用C8051F300设计锂离子电池充电器的解决方案

相关器件

本应用笔记适用于下列器件：

C8051F300

引言

鉴于产品使用的灵活性以及方便性的需要，许多系统都把可充电电池作为主要的能量来源。 电池充电器的较典型的实现方法是用一个专门功能的集成电路（IC）去控制充电电流/电压的范围。

C8051F300 系列单片机提供一个灵活的替代专门功能的电池充电器的解决方案。本应用笔记就是讨论如何使用 C8051F300 器件设计锂离子电池充电器。锂离子充电的算法同样也适用于其他的化学电池充电器。

关键点

- 片上高速 ADC 为控制充电电压提供较高的精度 (这一点对于防止锂离子电池应用中的过充电是非常重要的)，并使充电效率和电池寿命达到最大化。
- 片上 PWM（脉宽调制）提供了用一个小的外部电感实现快速转换的方法。
- 片上温度传感器为测量电池的温度提供了一个精确、稳定的驱动电压。也可以通过灵活的模拟输入 AMUX 用一个外部 RTD （电阻性温度器件）实现对电池温度的测量。
- 一个 C8051F300 可以为实现多节化学性能的充电器提供充分的硬件资源，大大加快了产品的市场适应期并减少库存。

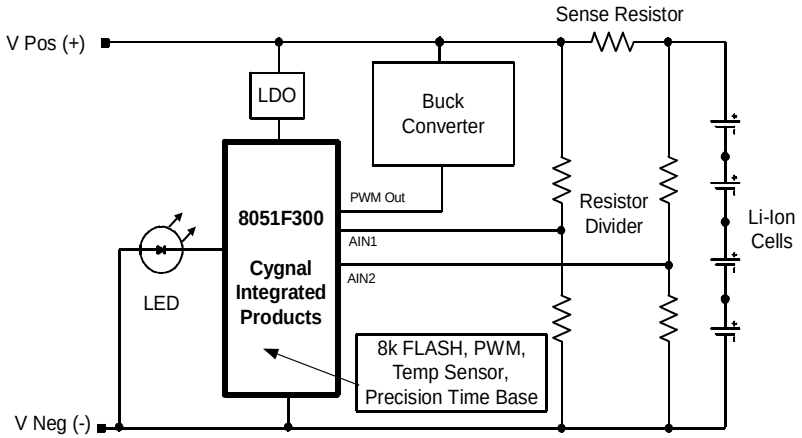


图1：锂离子电池充电模块图

充电原理

电池的特性唯一地决定其安全性能和充电的效率。电池的最佳充电方法是由电池的化学成分决定的（锂离子、镍氢、镍镉还是SLA电池等等）。尽管如此，大多数充电方案都包含下面的三个阶段：

1. 低电流调节阶段
2. 恒流阶段
3. 恒压阶段/充电终止

所有电池都是通过向自身传输电能的方法进行充电的。一节电池的最大充电电流取决于电池的额定容量（C）。例如，一节容量为1000mAh 的电池在充电电流为1000mA 时，可以充电 1C (电池容量的1倍)。也可

以用 1/50C(20m A) 或更低的电流给电池充电。尽管如此，这只是一个普通的低电流充电方式，不适用于要求短充电时间的快速充电方案。

现在使用的大多数充电器在给电池充电时都是既使用低电流充电方式又使用额定充电电流的方法（即所谓的容积充电）。低充电电流通常使用在充电的初始阶段，在这一阶段，需要将会导致充电过程终止的芯片初期的自热效应减小到最低程度。容积充电通常用在充电的中级阶段，电池的大部分能量都是在这一阶段存储的。

在电池充电的最后阶段（通常充电时间的绝大部分都是消耗在这一阶段），可以通过监测电流、电压或两者的值来决定何时结束充电。同样，结束方案依赖于电池的化学特性。例如，大多数锂离子电池充电器都是将电池电压保持在恒定值，同时检测最低电流。镍镉（NiCd）电池用电压或温度的变化率来决定充电的结束时间。

充电时， 部分电能被转换成热能，直至电池充满。而充满后，所有的电能将全部被转换成热能。 如果此时不终止充电未，电池就会被损坏或烧毁。快速充电器（电池完全充满的时间小于两小时的充电器）则可以解决这个问题，因为这些充电器是使用高充电电流来缩短充电时间的。因此，对于锂离子电池来说，监测它的温度是至关重要的，因为电池在过充电时会发生爆裂。在所有的充电阶段都应该随时监测温度的变化，并且在温度超过最大设定值时，立即停止充电。

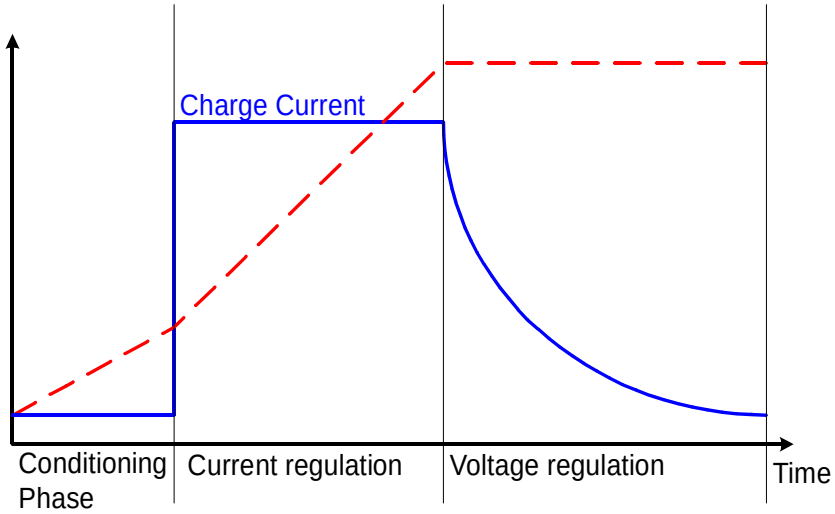
锂离子电池充电器 – 硬件部分

与其他的化学电池相比，锂离子电池具有较高的能量和容量比以及能量和重量比等性能，使得锂离子电池成为大多数应用首选的化学电池。现在使用的大多数锂离子电池充电器用渐弱终止充电和最小化电流（参见图2）的方法来确保电池完全充电，在本应用笔记的最后给出了程序示例源代码。

快速转换器

实现一个渐弱终止充电器的最经济的方法就是用一个快速转换器。快速转换器是一个用一个电感和/或一个变压器（需要隔离的时候用变压器）作为能量存储单元以离散的能量包的形式将能量从输入传输至输出的开关调节器。反馈电路通过晶体管来调节能量的传输，同时也作为过滤开关，以确保电压或电流在负载时保持恒定。

Figure 2. Lithium Ion Charge Profile.

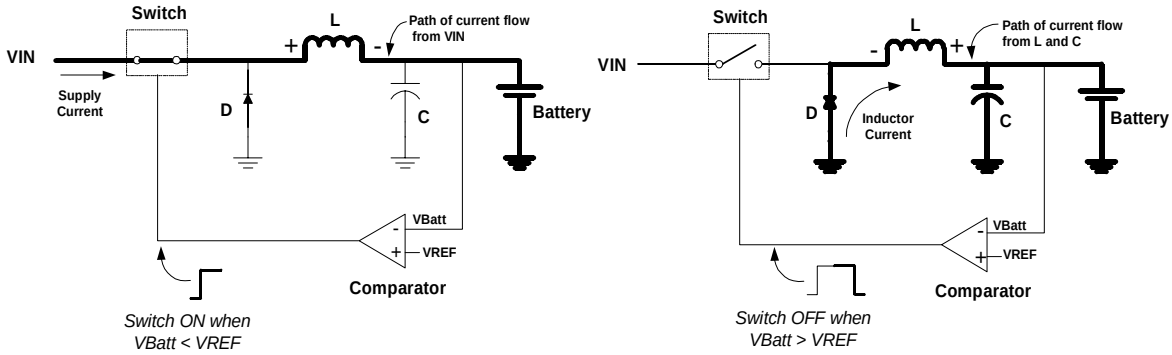


快速调节器操作

快速调节器的操作是通过控制一个晶体管开关的占空比来实现的。占空比会自动增加以使电池流入更多的电流。当 VBATT < VREF 时，一个比较器会将开关闭合。如图 3a所示，电流流入电池和电容 C。这个电流同时也存储在电感 L 中。 VBATT 持续升高，直到超过 VREF，此时比较器将开关断开（ 参见图 3b）。存储

在电感中的电流迅速下降直到二极管偏置，使得电感电流以减速度流入电池。电容C 在电感电流衰减后开始放电，并且最后 VBATT 开始下降。当 VBATT 低于 VREF时，比较器再次将开关闭合并开始另一次循环。在较大的范围内，如果减小占空比（缩短“闭合”的时间），平均电压就会下降，反之亦然。因此，可以通过控制占空比的方法调节电压或电流至所需要的值。

Figure 3. Buck Converter.



a) Switch ON　　b) Switch OFF

选择快速转换器的电感

要确定快速转换器中电感的大小，首先应假定晶体管的占空比为50%，因为此时的转换器操作操作效率最高。

占空比由方程式1给出，其中 T 是 PWM 的周期（在本例中T = 10.5 μS）。

占空比 = ton/T

方程式1. 占空比

至此，就可以选择一个PWM的转换频率。如方程式2所示，PWM的转换频率越大，则电感的值越小（也越节约成本）。我们的示例代码配置‘F300 的8位硬件 PWM 是使用内部24.5MHz主时钟的256分频来产生一个95.7kHz 的转换速率。

L = （Vi- Vsat-Vo）ton / （2Iomax）

方程式2. 电感值的确定

现在我们可以计算电感的大小了。假定充电电压 Vi 的值为 15V， 饱和电压 Vsat 的值为 0.5V， 需要获得的输出电压值为 4.2V，并且最大输出电流I0MAX为 1500 mA，那么，电感的值至少应选为 18μH 。

需要注意的是，在本电路中的电容仅仅是一个纹波衰减器。因为纹波与电容的大小成反比例关系，所以电容的值越大，衰减效果越好。

锂离子电池充电器 – 软件

本文后面的软件例程说明了如何使用C8051F300实现锂离子电池充电器。所讨论的算法完全是用“C”写的，以使程序更容易移植。关于该器件的具体说明，请参考F300的数据手册。

校正

为确保电压和电流的测量值的精确性，算法采用一个两点系统校正方案。在这个方案中，假定用户使用

两个已知的电压和两个已知的电流，一个点接近于地电平，另一个点接近于原测量值。然后算法采用这两个点，为电流和电压通道计算一个斜率和一个偏置值，并将结果存储在FLASH中。所有以后的转换都是相对于这些斜率和偏置计算值而言的。但需要注意的是，如果电流通道使用的是一个外部放大器，那么该放大器同样也需要使用一个类似的两点校正方案进行校正，以确保其精度。

温度

本例的算法使用片上温度传感器监测温度。温度传感器是没有经过校正的，但仍然可以提供充分精度的温度测量。如果需要获得更高精度的温度测量，可以通过一点或两点温度校正方案来实现。

当然，也可以使用一个外部温度传感器检测温度。可以通过重新配置 AMUX 来引入这个额外的输入电压。

电流

电池的充电电流是通过采集一个小的但精确的敏感电阻的差分电压的值来进行监控的。 经片上的PGA将电流放大后，采用片上8位ADC使用过采样的和均值的方法来获得16位的分辨率，再通过斜率和偏置校正系数计算出相应地电流值。如故想获得更高精度的电流测量值，就需要使用一个外部增益。

