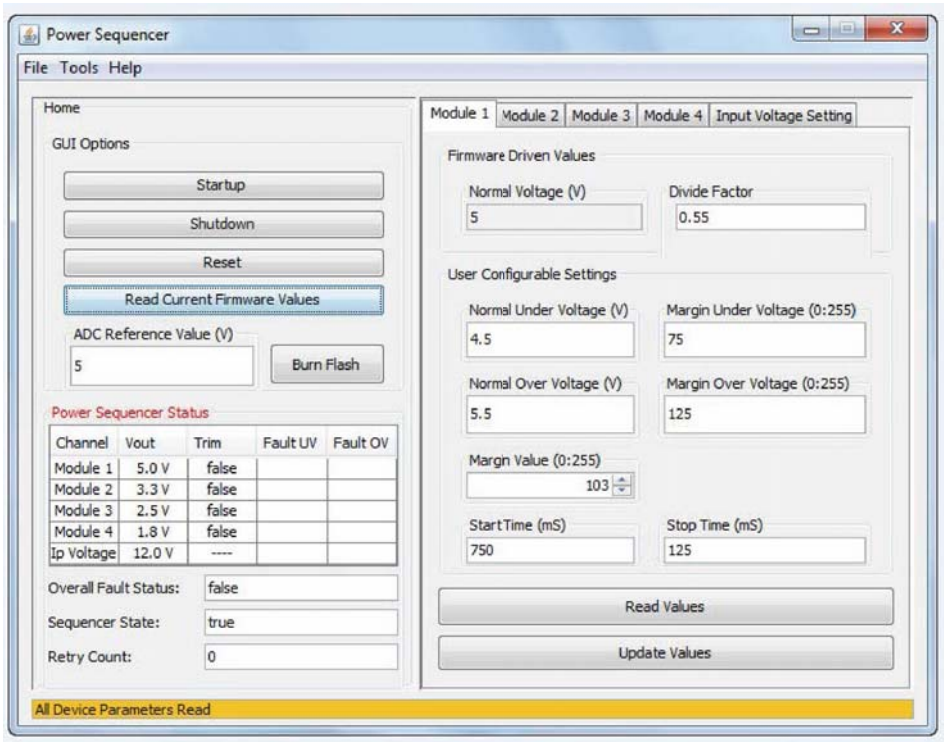
电源定序器 GUI

用户可使用所设计的电源定序器 GUI 进行以下操作：

1. 输入电源定序器应用程序的相关数据。
2. 监视电源定序器应用程序的相关数据。
3. 控制电源定序器应用程序。

图形用户界面 （GUI）如图 5 所示。

图 5：　　图形用户界面 （GUI）



主窗口左侧为系统选项，右侧选项卡为模块选项。

在系统选项中，用户可启动、停止、复位和读取当前固件值。状态窗口允许用户定义与模块索引对应的 VOUT。这些值可由用户修改并将在关闭 GUI 时保存。用户还可以输入 ADC 参考值，本应用笔记中设置为 5.0V。最后，用户可单击**Burn Flash**按钮将更新的模块设置烧写到闪存程序存储器中。

在每个模块选项卡下，用户还可设置或读取每个模块的现有值。模块 1 是 5V 模块，用户可为该模块设置正常 / 容限过压 / 欠压限值以及开始时间和停止时间（以 ms 计）。另外，还可在该选项卡中编辑和输入电压分压系数。 **Read Values**（读取值）按钮读取 RAM 中的现有值，而 **Update Values**（更新值）按钮将新值写入 RAM 中。如果用户想要永久使用这些值，必须使用 **Burn Flash** 按钮将值烧写到闪存中。单击相应的选项卡，用户可修改 / 读取 / 更新所有模块的值。

发生故障时， GUI 不自动更新，因为 I^2C 在 PIC16 中以从器件模式实现。用户必须单击**Read Current Firmware Values** （读取当前固件值）按钮获取状态更新并识别发生故障的模块。

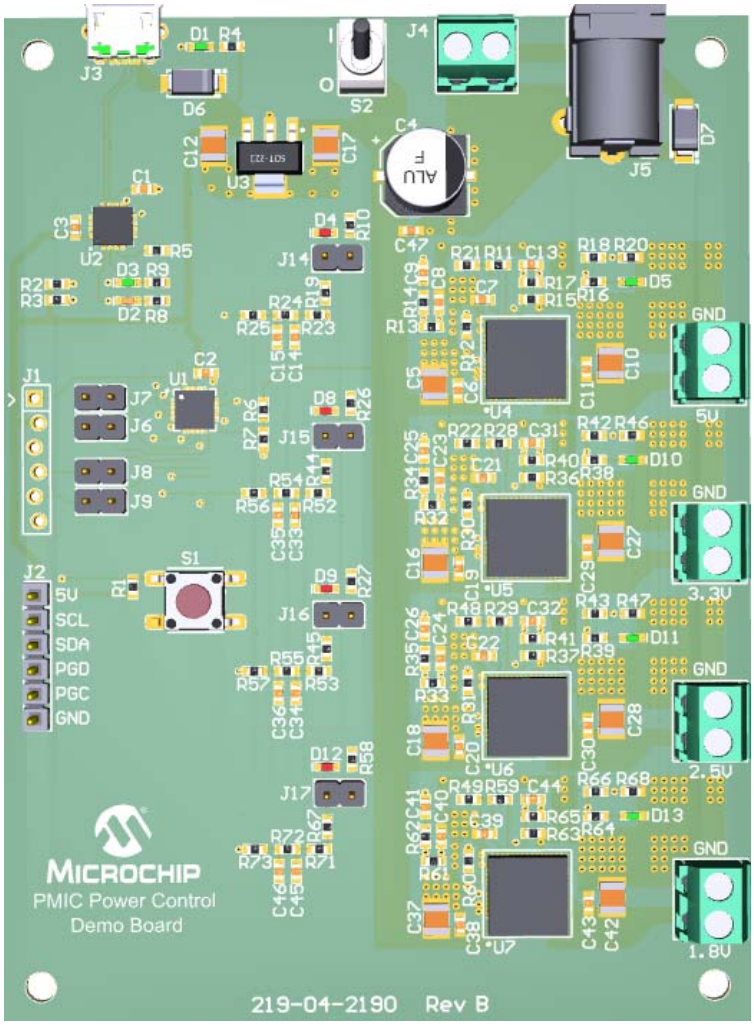
在每个模块选项卡中，可使用位于 DAC 值框一端的上 / 下箭头递增 / 递减 DAC 值。如果模块开启，递增或递减该值，那么将读取和更新输出电压。可能需要执行多个递增或递减才能看出电压变化。该功能允许用户在系统测试期间当输出电压达到其限值时增加或减小电压。这称为电压限值测试并允许用户在一个或多个输出电压达到其过压 / 欠压限值时测试整个系统。