电压

电池的电压是通过外部的电阻进行衰减和监测的。需要注意的是本应用笔记中的例子是用电源电压作为ADC的参考电压。为了更精确地检测，必须将检测到的高于参考电压的电压值衰减。如果需要更精确的参考电压，可以使用外部参考。并同时相应地调节分压电阻的值。

充电 – 第一阶段

在第一阶段，（为了便于描述，我们假定电池在充电开始时是处于放电状态），‘F30x 调节电池的电流至ILOWCURRENT (典型值 1/50C) 直到电池的电压达到 VMINVOLTBULK。需要注意的是电池的充电电流需要限定至 ILOWCURRENT，以确保安全地启动充电并将电池的自热效应减至最小。如果温度在任何时候超过限定值，充电就会自动停止。

充电 – 第二阶段

一旦电池到达 VMINVOLTBULK ，充电就进入了第二阶段。在这一阶段，电池的算法控制 PWM 通路开关以确保输出电压为电池提供一个恒定的充电电流 IBULK (充电速率或容积电流通常为1C，并且和 ILOWCURRENT 与 VMINVOLTBULK一样可以在头文件中定义)。

充电 – 第三阶段

电池到达 VTop (在单节充电器中的典型值为 4.2 V)以后，充电器算法进入第三阶段，在这一阶段，PWM 将信号反馈回来并调节电池的电压。在第三阶段，电池继续充电直到电池的充电电流到达 IMINIBULKl，此后，电池将被额外充电30分钟，随后，充电终止。充电的绝大部分时间都用在第三阶段。

注意在大多数实际应用中（比如便携式PC机），当启动电池充电时，充电状态可能会处于三阶段中的任一阶段。但这不会影响充电效果，因为系统只是在监视电池的电流状态并在那一点启动充电过程。

结论

C8051F300 的高模拟集成度、小体积、集成的 FLASH 存储器以及低供耗等特点使得该产品成为灵活的新一代电池充电器应用的理想选择。本应用笔记讨论了如何用 C8051F300 实现锂离子电池充电器的应用方案。同时本应用笔记也给出了软件示例。

参考文献

线性集成电路的应用（Applications of Linear Integrated Circuits）
Eugene Hnatek, John Wiley a
nd Sons, 1975.

Appendix

Figure 4. 1 Cell Battery Charger Schematic.

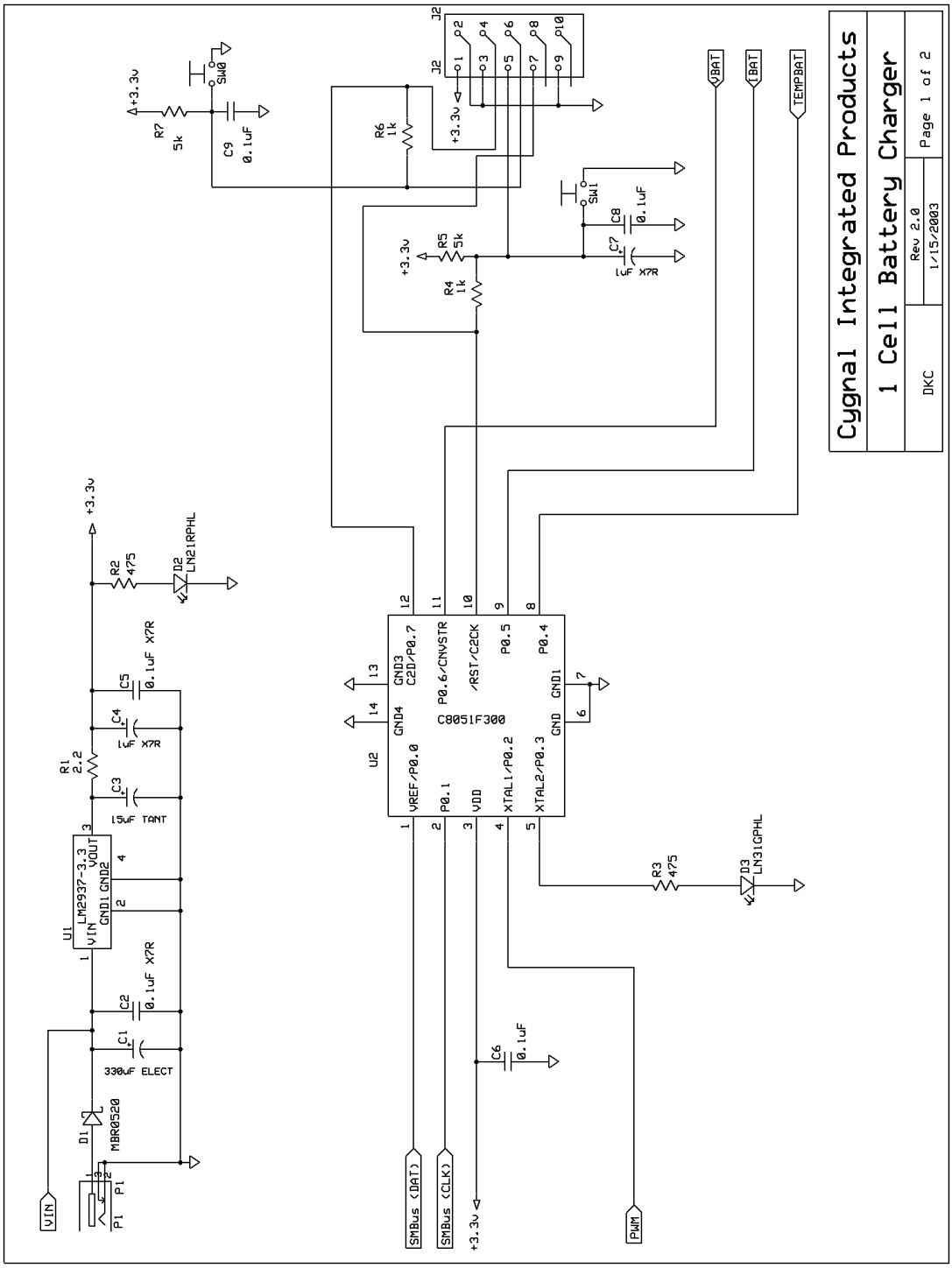


Figure 5. 1 Cell Buck Converter Schematic.

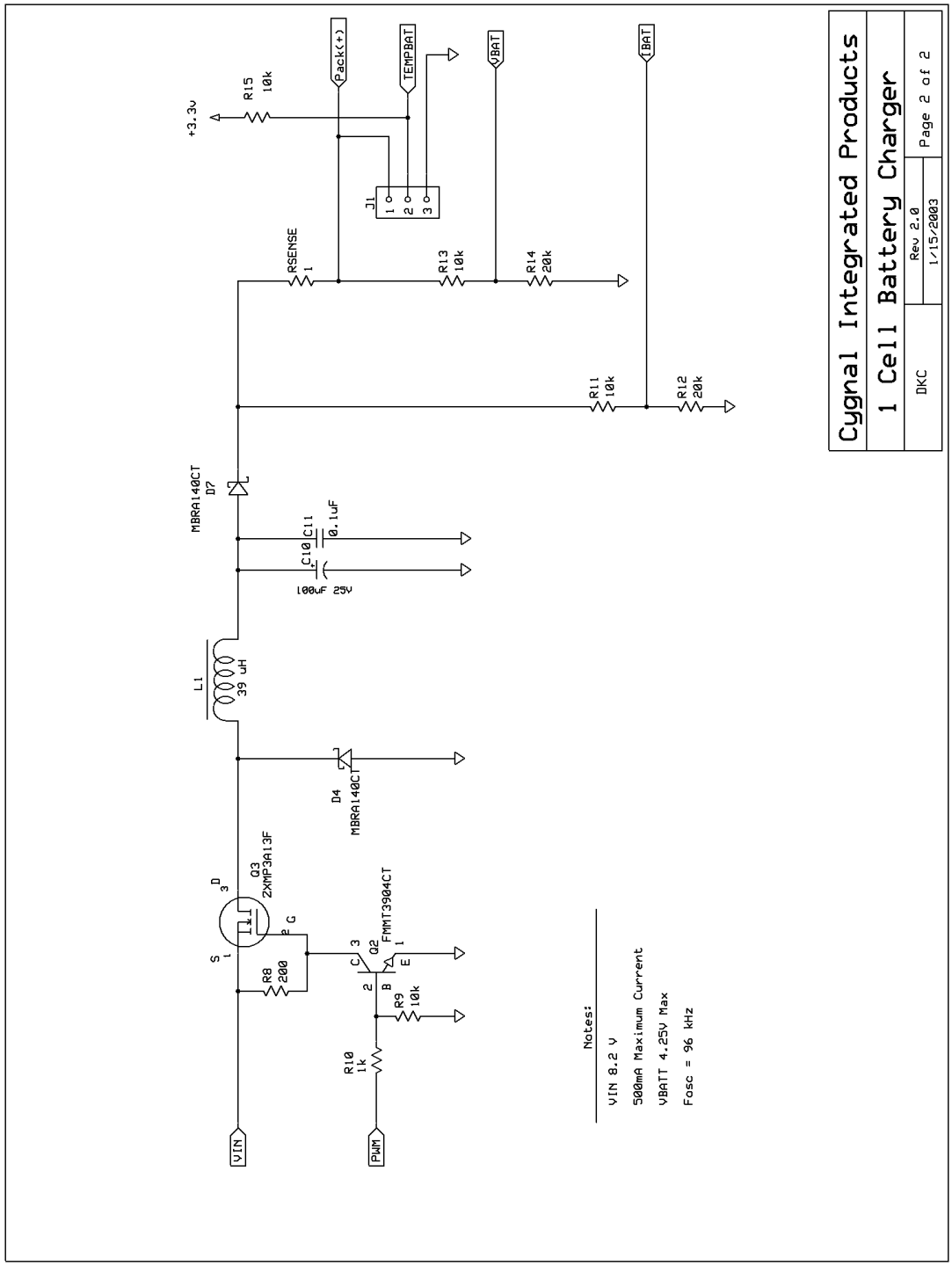


Figure 6. main() Flow Chart.