电压值显示为实际电压值（3.3V 或 2.5V）。 DAC 值和容限值显示为 8 位值，介于 0 和 255 之间。

结论

为方便用户使用PIC16FXXXX器件实现电源定序器设计应用，专门撰写了本应用笔记。用户可轻松修改本应用笔记在自己的设计应用中控制四个电压模块。此外，用户还可向其应用添加更多电源模块或除去模块实现更小的应用。硬件和固件均以模块化格式创建，可轻松实现这些目标。下面的图 6 描绘了完整的电路板。**附录 F：“电源定序器原理图”**显示了电路板的原理图。

图 6：　　电源控制演示板



附录 A： 串行命令定义和结构

A.1 I^2C 串行命令格式

A.1.1 符号

- S = 启动
- P = 停止
- A = 应答
- N = N
- W = 读 / 写位

A.2 写命令

S< 从器件地址 > W A < 命令 > A < 参数 > A <ValueL> A <ValueH> A P

注：	一些命令仅包含一个参数且不发送 ValueL 或 ValueH 参数。通常，写命令在读命令前面。

A.3 读命令

S < 从器件地址 > R A <ValueL> A <ValueH> N P

A.3.1 命令 0x10：烧写闪存

写入 0x10 后跟 0xAB 将启动烧写闪存操作。用户更新了所有模块的所有参数（使用 GUI 或本节中提到的 I^2C 串行指令）后，烧写闪存操作将这些值永久地烧写到 PIC16F18344 器件的闪存中。随后的复位或上电复位将从闪存中检索值。建议用户在更新所有模块参数后使用此操作。

A.3.2 命令 0x20：设置模块启动时间值

写入 0x20 命令后跟模块编号将为指定模块设置新的启动时间。紧跟该命令和参数的两个数据字节将包含所需的启动时间设置 （以 ms 为单位）。由于启动时间仅为 14 位，所以该值将限制在 1 到 16393 ms 内。

A.3.3 命令 0x24：读取模块启动时间值

写入 0x24 命令后跟模块编号将读取指定模块的启动时间。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。紧跟命令和参数的两个数据字节将包含所需的启动时间设置 （以 ms 为单位）。由于启动时间仅为 14 位，所以该值将限制在 1 到 16393 ms 内。

A.3.4 命令 0x28：设置模块停止时间值

写入 0x24 命令后跟模块编号将为指定模块设置新的停止时间。紧跟该命令和参数的两个数据字节将包含所需的停止时间设置 （以 ms 为单位）。由于停止时间仅为 14 位，所以该值将限制在 1 到 16393 ms 内。

A.3.5 命令 0x2C：读取模块停止时间值

写入 0x2C 命令后跟模块编号将读取指定模块的停止时间。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。紧跟该命令和参数的两个数据字节将包含所需的停止时间设置 （以 ms 为单位）。由于停止时间仅为 14 位，所以该值将限制在 1 到 16393 ms 内。

A.3.6 命令 0x30：设置 DAC 输出电压裕量

写入 0x30 命令，后跟模块编号作为参数，将发送参数指定的 PWM/DAC 通道的新裕量设置。紧跟该命令和参数的两个数据字节将包含所需的 PWM/DAC 输出设置。由于 PWM/DAC 仅为 8 位，因此 ValueH 为零。

A.3.7 命令 0x38：读取 DAC 容限电压

写入 0x38 命令，后跟模块编号，将读取当前的 DAC 值。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。读操作将返回 DAC 的高字节和低字节值；但是，由于该值仅为 8 位，因此高字节值将为 0，而低字节将包含 DAC 值。

A.3.8 命令 0x34：读取系统重试计数

写入 0x34 命令，后跟 0x00 作为参数，将读取为系统指定的预设重试计数参数。紧跟该命令和参数的两个数据字节将包含重试计数。由于该值通常小于 10，因此 ValueH 被忽略。

A.3.9 命令 0x3C：设置系统重试计数

写入 0x3C 命令，后跟 0x00 作为参数，将设置为系统指定的预设重试计数参数。紧跟该命令和参数的两个数据字节将包含重试计数。由于该值通常小于 10，因此 ValueH 为零。

A.3.10 命令 0x40：设置正常欠压限值

写入 0x40 命令，后跟模块编号作为参数，将发送该通道的新正常欠压限值。紧跟该命令和参数的两个数据字节将包含该 ADC 通道所需的正常欠压限值，并应用于该电源。该值为 8 位，因此 ValueH 为零。

A.3.11 命令 0x50：设置正常过压限值

写入 0x50 命令，后跟模块编号作为参数，将发送该通道的新正常过压限值。紧跟该命令和参数的两个数据字节将包含该ADC 通道所需的正常过压限值，并应用于该电源。该值为 8 位，因此 ValueH 为零。

A.3.12 命令 0x60：设置容限欠压限值

写入 0x60，后跟模块编号作为参数，将发送该通道的新容限欠压限值。紧跟该命令和参数的两个数据字节将包含该 ADC 通道所需的容限欠压限值，并应用于该电源。该值为 8 位，因此 ValueH 为零。

A.3.13 命令 0x70：设置容限过压限值

写入 0x70，后跟模块编号作为参数，将发送该通道的新容限过压限值。紧跟该命令和参数的两个数据字节将包含该 ADC 通道所需的容限过压限值，并应用于该电源。该值为 8 位，因此 ValueH 为零。

A.3.14 命令 0x80：读取正常欠压限值

写入 0x80，后跟模块编号作为参数，将启动读取该 ADC 通道的正常欠压限值操作。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。读取的两个数据字节将包含该模块的当前正常欠压限值。由于值仅为 8 位，因此低字节将包含该值，而高字节将为零。该命令将允许主器件时钟延长几微秒。

A.3.15 命令 0x90：读取正常过压限值

写入 0x90，后跟模块编号作为参数，将启动读取该 ADC 通道的正常过压限值操作。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。读取的两个数据字节将包含该 ADC 通道的当前正常过压限值。由于值仅为 8 位，因此低字节将包含该值，而高字节将为零。该命令将允许主器件时钟延长几微秒。

A.3.16 命令 0xA0：读取容限欠压限值

写入 0xA0，后跟模块编号作为参数，将启动读取该 ADC 通道的容限欠压限值操作。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。读取的两个数据字节将包含该 ADC 通道的预设容限欠压限值。由于值仅为 8 位，因此低字节将包含该值，而高字节将为零。该命令将允许主器件时钟延长几微秒。

A.3.17 命令 0xB0：读取容限过压限值

写入 0xB0，后跟模块编号作为参数，将启动读取该 ADC 通道的容限过压限值操作。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。读取的两个数据字节将包含该 ADC 通道的当前容限过压限值。由于值仅为 8 位，因此低字节将包含该值，而高字节将为零。该命令将允许主器件时钟延长几微秒。

A.3.18 命令 0xC0：读取电源定序器状态

写入 0xC0，后跟所需的 STATUS 寄存器作为参数，将启动读取所需 STATUS 寄存器操作。然后发送重新启动命令，后跟读周期和器件从地址以读取 STATUS 寄存器的两个数据字节。 STATUS 寄存器如下表 A-1 所示。

表 A-1： STATUS 寄存器

寄存器编号	值
0	电源欠压错误位，每个 PM 1 个
1	电源过压错误位，每个 PM 1 个
2	Fault<15>、 Seq.State<14> 和 Margin Enable<0:9>
3	自上个电源 ADC 通道以来的系统序列重试次数

A.3.19 命令 0xC8：清零状态错误标志和重试次数

写入 0xC8，后跟 0xCE 将发送清零命令以清零上面执行的 0xC0 命令中提及的所有状态位。STATUS 寄存器 0 到 3 将清零。容限使能位不会清零。

A.3.20 命令 0xD0：读取模块模拟电压

写入 0xD0，后跟模块编号作为参数，将启动读取模块电压读数操作。然后发送重新启动命令，后跟读周期和器件从地址以读取两个数据字节。读取的两个数据字节将包含该模块电压的当前电压。由于值仅为 8 位，因此低字节将包含该值，而高字节将为零。该命令将允许主器件时钟延长几微秒。

A.3.21 命令 0xE0：上电

写入 0xE0，后跟 0xEF 将启动上电序列。如果电源定序器已打开，将忽略该命令。该命令通常用于在发生过压 / 欠压故障触发的反序列事件后重新启动模块。

A.3.22 命令 0xF0：掉电

写入 0xF0，后跟 0xDF 将启动掉电序列。如果控制器当前处于启动或掉电状态，则该命令将被忽略。如果控制器处于开启状态，则该命令将启动掉电序列。

表 A-2：　　命令汇总

命令	参数	写入数据 1	写入数据 2	读取数据 1	读取数据 2	说明
0x10	0xAB	—	—	—	—	烧写闪存
0x20	DAC 编号	ValueL	ValueH			设置启动时间
0x24	DAC 编号	—	—	ValueL	ValueH	读取启动时间
0x28	DAC 编号	ValueL	ValueH	—	—	设置停止时间
0x2C	DAC 编号	—	—	ValueL	ValueH	读取停止时间
0x30	DAC 编号	ValueL	ValueH	—	—	设置 DAC 输出
0x38	DAC 编号	—	—	ValueL	ValueH	读取 DAC 值
0x34	0x00	—	—	ValueL	ValueH	读取重试计数
0x3C	0x00	ValueL	ValueH	—	—	设置重试计数
0x40	通道编号	ValueL	ValueH	—	—	设置正常欠压限值
0x50	通道编号	ValueL	ValueH	—	—	设置正常过压限值
0x60	通道编号	ValueL	ValueH	—	—	设置容限欠压限值
0x70	通道编号	ValueL	ValueH	—	—	设置容限过压限值
0x80	通道编号	—	—	ValueL	ValueH	读取正常欠压限值
0x90	通道编号	—	—	ValueL	ValueH	读取正常过压限值
0xA0	通道编号	—	—	ValueL	ValueH	读取容限欠压限值
0xB0	通道编号	—	—	ValueL	ValueH	读取容限过压限值
0xC0	STATUS 寄存器编号	—	—	Status L	Status H	读取电源定序器状态
0xC8	0xCE	—	—	—	—	清零 STATUS 寄存器
0xD0	通道编号	—	—	ValueL	ValueH	读取模拟电压
0xE0	0xEF	—	—	—	—	上电命令
0xF0	0xDF	—	—	—	—	掉电命令