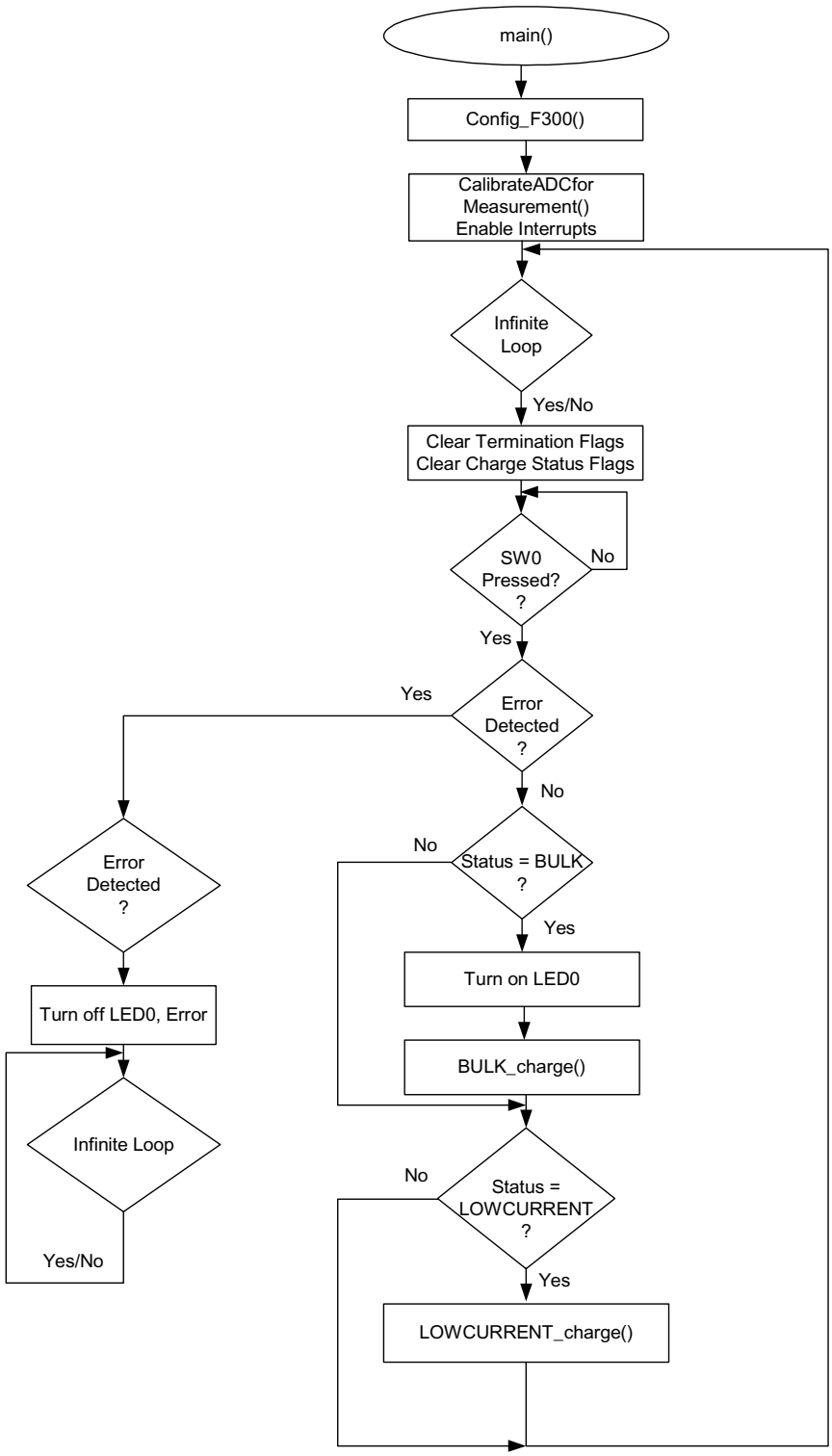


Figure 7. CalibrateADCforMeasurement() Flow Chart.

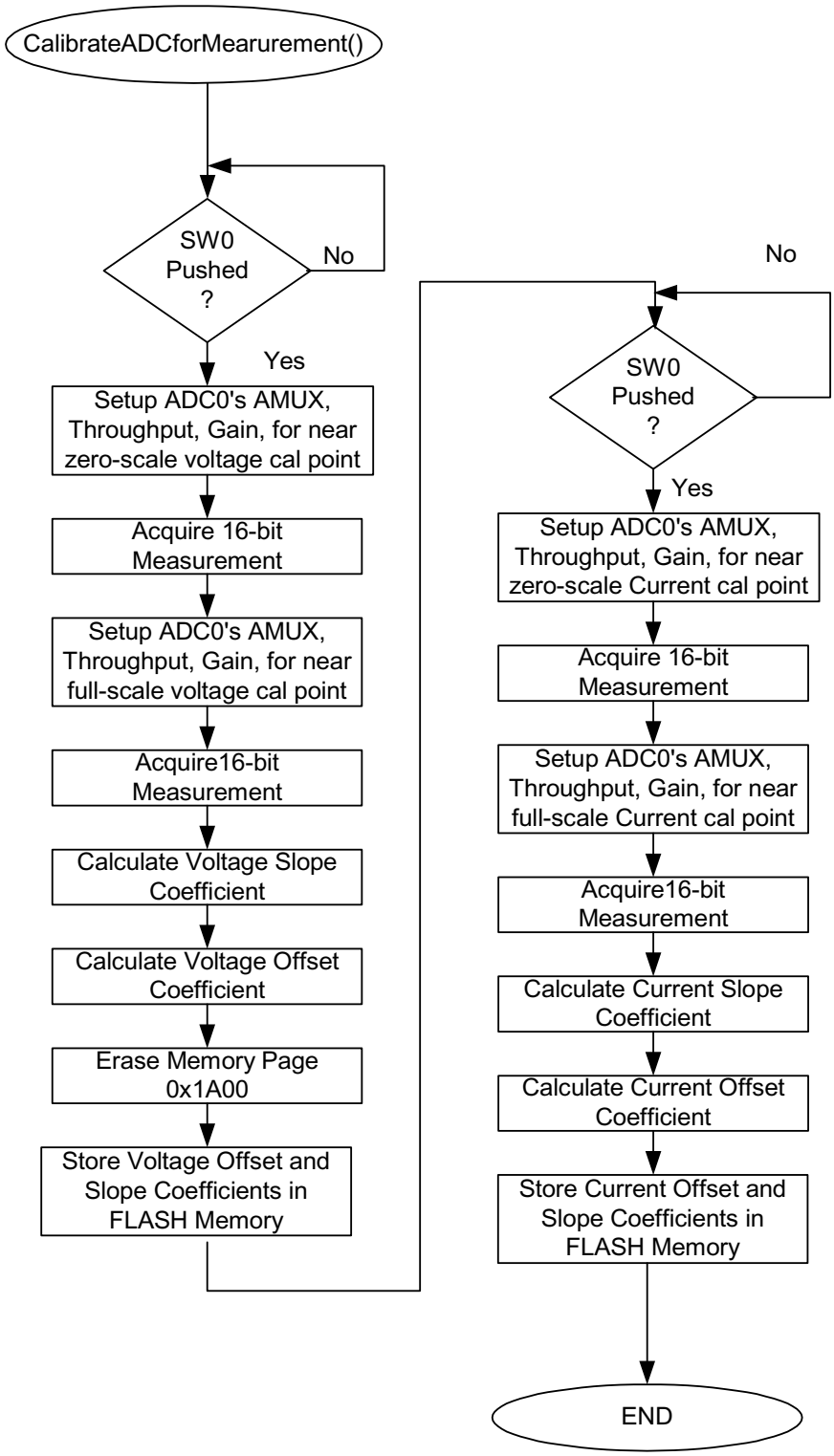


Figure 8. Monitor_Battery() Flow Chart.

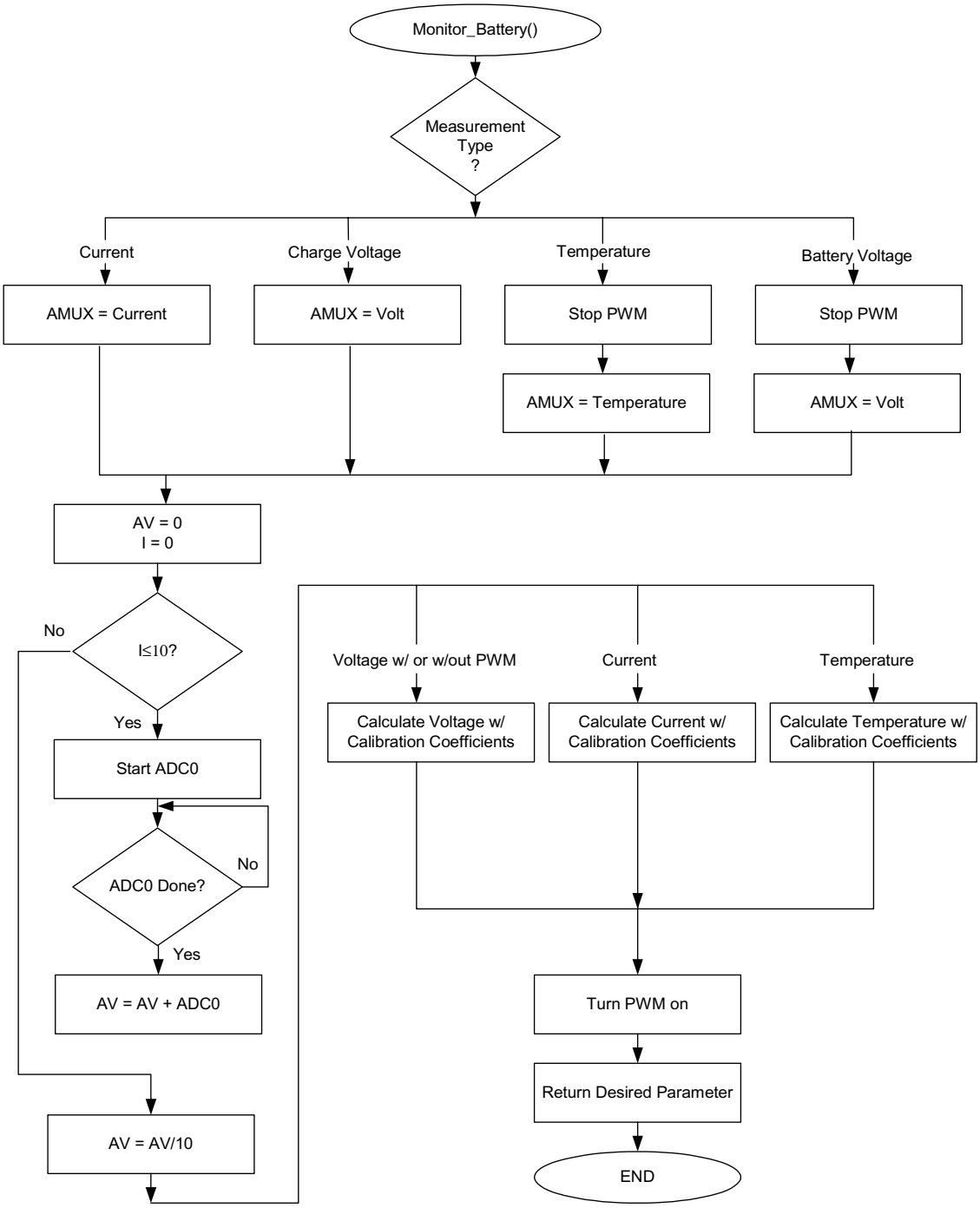


Figure 9. Bulk_Charge() Flow Chart (Part 1).

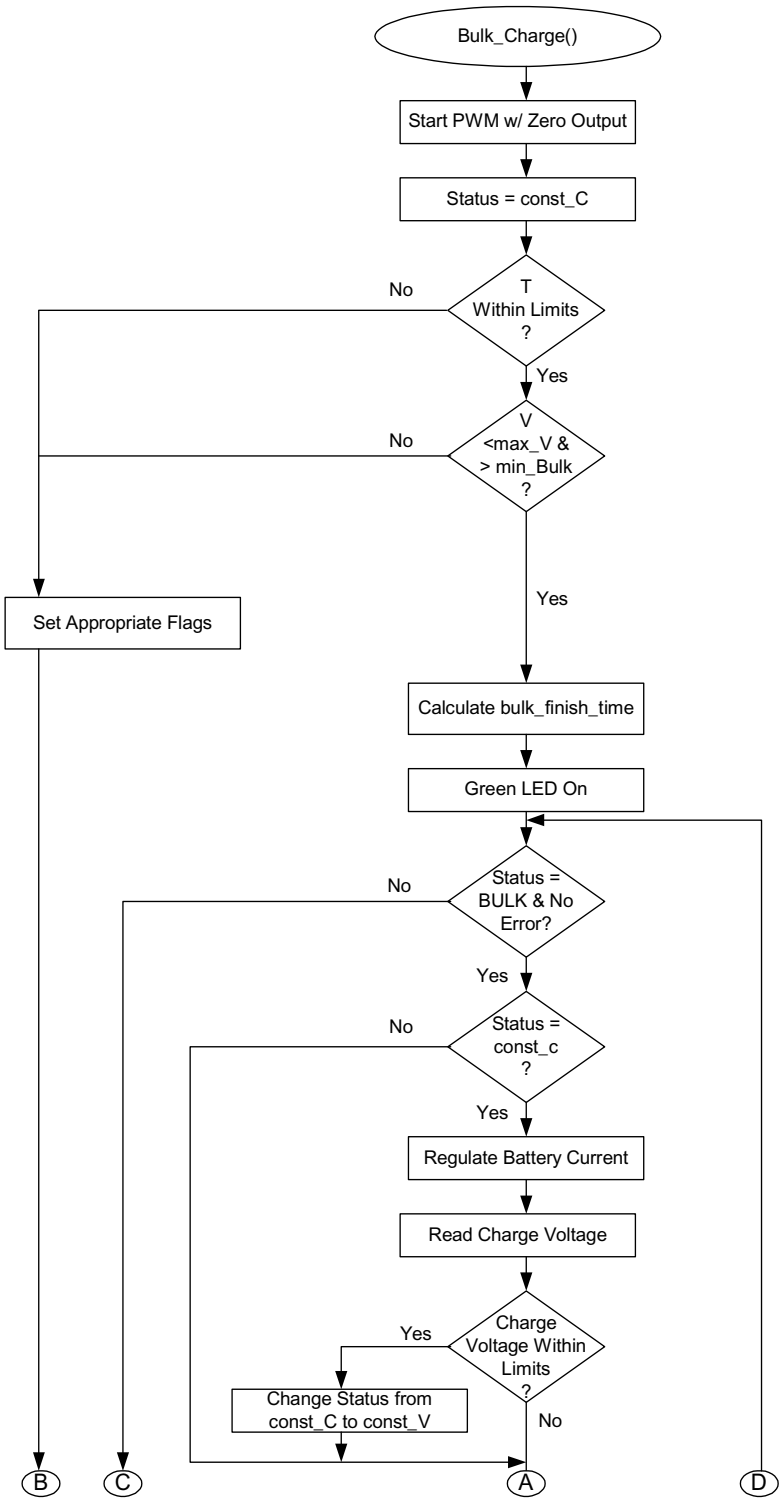


Figure 10. BULKCurrent() Flow Chart (Part 2).

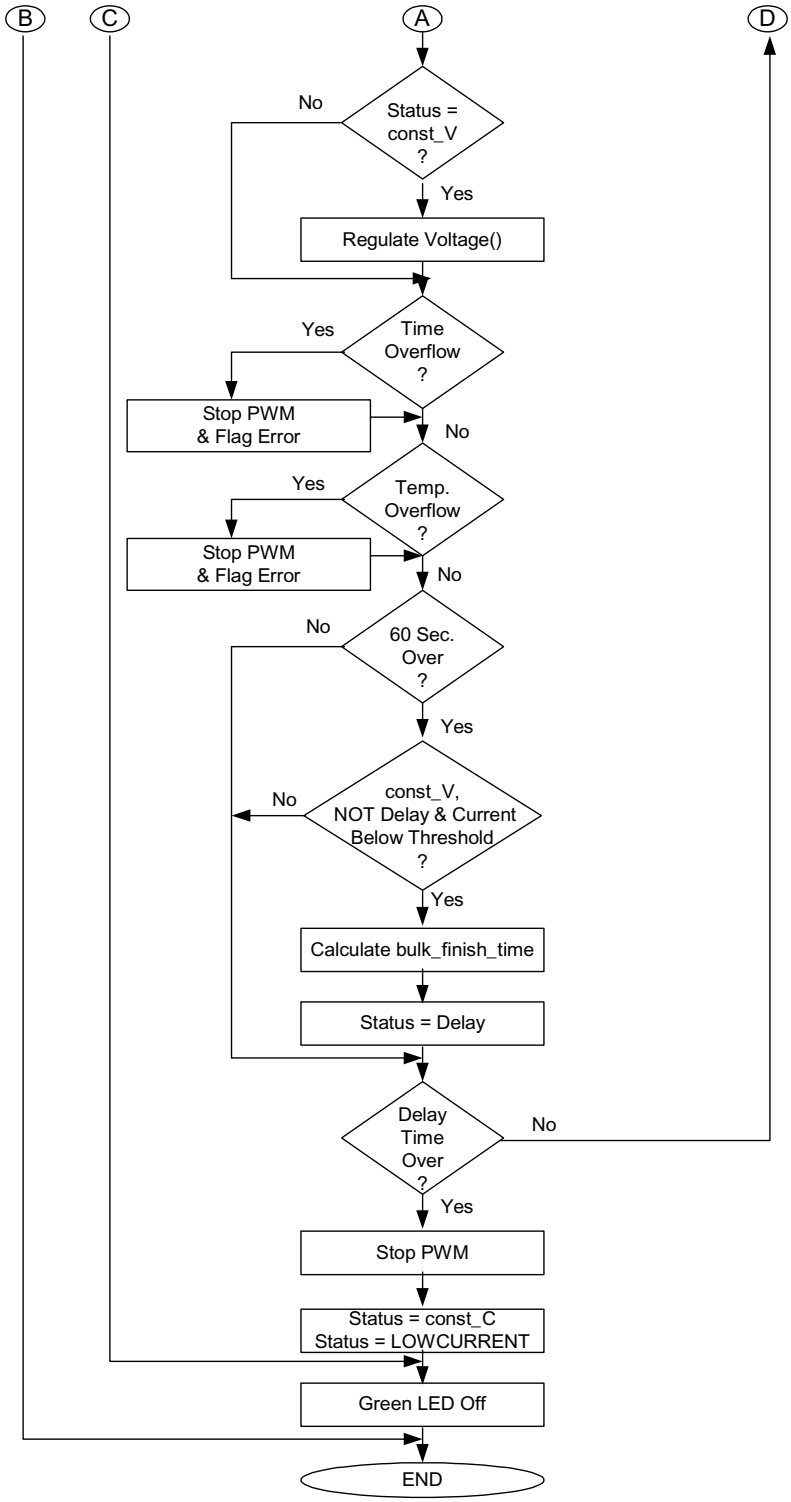


Figure 11. LowCurrent_Charge() Flow Chart.

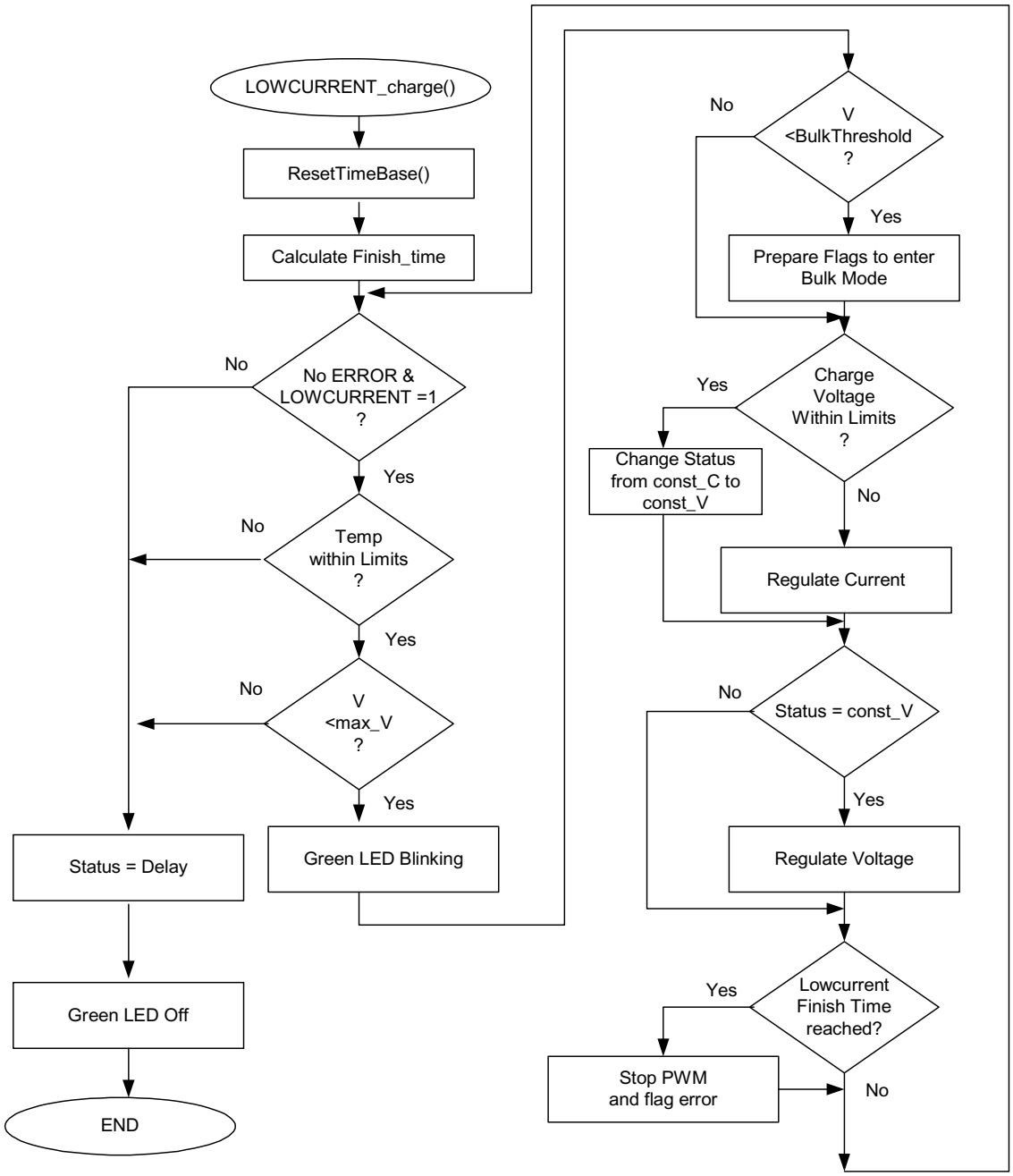


Figure 12. Turn_PWM_Off() Flow Chart.