表 A-3：　　STATUS 寄存器位定义

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Status 0 H	VIN UV	—	—	—	—	—	Mod10 UV	Mod9 UV
Status 0 L	Mod8 UV	Mod7 UV	Mod6 UV	Mod5 UV	Mod4 UV	Mod3 UV	Mod2 UV	Mod1 UV
Status 1 H	VIN OV	—	—	—	—	—	Mod10 OV	Mod9 OV
Status 1 L	Mod8 OV	Mod7 OV	Mod6 OV	Mod5 OV	Mod4 OV	Mod3 OV	Mod2 OV	Mod1 OV
Status 2 H	Fault On	Seq State	0	0	0	0	Mod10 Mrg En	Mod9 Mrg En
Status 2 L	Mod8 Mrg En	Mod7 Mrg En	Mod6 Mrg En	Mod5 Mrg En	Mod4 Mrg En	Mod3 Mrg En	Mod2 Mrg En	Mod1 Mrg En
Status 3 H	0	0	0	0	0	0	0	0
Status 3 L	系统重试计数器值							

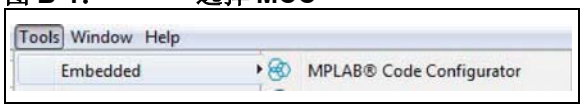
图注：　Modx = 模块 x
MRG En = 容限使能
OV = 过压
UV = 欠压

附录 B： 使用 MPLAB® 代码配置器（MCC）将 TIMER1 功能添加到应用程序

要使用 MCC 将 Timer1 功能添加到应用程序，请按照下列步骤操作：

1. 在 MPLAB® X IDE 中打开项目，从主菜单选择 Tools → Microchip Embedded → MPLAB Code Configurator（工具 → Microchip 已安装工具 → MPLAB 代码配置器），如图 B-1 中所示。

图 B-1： 选择 MCC

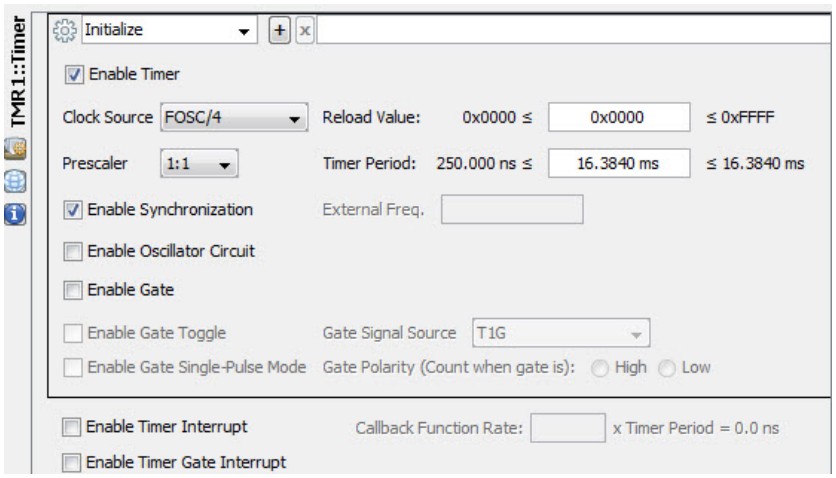


2. MCC 打开后，从 Device Resources（器件资源）菜单中，选择 Timer1 将其加入 Project Resources（项目资源）窗口，如图 B-2 所示。

图 B-2： 在 PROJECT RESOURCES 窗口中打开 TIMER1

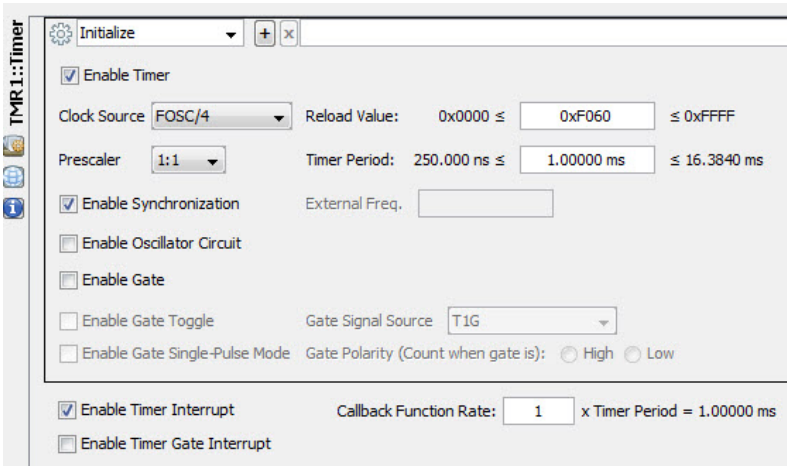
3. 单击 TMR1 编辑和输入值以配置 Timer1，如图 B-3 所示。

图 B-3： TMR1 配置



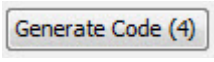
4. 要使用 16 MHz 内部时钟将 TMR1 配置为 1 ms 中断，请将“Time Period”（时间周期）编辑为 1.0 ms，勾选 “Enable Timer Interrupt”（允许定时器中断）框并将回调函数速率编辑为 1，如下图 B-4 所示。

图 B-4： TMR1 中断配置



5. 单击 MCC 的 **Generate Code** （生成代码）按钮，创建 TMR1 代码。

图 B-5： GENERATE CODE 按钮



通过 MCC 使用该方法可初始化和创建 ADC、 PWM、GPIO、闪存和 I^2C 功能。

附录 C： 修改 MCC 创建的 I²C 从器件中断文件实现应用串行接口

使用 MCC 生成 I²C 从器件中断文件时，MCC 在 I2c.c 文件中为用户生成一个 EEPROM 示例。由于未使用 EEPROM 代码，本章节将阐述如何修改现有代码来适应电源定序器 I²C 接口代码。

I²C 命令结构最多可以接收四个字节的信息并发送两个字节，因此需要创建和定义缓冲器以保存该数据：

- unsigned char RcvBuf[4];
- unsigned char TrmtBuf[2];