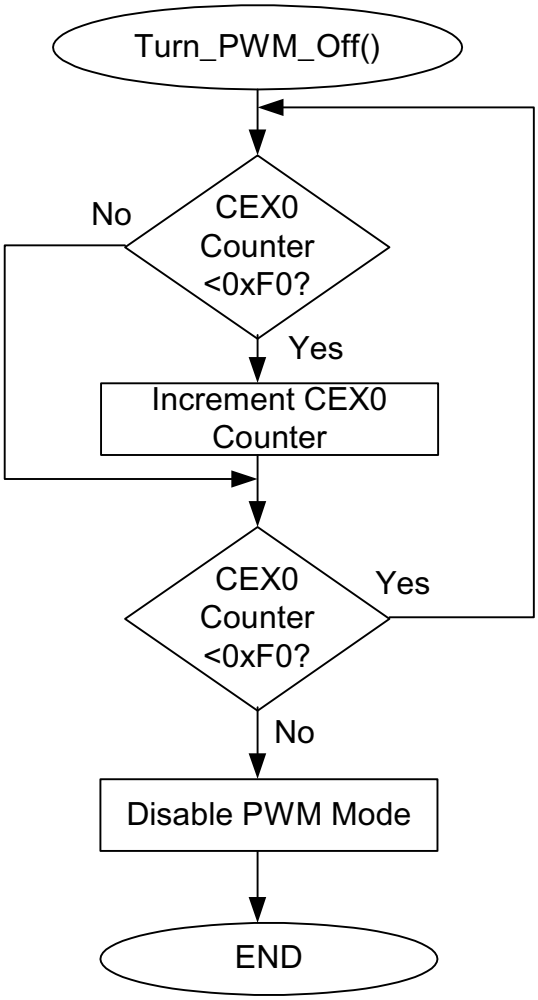


Figure 13. Measure() Flow Chart.

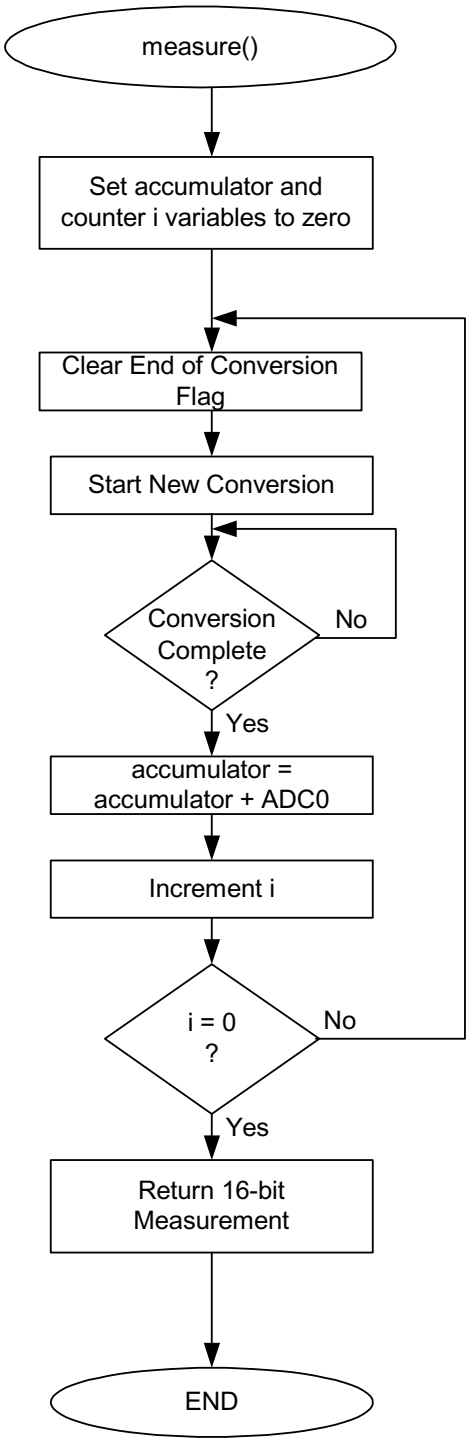


Figure 14. Regulate_Voltage() Flow Chart.

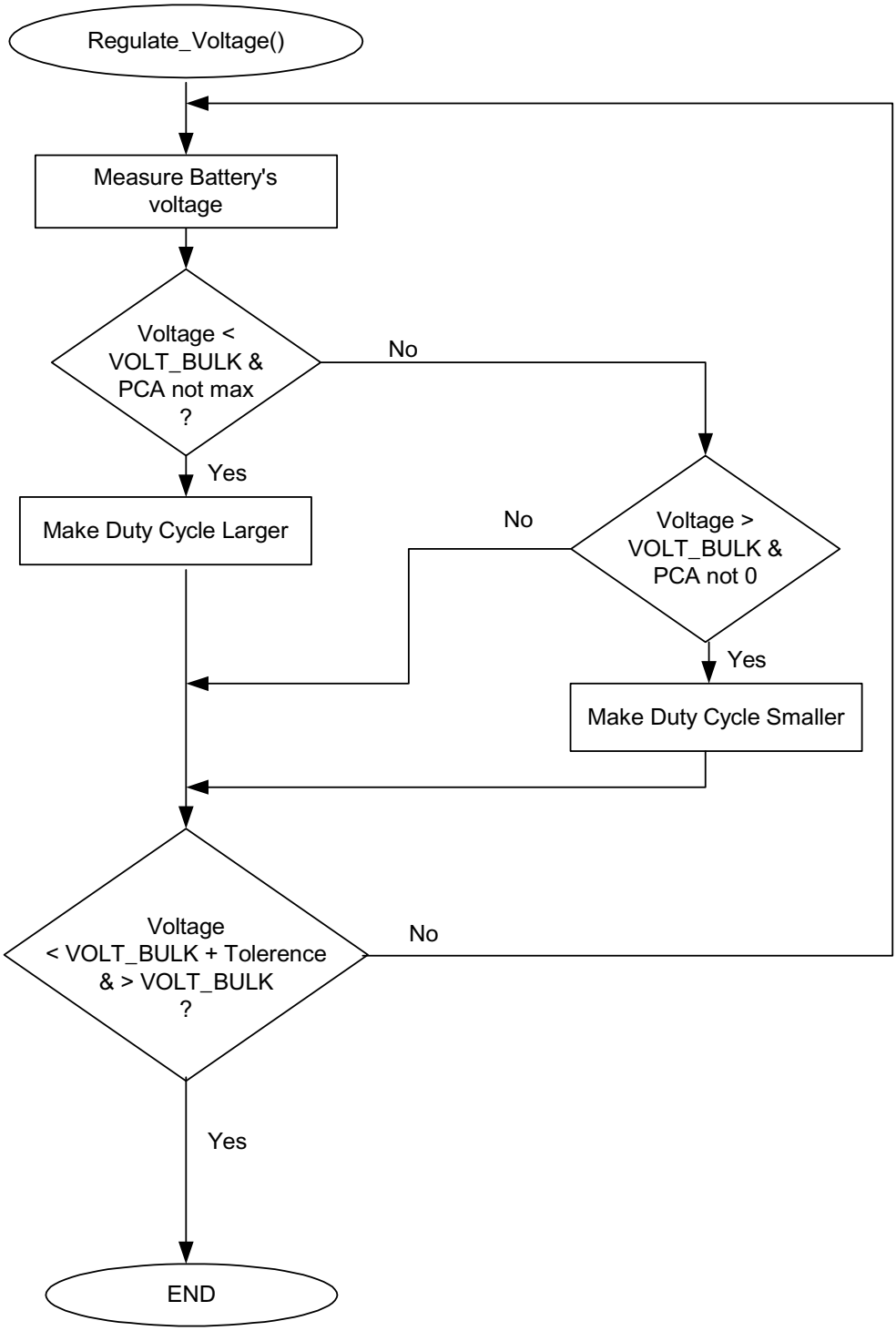


Figure 15. Regulate_Current() Flow Chart.

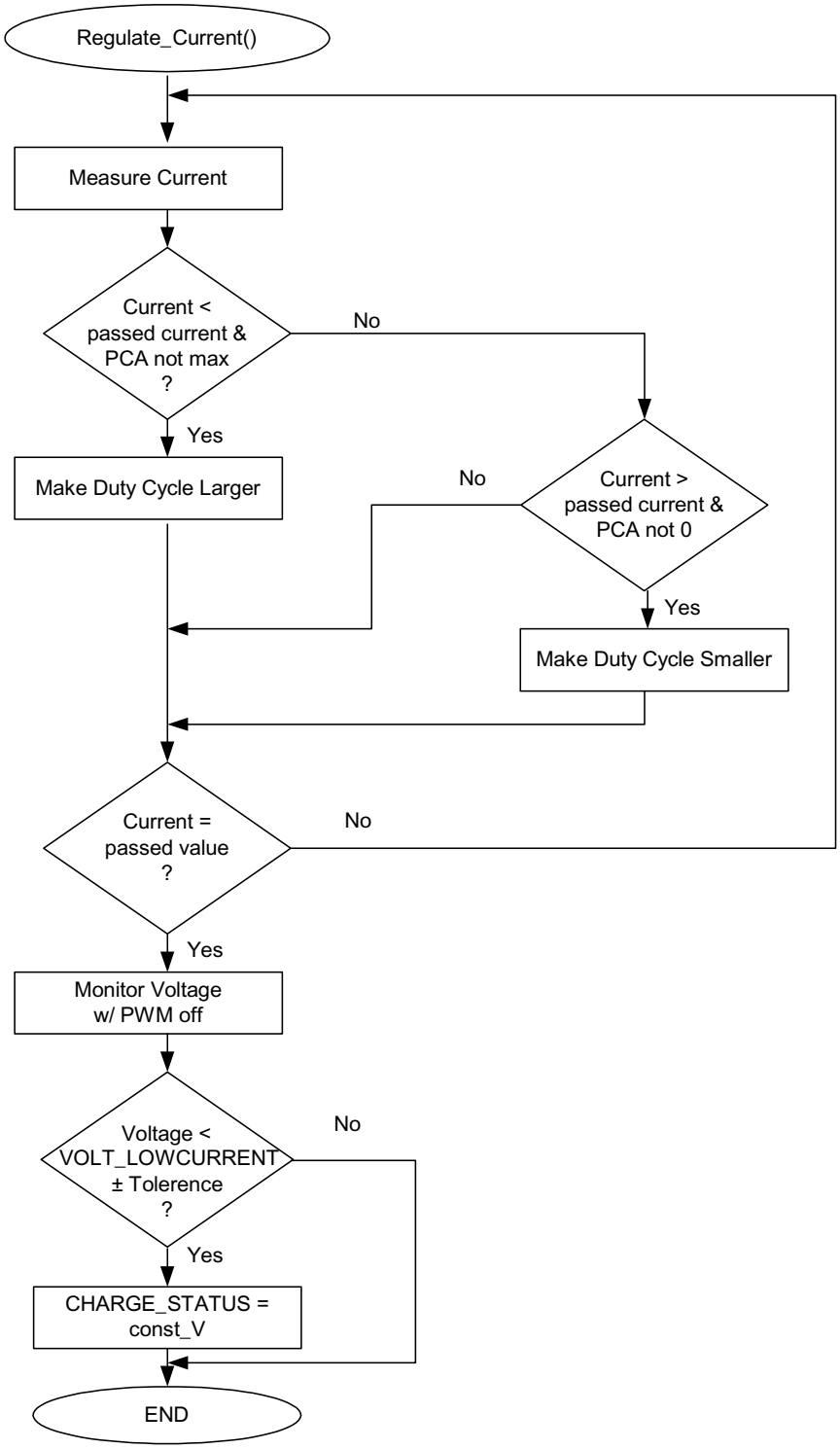
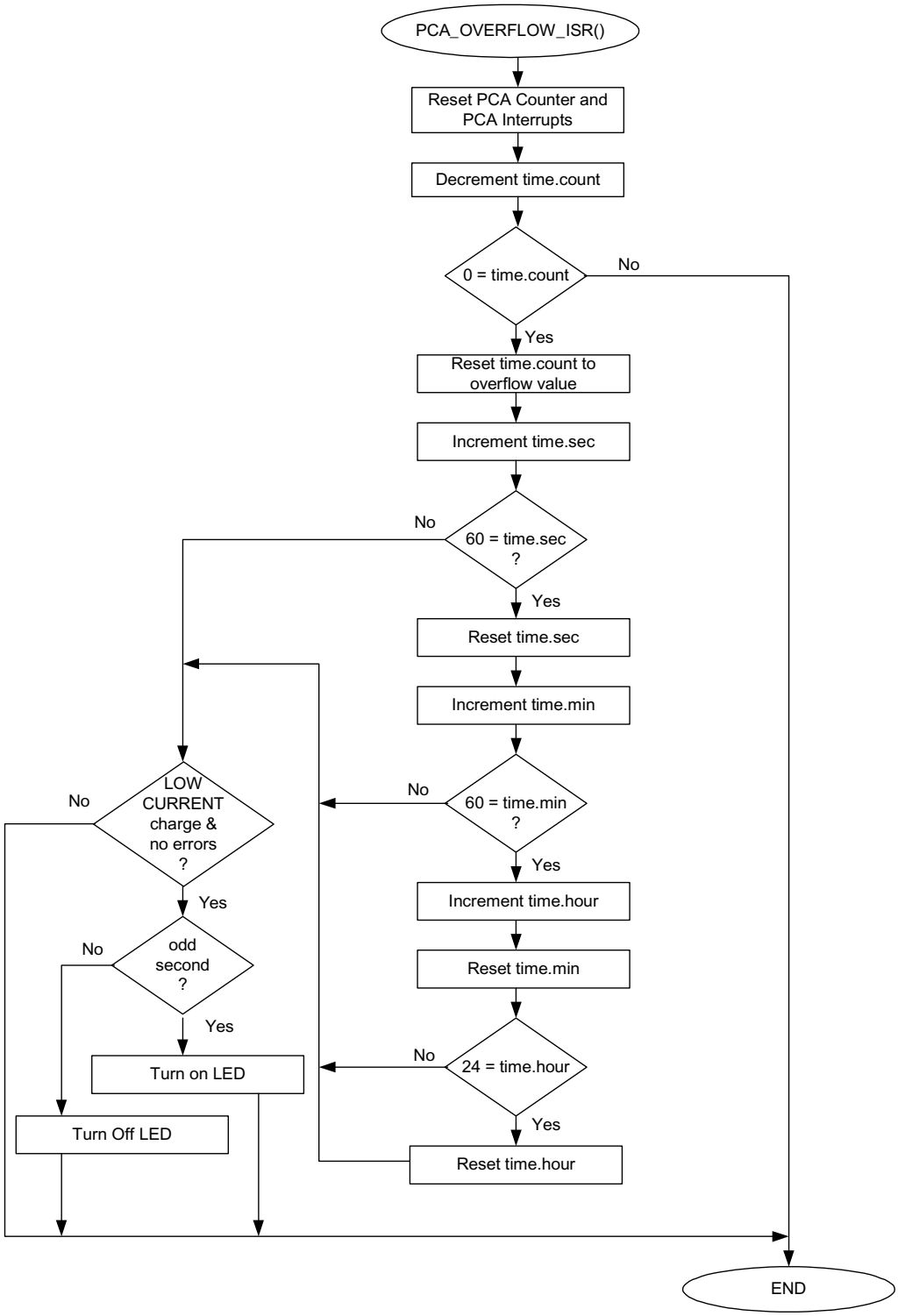


Figure 16. PCA_OVERFLOW_ISR() Flow Chart.



```
//-----------------------------------------------------------------------------
//
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// Filename:      LIION_BC_MAIN.h
// Target Device: 8051F300
// Created:       11 SEP 2002
// Created By:    DKC
// Tool chain: KEIL Eval C51
//
// This header file is used to define all preprocessor directives, prototypes,
// and global variable for LIION_BC_MAIN.c.
//
// The user should modify this header file before proceeding as key
//    battery parameter limits are set here.
//

//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
void Config_F300(void);
void Reset_Time_Base(void);
void CalibrateADCforMeasurement(void);
void Regulate_Current(int);
void Regulate_Voltage(void);
void Turn_PWM_Off(void);
int  Monitor_Battery(unsigned char);
void Bulk_Charge(void);
void Lowcurrent_Charge(void);
unsigned int Measure(void);
void Delay_Loop(void);

//-----------------------------------------------------------------------------
// UNIONs, STRUCTUREs, and ENUMs
//-----------------------------------------------------------------------------
typedef union LONG {                    // byte-addressable LONG
   long l;
   unsigned char b[4];
} LONG;

typedef union INT {                     // byte-addressable INT
   int i;
   unsigned char b[2];
} INT;

typedef struct
{
    unsigned long int t_count;
    int sec;                            // global seconds
    int min;                            // global minutes
    int hour;                           // global hour
}time_struct;
```

```
//-----------------------------------------------------------------------------
// Global Variable Definitions
//-----------------------------------------------------------------------------
time_struct TIME;                       // Global Struct to Track Time
char bdata TERMINATION;                 // Global Variable to Track Termination
char bdata CHARGE_STATUS;               // Global Variable to Track Charging
INT code CHECK_BYTE      _at_ 0x1A00;   // 0x0A0A Default value, for later use
LONG code VOLT_SLOPE     _at_ 0x1A60;   // Volt Slope Register
LONG code VOLT_OFFSET    _at_ 0x1A64;   // Volt Offset Register
LONG code I_NOAMP_SLOPE  _at_ 0x1A70;   // Current Slope Register,ext. amp off
LONG code I_NOAMP_OFFSET _at_ 0x1A74;   // Current Offset Register,ext. amp.off
LONG temp_LONG_1,temp_LONG_2;           // Temporary Storage Variables
INT  temp_INT_1,temp_INT_2;             // Temporary Storage Variables

//-----------------------------------------------------------------------------
// Bit maskable CHARGE STATUS Register Definition
//-----------------------------------------------------------------------------
sbit BULK       = CHARGE_STATUS^0;      // bit 0 : BULK charge status bit
sbit LOWCURRENT = CHARGE_STATUS^1;      // bit 1 : LOWCURRENT charge status bit
sbit ERROR      = CHARGE_STATUS^2;      // bit 2 : ERROR before/during charging
sbit CONST_V    = CHARGE_STATUS^3;      // bit 3 : charged w/ constant VOLTAGE
sbit CONST_C    = CHARGE_STATUS^4;      // bit 4 : charged w/ constant CURRENT
sbit DELAY      = CHARGE_STATUS^5;      // bit 5 : BULK charge DELAY for LiIon
                                        //    after CURRENT threshold detection
sbit READY      = CHARGE_STATUS^6;      // bit 6 : Lowcurrent charge is
                                        //     terminated; battery is charged
sbit FREE1      = CHARGE_STATUS^7;      // bit 7 : Not Currently used

//-----------------------------------------------------------------------------
// Bit Maskable TERMINATION Register Definition
//-----------------------------------------------------------------------------
sbit TEMP_MIN  = TERMINATION^0;         // bit 0 : minimum TEMPERATURE overflow
sbit TEMP_MAX  = TERMINATION^1;         // bit 1 : maximum TEMPERATURE overflow
sbit I_MIN     = TERMINATION^2;         // bit 2 : minimum CURRENT overflow
sbit I_MAX     = TERMINATION^3;         // bit 3 : maximum CURRENT overflow
sbit TIME_MAX  = TERMINATION^4;         // bit 4 : maximum time overflow
sbit VOLT_MAX  = TERMINATION^5;         // bit 5 : maximum VOLTAGE overflow
sbit VOLT_MIN  = TERMINATION^6;         // bit 6 : minimum VOLTAGE overflow
sbit FREE2     = TERMINATION^7;         // bit 7 : Not Currently used

//-----------------------------------------------------------------------------
// Bit maskable PORT Definitions
//-----------------------------------------------------------------------------
sbit SDA       = P0 ^ 0;                // bit 0 : SDA In/Output, Pin P0.
sbit SCL       = P0 ^ 1;                // bit 1 : SCL Output, Pin P1.
sbit CEX0      = P0 ^ 2;                // bit 2 : PWM Output, Pin P2.
sbit LED0      = P0 ^ 3;                // bit 3 : LED0, Pin P0.3
sbit SW0       = P0 ^ 7;                // bit 7 : Switch0, Pin P0.7

                                        // AMUX Selections; Analog Inputs
#define TBAT   0xF8;                    // bit 4 : Temp.   Ch.; Analog In
#define IBAT   0x65;                    // bit 5 : Current Ch.; Analog In
#define VBAT   0xF6;                    // bit 6 : Voltage Ch.; Analog In
```

```
//-----------------------------------------------------------------------------
// 8051F300 PARAMETERS
//-----------------------------------------------------------------------------
#define SYSCLK             24500000     // System clock frequency
#define TEMP_SENSOR_GAIN   3300         // Temp Sensor Gain in (uV / degC)
#define TEMP_GAIN          2            // PGA gain setting
#define CURRENT_GAIN       4            // PGA gain setting
#define VREF               3200         // ADC Voltage Reference (mV)
#define SCRATCH_PAGE       0x1C00       // FLASH page used for temp storage
#define PWM_CLOCK          SYSCLK/255   // PWM frequency is 96 kHz

//-----------------------------------------------------------------------------
// Calibration/Calculation PARAMETERS
//-----------------------------------------------------------------------------
#define V1_CAL             67           // 1st cal point for 2 point cal.
#define V2_CAL             2800         // 2nd cal point for 2 point cal.
#define I1_CAL             67           // 1st cal point for 2 point cal.
#define I2_CAL             133          // 2nd cal point for 2 point cal.
#define RSENSE             1            // RSENSE is assumed to be 1/2 ohm
#define RESB               20           // 10k Ohms, Voltage Divide Resistor
#define RESAB              30           // 20k Ohms, Voltage Divide Resistor


#define TEMP_SLOPE ((long) TEMP_GAIN * TEMP_SENSOR_GAIN * 65536 / 100 / VREF)
                                        // An estimate of the Temperature<SLOPE>
                                        // in [tenth codes / K]
                                        // The temperature measurement is
                                        // within 3 degrees of accuracy.

//-----------------------------------------------------------------------------
// Monitor_Battyer Switch PARAMETERS
//-----------------------------------------------------------------------------
#define TEMPERATURE        7            // Value for Switch Statement
#define VOLTAGE            5            // Value for Switch Statement
#define VOLTAGE_PWM_OFF    3            // Value for Switch Statement
#define CURRENT            1            // Value for Switch Statement


//-----------------------------------------------------------------------------
// Battery/Pack Parameters
//-----------------------------------------------------------------------------
#define  CELLS              1            // Number of cells in the battery pack
#define  CAPACITY           150          // mAh, Battery Capacity (LiIon)
#define  LiIon_CELL_VOLT    4200         // mV, Nominal Charge Voltage
#define I_BULK             (unsigned int)(CAPACITY)
#define I_LOWCURRENT       (unsigned int)(CAPACITY/4)
#define VOLT_BULK          (unsigned int)(LiIon_CELL_VOLT)

#define VOLT_LOWCURRENT    (unsigned int)(LiIon_CELL_VOLT)

#define VOLT_TOLERANCE     (unsigned int)(LiIon_CELL_VOLT/100)// 1 Percent Acc
#define CURRENT_TOLERENCE  (unsigned int)(CAPACITY/10)        // 10 Percent Acc
```

```
//-----------------------------------------------------------------------------
// Battery Characteristics: Charge TERMINATION Limits
//-----------------------------------------------------------------------------
#define  MIN_TEMP_ABS         26300      // Abs. min. TEMPERATURE = -10 C, 263K
#define  MAX_TEMP_ABS         32300      // Abs. max. TEMPERATURE = 50C, 323K:
#define  MIN_VOLT_BULK        3000       // Minimum BULK Voltage
#define  MAX_VOLT_ABS         (unsigned int)(CELLS * LiIon_CELL_VOLT)
#define  MIN_I_BULK           (unsigned int)(CAPACITY/4)
#define  MAX_TIME_LOWCURRENT  30         // Max Lowcurrent Charge Time = 90min
#define  MAX_TIME_BULK        90         // Maximum BULK Charge Time = 90 min
                                         //   at 1C CURRENT
#define  BULK_TIME_DELAY      30         // DELAY  = 30min after "MIN_I_BULK"

// END OF FILE
```

```
//-----------------------------------------------------------------------------
//
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// Filename:      LIION_BC_MAIN.c
// Target Device: 8051F300
// Created:       11 SEP 2002
// Created By:    DKC
// Tool chain:    KEIL Eval C51
//
// This is a stand alone battery charger for a Lithium ION battery.
// It utilizes a buck converter, controlled by the on-chip 8-bit PWM,
// to provide constant current followed by constant voltage battery charge.
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f300.h>
#include "LIION_BC_MAIN.h"              // Battery Hearder File

//-----------------------------------------------------------------------------
// Functions
//-----------------------------------------------------------------------------

void Config_F300(void)
{ RSTSRC   = 0x02;                      // Enable VDD Monitor
  XBR0     = 0x70;                      // Skip P0.4,5,6; they're analog In
  XBR1     = 0x44;                      // Enable SMBus on P0.0, P0.1, and CEX0
  XBR2     = 0x40;                      // as PWM at P0.2
                                        // Enable crossbar and weak pull-ups

  P0MDOUT  = 0x0C;                      // Set P0.2 & P0.3 output to push-pull
  P0MDIN   = 0x8F;                      // Configure P0.4,5,6 as Analog Inputs

  OSCICN   = 0x07;                      // Set SYSCLK to 24.5MHz, internal osc.