要访问这些缓冲器，还应定义相应的指针：

- unsigned char RcvPtr=0;
- unsigned char TrmtPtr=0;

下面的示例是可能发生的情况。假定只执行两个命令：0x20（设置启动时间）和 0x24（读取启动时间）。所有其他命令遵循与此处提到的两个命令相同的结构，并且可以很容易地添加到它们各自的状态机代码中。

ProcessRcvBuf() 函数处理从主器件或GUI接收到的 I²C 数据。

例 C-1： 部分 ProcessRcvBuf() 函数

```
void  ProcessRcvBuf(void)
{
      unsigned int r;
      if (RcvPtr >= 4)
        RcvPtr = 0;
      switch (RcvBuf[0])
      {
        case 0x20:  // 启动时间初始值
            r = RcvBuf[3];
            r = r << 8;
            appmData[RcvBuf[1]].OnTime = r +
(int)RcvBuf[2];
            break;
        default:
            break;
      }
}
```

在接收到所有串行数据后，应在 I²C 回调函数中调用 ProcessRcvBuf() 函数。根据具体命令，串行接收两个或四个数据字节。调用 SetDataLength() 函数确定将串行接收的数据字节数（2 或 4）。一旦接收到相应的字节，就调用 ProcessRcvBuf() 函数并处理串行数据。例 C-2 显示了调用 ProcessRcvBuf() 函数的位置。

例 C-2： ProcessRcvBuf() 函数调用

```
case SLAVE_NORMAL_DATA:
default:
      // 主器件写入要处理的数据
      RcvBuf[RcvPtr++] = I2C_slaveWriteData;
      if (RcvPtr == 1)
      SetDataLength();
      if (RcvPtr >= DataLength)
      {RcvPtr = 0;ProcessRcvBuf();}
      break;
```

执行 ProcessRcvBuf() 操作期间会加载发送缓冲器。需要发送回复的命令会加载相应的发送缓冲器。实际的发送发生在回调函数中，如例 C-3 所示。

例 C-3： 发送代码位置

```
case  I2C_SLAVE_READ_REQUEST:
            if (TrmtPtr == 0)
                LoadTrmtBuf();
            SSP1BUF = TrmtBuf[TrmtPtr++];
            if (TrmtPtr >= 2)
                TrmtPtr = 0;
            break;
```

通过这些修改， I²C 接口将按照应用要求工作。

附录 D： 将额外的模块添加到现有代码

用户的代码已经相当模块化，所以现有的应用程序可以扩展到包含更多的模块。在本附录中，将一个额外的模块 4 添加到程序中现有的三个模块中。

每个模块有两个函数，定义如下：

- `APP_M4_Initialize()`——初始化模块 4 状态机和硬件 / 固件的函数
- `APP_MX_Task()`——所有模块通用的函数，调用时使用正确的模块索引 （MI）

另外，还定义一个数据结构：

- `appmData[4].XXXXX`——该数据结构包含一些元素

  例如：`State`、 `DACValue` 和 `PWMValue` 等是每个模块的数据元素。它们驻留在数组结构中，且该数组的第四个元素将对应模块 4。

在文件 `app.h` 中，结构已存在，但需要添加一个额外的元素。常数 “`AllModules`” 已在 `app.h` 中定义，需要将其更改为 5，如例 D-1 所示。

例 D-1： 重新定义所有模块

```
#define AllModules 5  // M1 至 M4 +
                         输入电压 = 5。
```

重新定义常数 “`AllModules`” 后，大部分参数将需要相应调整以适应添加到应用程序中的附加模块 4。以下是需要用户修改的参数。

注： 模块 0 （即该数组的元素 0）指定为输入电压。用户不需要复制/粘贴整个结构；然而，结构需要从 4 个增加一个元素至 5 个，如上所示。

模块 4 有三个硬件元素与其相关。

1. 在 MCC 初始化期间使能用户分配的信号 I/O 引脚。
2. PWM 输出用于驱动模块上的微调线路。这由用户在 MCC 初始化期间指定。
3. ADC通道用于采样和转换模块输出电压。该ADC通道由用户在 MCC 初始化期间定义。

在 MCC 初始化期间对三个硬件元素指定以下名称：

1. EN4 表示使能 I/O 引脚。
2. PWM4 表示微调电压 PWM。
3. VO4 表示 ADC 通道编号。

在用户复制和粘贴现有模块以便重新创建一个模块时需要更改这些值。

除了硬件元素外，还需要根据新模块电压为每个模块定义一些唯一固件元素。在本应用笔记中，模块 4 是 1.8V 模块，在文件 `flash.as` 中定义了自己的唯一默认值。用户可复制该文件中的现有模块参数，然后使用新的模块参数更改该值。用户可在 `flash.as` 文件中获取模块 1 的值，将其复制并粘贴到模块 3 值的下方。

在 `flash.as` 文件中，复制模块 1 定义并粘贴在模块 3 定义正下方，如例 D-2 所示。

例 D-2： 复制 / 粘贴模块 1 参数

```
M3MarginUVL   EQU 10   ;(0.2/5.0)*256

;M1 = 5V 模块的默认电压 / 时间定义
M1NormalOVL   EQU 155  ;(5.5V*0.55/5.0V)*256; 0.55 =
                         Res.Divide Factor
M1NormalUVL   EQU 127  ;(4.5V*0.55/5.0V)*256; 0.55 =
                         Res.Divide Factor
PWM1DC        EQU 100  ;
M1OnTime      EQU 250
M1OffTime     EQU 2000
M1MarginOVL   EQU 51   ;(1.2V/5.0V)* 256
M1MarginUVL   EQU 10   ;(0.2/5.0)*256
```