  ADC0CN   = 0xC0;                      // Turn on the ADC Module;
                                        //   enable low power mode for settling

  REF0CN   = 0x0C;                      // Configure ADC's to use VDD for
                                        //   Voltage Reference,
                                        //   Enable On-chip Temperature Sensor
//-----------------------------------------------------------------------------
// PCA Configuration
//-----------------------------------------------------------------------------
  PCA0MD   = 0x00;                      // Disable WDT
  PCA0MD   = 0x08;                      // Set PWM Time base = SYSCLK

  PCA0L    = 0x00;                      // Initialize PCA Counter to Zero
  PCA0H    = 0x00;

  PCA0CN   = 0x40;                      // Enable PCA Counter
                                        // Clear PCA Counter Overflow flag
  //Module 0
  PCA0CPM0 = 0x00;                      // Configure CCM0 to 8-bit PWM mode
```

```
  PCA0CPL0 = 0xF0;                      // Initialize PCA PWM to small duty cycle
  PCA0CPH0 = 0xF0;                      // 0xF0 Ensures a Soft Initial Charge

  //Module 1
  PCA0CPM1 = 0x49;                      // Configure Module 1 as software timer
  PCA0CPL1 = 0xFF;                      // Initialize to 255 so that Interrupt
                                        //     is generated when PCA ends
                                        // 8-bit PWM Cycle
  PCA0CPH1 = 0x00;                      // PCA0CPH is the high byte of the
                                        //    Output Compare Module

  EIE1     = 0x08;                      // Enable PCA Overflow Interrupt
}

//-------------------------------------------------------------------------
// Reset_Time_Base - Resets all Time Counting Values
//-------------------------------------------------------------------------
void Reset_Time_Base()
{
  TIME.sec     = 0x00;
  TIME.min     = 0x00;
  TIME.hour    = 0x00;
  TIME.t_count = PWM_CLOCK;
}

//-------------------------------------------------------------------------
// Delay - This is a Delay to permit time for Switches to Debounce
//-------------------------------------------------------------------------
void Delay_Loop (void)
{
  long i=0;
  for (i=0;i<100000;i++);
}

//-------------------------------------------------------------------------
// Initialize CalibrateADCforVoltageMeasurement
//-------------------------------------------------------------------------
// This function calibrates the voltage channel and stores the calibration
// coefficients in the parameters volt_slope and volt_offset.
//
void CalibrateADCforMeasurement()
// This calibration routine uses a 2 point cal.
{ unsigned char xdata *pwrite;          // FLASH write pointer

  EA = 0;                               // Disable All Interrupts

  // Wait until 1st calibration voltage is ready for cal
  while (SW0 == 1);                     // Wait until SW0 pushed
  Delay_Loop();                         // Wait for Switch Bounce

  // Once ready, Get the first calibration voltage
  AMX0SL = VBAT;                        // Select appropriate input for AMUX
  ADC0CF = (SYSCLK/5000000) << 3;       // ADC conversion clock = 5.0MHz
  ADC0CF &=0xF8;                        // Clear any Previous Gain Settings
  ADC0CF |= 0x01;                       // PGA gain = 1
  temp_INT_1.i = Measure();
```

```
// Wait until 2nd calibration voltage is ready for cal
while (SW0 == 1);                       // Wait until SW0 pushed
Delay_Loop();                           // Wait for Switch Bounce

// Once ready, Get the 2nd calibration voltage
AMX0SL = VBAT;                          //   Change Mux for second point
temp_INT_2.i = Measure();

// Calculate the SLOPE                  // V1 and V2 are in tenth of a degree
temp_LONG_1.l = (unsigned)(temp_INT_2.i-temp_INT_1.i);
temp_LONG_1.l *= (unsigned)100;         // Account for Math Truncation Error
temp_LONG_1.l /= (unsigned)(V2_CAL - V1_CAL);


// Calculate the OFFSET
temp_LONG_2.l  = (unsigned)temp_INT_1.i;
temp_LONG_2.l -= (signed)(temp_LONG_1.l * V1_CAL/100);

temp_LONG_1.l = 2050;                   // If no cal. use these
temp_LONG_2.l = 0;                      //  as default values

// Erased memory at page 0x1A00
pwrite = (char xdata *)&(CHECK_BYTE.b[0]);

PSCTL = 0x03;                           // MOVX writes target FLASH memory;
                                        // FLASH erase operations enabled

FLKEY = 0xA5;                           // FLASH key sequence #1
FLKEY = 0xF1;                           // FLASH key sequence #2
*pwrite = 0x00;                         // initiate PAGE erase

// Write the Volt SLOPE and OFFSET to Flash
PSCTL = 1;                              // MOVX writes to Flash

pwrite = (char xdata *)&(VOLT_SLOPE.b[0]);
FLKEY = 0xA5;
FLKEY = 0xF1;                           // enable flash write
*pwrite = temp_LONG_1.b[0];
pwrite = (char xdata *)&(VOLT_SLOPE.b[1]);
FLKEY = 0xA5;
FLKEY = 0xF1;                           // enable flash write
*pwrite = temp_LONG_1.b[1];
pwrite = (char xdata *)&(VOLT_SLOPE.b[2]);
FLKEY = 0xA5;
FLKEY = 0xF1;                           // enable flash write
*pwrite = temp_LONG_1.b[2];
pwrite = (char xdata *)&(VOLT_SLOPE.b[3]);
FLKEY = 0xA5;
FLKEY = 0xF1;                           // enable flash write
*pwrite = temp_LONG_1.b[3];

pwrite = (char xdata *)&(VOLT_OFFSET.b[0]);
FLKEY = 0xA5;
FLKEY = 0xF1;                           // enable flash write
*pwrite = temp_LONG_2.b[0];
pwrite = (char xdata *)&(VOLT_OFFSET.b[1]);
FLKEY = 0xA5;
```

```
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[1];
  pwrite = (char xdata *)&(VOLT_OFFSET.b[2]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[2];
  pwrite = (char xdata *)&(VOLT_OFFSET.b[3]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[3];

  PSCTL = 0;                            // MOVX writes target XRAM

//-----------------------------------------------------------------------------
// Initialize CalibrateADCforCurrentMeasurement_NOAMP
//-----------------------------------------------------------------------------
// This function calibrates the current channel with no external amp
// and stores the calibration coefficients in the
// parameters i_noamp_slope and i_noamp__offset.
//
// This calibration routine uses a 2 point cal.
  // Wait until calibration voltage is ready for cal
  while (SW0 == 1);                     // Wait until SW0 pushed
  Delay_Loop();                         // Wait for Switch Bounce
  // Once ready, Get the first calibration voltage
  AMX0SL = IBAT;                        // Select appropriate input for AMUX
  ADC0CF = (SYSCLK/5000000) << 3;       // ADC conversion clock = 5.0MHz
  ADC0CF &=0xF8;                        // Clear any Previous Gain Settings
  ADC0CF |= 0x03;                       // Set PGA gain = 4
  temp_INT_1.i = Measure();             // Acquire 16-bit Conversion
  temp_INT_1.i *= 2;                    // Account for Differential Mode
  // Wait until 2nd calibration voltage is ready for cal
  while (SW0 == 1);                     // Wait until SW0 pushed
  Delay_Loop();                         // Wait for Switch Bounce

  // Once ready, Get the 2nd calibration voltage
  temp_INT_2.i = Measure();             // Acquire 16-bit Conversion
  temp_INT_2.i *=2;                     // Account for Differential Mode

  // Calculate the SLOPE
  temp_LONG_1.l =  (unsigned)(temp_INT_2.i - temp_INT_1.i);
  temp_LONG_1.l *= (unsigned)100;       // Account for Math Truncation Error
  temp_LONG_1.l /= (unsigned)(I2_CAL - I1_CAL);
  temp_LONG_1.l /= (unsigned)CURRENT_GAIN;// Account for Gain

  // Calculate the OFFSET
  temp_LONG_2.l =  (signed)(temp_INT_1.i/CURRENT_GAIN);
  temp_LONG_2.l -= (signed)(temp_LONG_1.l * V1_CAL/100);

  temp_LONG_1.l = 2050;                 // If no cal. use these
  temp_LONG_2.l = 0;                    //  as default values

  // Memory at 0x1A00 is already erased
  // Write the Volt SLOPE and OFFSET to Flash
  PSCTL = 1;                            // MOVX writes to Flash

  pwrite = (char xdata *)&(I_NOAMP_SLOPE.b[0]);
```

```
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_1.b[0];
  pwrite = (char xdata *)&(I_NOAMP_SLOPE.b[1]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_1.b[1];
  pwrite = (char xdata *)&(I_NOAMP_SLOPE.b[2]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_1.b[2];
  pwrite = (char xdata *)&(I_NOAMP_SLOPE.b[3]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_1.b[3];
  pwrite = (char xdata *)&(I_NOAMP_OFFSET.b[0]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[0];
  pwrite = (char xdata *)&(I_NOAMP_OFFSET.b[1]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[1];
  pwrite = (char xdata *)&(I_NOAMP_OFFSET.b[2]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[2];
  pwrite = (char xdata *)&(I_NOAMP_OFFSET.b[3]);
  FLKEY = 0xA5;
  FLKEY = 0xF1;                         // enable flash write
  *pwrite = temp_LONG_2.b[3];

  PSCTL = 0;                            // MOVX writes target XRAM
}

//-----------------------------------------------------------------------------
// Measure
//-----------------------------------------------------------------------------
//
// This routine averages 65536 ADC samples and returns a 16-bit unsigned
// result.
//
unsigned int Measure (void)
{
  unsigned i;                           // sample counter
  unsigned long accumulator=0L;         // here's where we integrate the
                                        // ADC samples

  // read the ADC value and add to running total
  i = 0;
  do {
    AD0INT = 0;                         // clear end-of-conversion indicator
    AD0BUSY = 1;                        // initiate conversion
    while(!AD0INT);                     // wait for conversion to complete
    accumulator += ADC0;                // read adc value and accumulate
    i++;                                // update counter
  } while (i != 0x0000);
```

```
  // the accumulator now contains 16 added bits of which 8 are usable
  return (unsigned int) (accumulator >> 8);
}

//-----------------------------------------------------------------------------
// Regulate_Current
//-----------------------------------------------------------------------------
// This routine monitors the battery's current and adjusts
// the PWM (i.e. duty cycle) to keep the current at a known value
//
void Regulate_Current(int passed_current)
{ unsigned int temp = 0;

  do{
    temp = Monitor_Battery(CURRENT);    // Measure Current