注： 模块 1 是 5.0V 模块，因此所有与其相关的参数均针对 5.0V 电压。

然后，用户可按照例 D-3 编辑并修改 M1 参数为 M4 以及将 1.8V 的相关值更改为 5.0V。

例 D-3： 模块 1 到模块 4 参数更改

```
;M4 = 1.8V 模块的电压 / 时间定义
M4NormalOVL   EQU 118   ;(3.8V/5.0V)*256;
M4NormalUVL   EQU 67    ;(2.8V/5.0V)*256;
PWM4DC        EQU 52    ;
M4OnTime      EQU 2000
M4OffTime     EQU 500
M4MarginOVL   EQU 51    ;(1.2V/5.0V)* 256
M4MarginUVL   EQU 10    ;(0.2/5.0)*256
```

在 app.h 文件中，新模块需要定义标称电压输出电平，如下面的例 D-4 所述。

例 D-4：　　定义 V04Nominal

```
// M4 = 1.8 V模块的标称电压
#define  VO4Nominal 93    // (1.8V/5.0V)*256
```

另外需要将 VO4Nominal 作为第 5 个元素添加到数组 VONominal[AllModules] 中，如下面的例 D-5 所示。

例 D-5：　　将 V04Nominal 添加到数组

```
unsigned  char  VONominal[AllModules] =
{0,VO1Nominal, VO2Nominal, VO3Nominal, VO4Nominal};
```

此外，flash.as 中定义的 M4 参数默认值需要保存在对闪存 PM 执行读 / 写操作期间使用的 Buf[] 数组中。每个元素都有其自己唯一的缓冲器索引（Buffer Index，BI），用户可以复制模块 1 的现有列表并针对模块 4 进行编辑，如下面的例 D-6 所示。

例 D-6：　　定义模块 4 缓冲器索引参数

```
//M4 参数的缓冲器索引（BI）
#define   M4NormalOVLBI     24
#define   M4NormalUVLBI     25
#define   PWM4DCBI          26
#define   M4OnTimeBI        27
#define   M4OffTimeBI       28
#define   M4MarginOVLBI     29
#define   M4MarginUVLBI     30
```

注：　Buf [] 数组是包含 16 位无符号值的 32 元素缓冲器。

在 app.h 文件中，需要创建 APP_M4_Initialize() 函数的原型。用户可复制现有的原型定义，然后更改指示符。

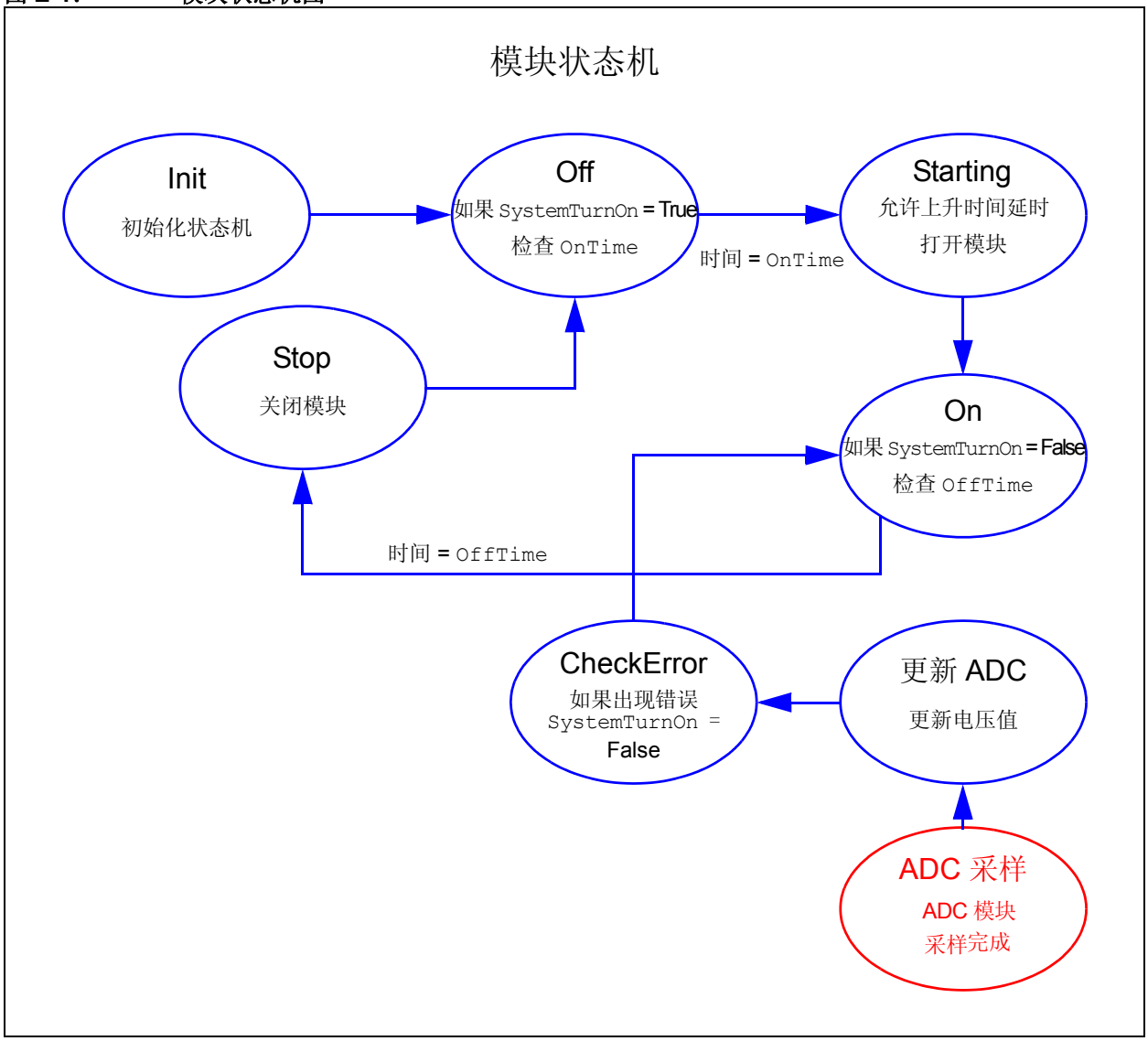
对 app.h 文件完成所有必要的更改后，用户就可以切换到 app.c 文件，对 APP_M4_Initialize()、SetUVFault()、SetOVFault()、TurnOnMod() 和 TurnOffMod() 进行更改。只需简单复制粘贴现有模块，然后对所需模块指示符进行修改即可。因此，复制并粘贴 APP_M1_Initialize() 函数。然后将函数中的 1 指示符改为 4 使其成为 APP_M4_Initialize() 函数。在 SetUVFault()、SetOVFault()、TurnOnMod() 和 TurnOffMod() 函数中，剪切并粘贴模块 1 的现有 case 语句，并将指示符从 1 改为 4，即为模块 4 创建新的 case 语句。

最后，在文件 main.c 中，用户必须设置对 APP_M4_Initialize() 程序的函数调用。

所有这些更改完成并彻底检查后，用户可编译和烧写器件，并验证操作是否正确。如果出现错误，很有可能是复制和粘贴操作期间发生输入错误。强烈建议检查新模块编辑并确保所有参数符合要求。

附录 E： 模块状态机图

图 E-1： 模块状态机图



模块状态机的状态列示如下：

- **Init**——初始化状态机并移至 OFF 状态。
- **Off**——检查 SystemTurnOn 标志。如果为 true，则检查 OnTime。当时间 = OnTime 时，打开模块并移至 Starting 状态。
- **Starting**——允许固定延时来处理电压上升时间。移至 ON 状态。
- **On**——检查 SystemTurnOn 标志。如果为 false，则检查 OffTime。当时间 = OffTime 时，移至 Stop 状态。
- **Stop**——关闭模块并移至 OFF 状态。
- **CheckADC**——在 ADC 状态机中，已采样模块电压时，模块状态自动移至 CheckADC 状态。在该状态下，更新电压值。移至 CheckError 状态。
- **CheckError**——执行过压 / 欠压检查。如果检测到错误，则 SystemTurnOn 标志设置为 false。移至 ON 状态。

附录 F： 电源定序器原理图

图 F-1： 电源定序器原理图版本 C （第 1 页，共 2 页）

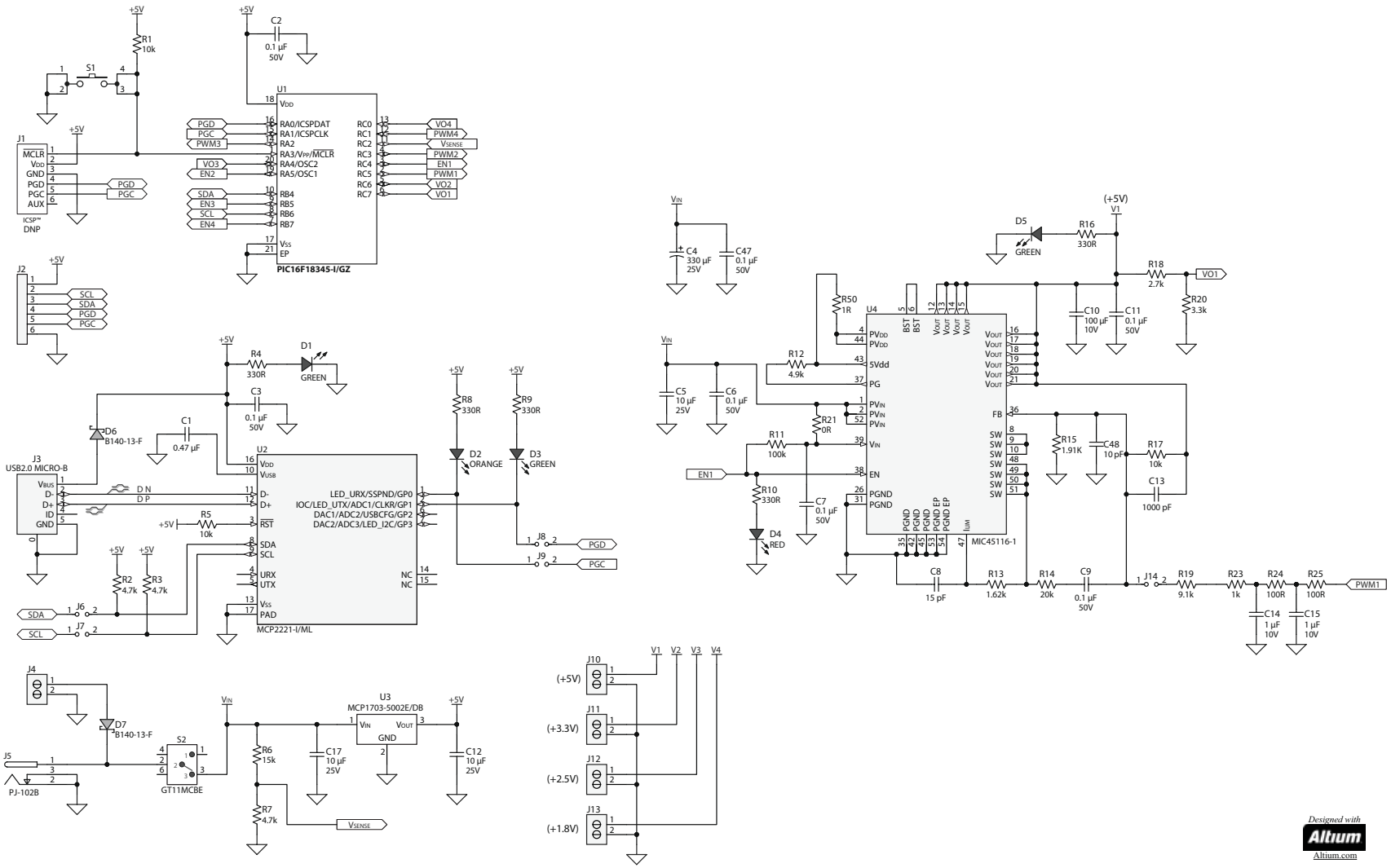
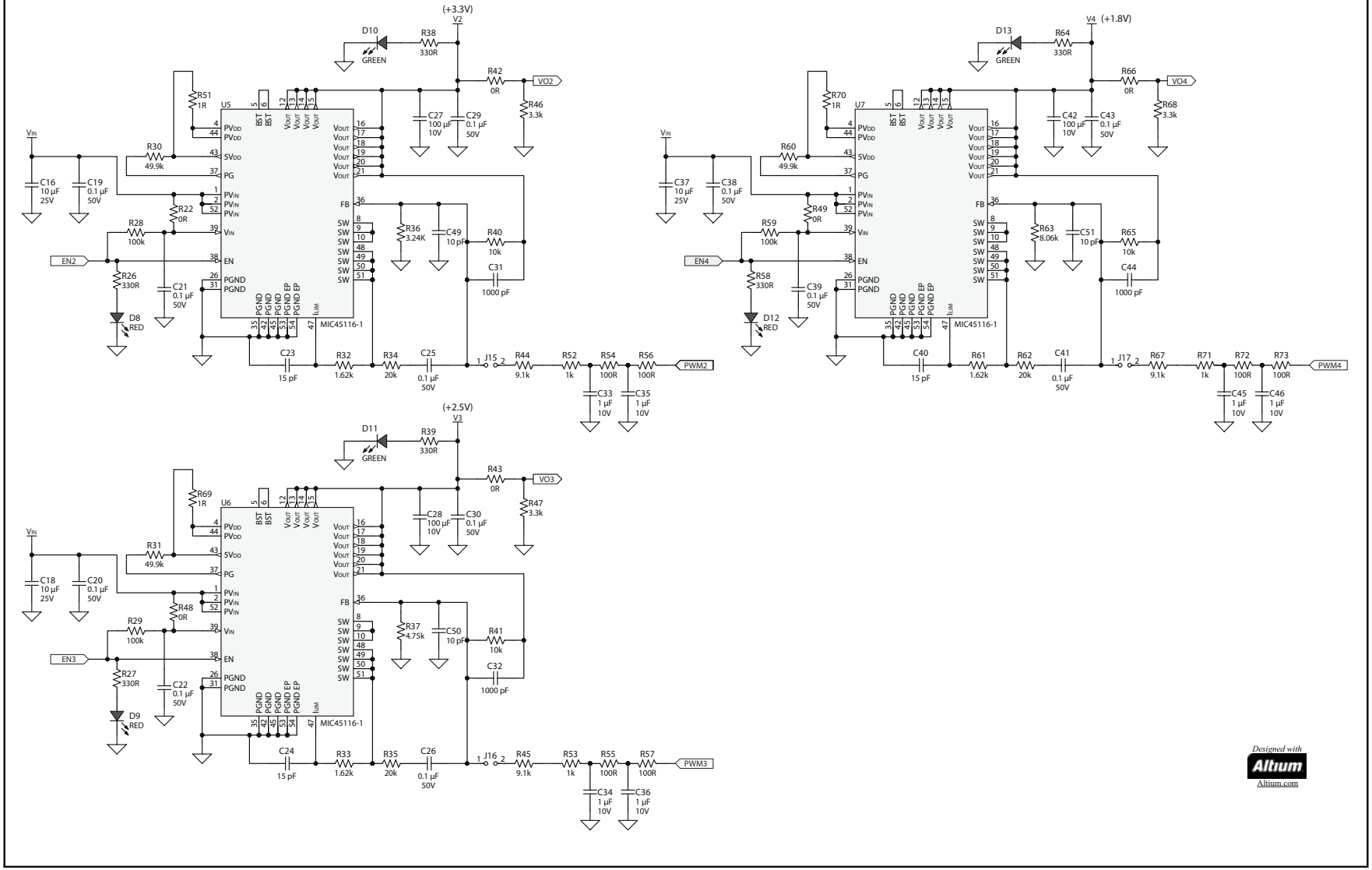


图 F-2：　　电源定序器原理图版本 C （第 2 页，共 2 页）



注：

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了 《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。Microchip 的 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949═

ISBN：978-1-5224-2578-6

10/25/17

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://www.microchip.com/support
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 Austin, TX
Tel: 1-512-257-3370

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Novi, MI
Tel: 1-248-848-4000

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453
Tel: 1-317-536-2380

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608
Tel: 1-951-273-7800

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110
Tel: 1-408-436-4270

加拿大多伦多 Toronto
Tel: 1-905-695-1980
Fax: 1-905-695-2078

亚太地区

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733

印度 India - Bangalore
Tel: 91-80-3090-4444

印度 India - New Delhi
Tel: 91-11-4160-8631

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870

菲律宾 Philippines - Manila
Tel: 63-2-634-9065

新加坡 Singapore
Tel: 65-6334-8870

泰国 Thailand - Bangkok
Tel: 66-2-694-1351

越南 Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

芬兰 Finland - Espoo
Tel: 358-9-4520-820

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

德国 Germany - Heilbronn
Tel: 49-7131-67-3636

德国 Germany - Karlsruhe
Tel: 49-721-625370

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

德国 Germany - Rosenheim
Tel: 49-8031-354-560

以色列 Israel - Ra'anana
Tel: 972-9-744-7705

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

意大利 Italy - Padova
Tel: 39-049-7625286

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

罗马尼亚 Romania - Bucharest
Tel: 40-21-407-87-50

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

瑞典 Sweden - Gothenberg
Tel: 46-31-704-60-40

瑞典 Sweden - Stockholm
Tel: 46-8-5090-4654

英国 UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820