   if (temp < passed_current)
      PCA0CPH0--;
    if (temp > passed_current)
        PCA0CPH0++;

  }while ((temp < (passed_current - CURRENT_TOLERENCE)) ||
          (temp > (passed_current + CURRENT_TOLERENCE)));
                                        // I_BULK or I_LOWCURRENT is set now

  temp = Monitor_Battery(VOLTAGE_PWM_OFF);
                                        // If VOLTAGE within range,
                                        // change from constant CURRENT charge
                                        // mode to constant VOLTAGE charge mode
  if ((temp >= (VOLT_LOWCURRENT - VOLT_TOLERANCE)) &&
   (temp <= (VOLT_LOWCURRENT + VOLT_TOLERANCE)))
  {
    CONST_C = 0;
    CONST_V = 1;
  }

}

//-----------------------------------------------------------------------------
// Regulate_Voltage
//-----------------------------------------------------------------------------
// This routine monitors the battery's voltage and adjusts
// the PWM (i.e. duty cycle) to keep the voltage at a known value
//
void Regulate_Voltage(void)
{ unsigned int temp = 0;
                                        // set VOLT_BULK (with "soft start")
  do{
    temp = Monitor_Battery(VOLTAGE);

   if (temp < VOLT_BULK)
      PCA0CPH0--;
    if (temp > VOLT_BULK)
      PCA0CPH0++;

  }while ((temp < (VOLT_BULK - VOLT_TOLERANCE)) ||
```

```
            (temp > (VOLT_BULK + VOLT_TOLERANCE)));
                                        // VOLTAGE is set now
}

//-----------------------------------------------------------------------------
// Turn_PWM_Off
//-----------------------------------------------------------------------------
// This routine peforms a soft charge turn off by taking the PWM's
// duty cycle slowly to zero.
//
void Turn_PWM_Off(void)
{
  do{
    if (PCA0CPH0 < 0xF0)
      PCA0CPH0++;

  }while (PCA0CPH0 < 0xF0);
  // Duty Cycle is now small and safe to turn off.

  PCA0CPM0 = 0x00;                      // Disable PWM
}

//-----------------------------------------------------------------------------
// Monitor_Battery
//-----------------------------------------------------------------------------
// This routine acts as a switch when gathering different conversion types.
// It adjusts the throughput, adjust the AMUX and returns the current in mA,
//  voltage in mV, and temperature in C, 2% accurate.
//
int Monitor_Battery(unsigned char value)
{
  char i;
  unsigned long av =0;
  long signed result;

  ADC0CF = (SYSCLK/5000000) << 3;       // ADC conversion clock = 5.0MHz
  ADC0CF &= 0xF8;                       // Clear any Previous Gain Settings

  switch (value)
  {
    case TEMPERATURE:
      Turn_PWM_Off();                   // Turn PWM Off
      AMX0SL = TBAT;                    // Select appropriate input for AMUX
      ADC0CF |= 0x02;                   // Set PGA gain = 2
      break;

    case VOLTAGE:
      AMX0SL = VBAT;                    // Select appropriate input for AMUX
      ADC0CF |= 0x01;                   // Set PGA gain = 1
      break;

    case VOLTAGE_PWM_OFF:
      Turn_PWM_Off();                   // Turn PWM Off
      AMX0SL = VBAT;                    // Select appropriate input for AMUX
      ADC0CF |= 0x01;                   // Set PGA gain = 1
      break;
```

```
   case CURRENT:
      AMX0SL = IBAT;                    // Select appropriate input for AMUX
      ADC0CF |= 0x03;                   // Set PGA gain = 4
      break;

  }

  //Compute average of next 10 A/D conversions
  for(av=0,i=10;i;--i){
    AD0INT = 0;                         // clear end-of-conversion indicator
    AD0BUSY = 1;                        // initiate conversion
    while(!AD0INT);                     // wait for conversion to complete
    av = av+ADC0;
  }

  av = av/10;                           // Compute the average
  av = av<<8;                           // Convert to 16-bit conversion
                                        // ...to account for 16-bit cal.
                                        //    coefficients

  PCA0CPM0 = 0x42;                      // Turn on PWM

  switch (value)
  { case TEMPERATURE:
      result =  (long) av * 1000/TEMP_SLOPE;
      break;

    case VOLTAGE:
    case VOLTAGE_PWM_OFF:
    result = (av - VOLT_OFFSET.l);      // Account for System Errors
    result /= VOLT_SLOPE.l;             // Convert to Voltage in Millivolts
    result *= 100;                      // Account for Math Truncation Error
    result *= RESAB;                    // Account for Divide Resistors
    result /= RESB;
    break;
    case CURRENT:
      result =  av*2;                   // Account for Differential Mode
      result -= I_NOAMP_OFFSET.l;       // Account for System Errors
      result /= I_NOAMP_SLOPE.l;        // Convert to Milliamps
      result *= 100;                    // Account for Math Truncation Error
      result /= RSENSE;                 // Account for Sense Resistor
      result *= RESAB;                  // Account for Divide Resistors
      result /= RESB;
      result /= CURRENT_GAIN;
      break;
  }

  return (int) result;
}

//-----------------------------------------------------------------------------
// Bulk_Charge Function
//-----------------------------------------------------------------------------
void Bulk_Charge(void)
{
  unsigned int temp = 0;
  unsigned int bulk_finish_hour = 0;
```

```
unsigned int bulk_finish_min = 0;
unsigned int delay_hour = 0;
unsigned int delay_min = 0;
unsigned int last_min = 0;

Reset_Time_Base();                    // Reset Time Base to zero

                                      // Calculate BULK charge finish time
bulk_finish_min = (TIME.min + MAX_TIME_BULK);
bulk_finish_hour = TIME.hour;
while (bulk_finish_min > 60)
{
  bulk_finish_min = bulk_finish_min - 60;
  bulk_finish_hour++;
}

CONST_C = 1;                          // Start in constant current charge mode
DELAY   = 0;                          // Reset timer DELAY


temp = Monitor_Battery(TEMPERATURE);  // Monitor Temperature
                                      // Is temperature within range?

if ((temp > MIN_TEMP_ABS) && (temp < MAX_TEMP_ABS))
{
  temp = Monitor_Battery(VOLTAGE);    // Monitor Voltage
                                      // Is Voltage within range?
  if ((temp <= (MAX_VOLT_ABS + VOLT_TOLERANCE)) && temp > MIN_VOLT_BULK)
  {
    PCA0CPM0 = 0x42;                  // Configure CCM0 to 8-bit PWM mode

    // Enter main loop in Bulk_Charge()
    while ((BULK == 1) && (ERROR == 0))
    {
      if (CONST_C == 1)
        Regulate_Current(I_BULK);     // Charge with Constant Current

      else if (CONST_V == 1)
        Regulate_Voltage();           // Charge with Constant Voltage

      // Now, Check for error and charge termination conditions
      // If above max charge time, flag error
      // Test for BULK Charge Time Out

                                      // Monitor Time
      if ((TIME.hour == bulk_finish_hour) && (TIME.min == bulk_finish_min)
         &&  (DELAY == 0))
      {
        Turn_PWM_Off();               // Turn Off PWM
        TIME_MAX = 1;                 // Set Time max error flag
        ERROR    = 1;                 // Set general error flag
      }

                                      // Monitor Temperature
      temp = Monitor_Battery(TEMPERATURE);
      if ((temp < MIN_TEMP_ABS) && (temp > MAX_TEMP_ABS))
    {
```

```
          Turn_PWM_Off();                // Turn Off PWM

        if (temp < MIN_TEMP_ABS)
            TEMP_MIN = 1;                // Set Temperature below minimum flag
          else
            TEMP_MAX = 1;                // Set Temperature exceeds maximum flag

          ERROR    = 1;                  // Set general error flag
        }

                                         // Minute elapsed?
                                         // Check for minimum current
                                         // if reached, enter last DELAY charge
        if (TIME.min != last_min)
        {
          last_min = TIME.min;
          if ((CONST_V == 1) && (DELAY == 0) && (Monitor_Battery(CURRENT)
             <= MIN_I_BULK))
          {
                                         // Calculate TOP OFF Battery Time finish time
            delay_min = (TIME.min + BULK_TIME_DELAY);
            delay_hour = TIME.hour;
            while (delay_min > 60)
            {
              delay_min = delay_min - 60;
              delay_hour++;
            }
            DELAY = 1;                   // Set Delay Flag
          }

                                         // Monitor Delay time, time up?
          if ((TIME.hour == delay_hour)&&(TIME.min == delay_min) &&
             (DELAY == 1))
          {
            Turn_PWM_Off();              // Turn Off PWM
            CONST_V = 0;                 // Exit CONST_V
            CONST_C = 1;                 // Prepare to enter CONST_C
            BULK = 0;                    // Prepare to exit BULK mode
            LOWCURRENT = 1;              // Prepare to enter LOWCURRENT Mode
          }
        }
      }                                  // End Main While loop
    }

    else if(ERROR == 0)
    {
      if (temp > (MAX_VOLT_ABS + VOLT_TOLERANCE))
      { VOLT_MAX = 1;                    // Set Max Voltage error flag
        ERROR    = 1;                    // Set general error flag
      }
     else if(temp < MIN_VOLT_BULK)
      { VOLT_MIN = 1;                    // Set Minimum bulk voltage error flag
        LOWCURRENT = 1;                  // Switch to LOWCURRENT mode
        BULK = 0;                        // Exit Bulk Charge mode
      }                                  // battery's voltage very low
    }
  }
```

```
  else if(ERROR == 0)                    // Absolute temperature out of range?
  {
    if (temp < MIN_TEMP_ABS)
     TEMP_MIN = 1;                       // Set Temperature below minimum flag
    else
     TEMP_MAX = 1;                       // Set Temperature exceeds maximum flag

      ERROR = 1;                         // Set general error flag
  }
}

//-----------------------------------------------------------------------------
// Lowcurrent_Charge
//-----------------------------------------------------------------------------

void Lowcurrent_Charge(void)
{
  unsigned int temp = 0;
  unsigned int lowcurrent_finish_min = 0;
  unsigned int lowcurrent_finish_hour = 0;

  Reset_Time_Base();                     // Reset Time base to zero

                                         // Calculate LOWCURRENT finish time
  lowcurrent_finish_min = (TIME.min + MAX_TIME_LOWCURRENT);
  lowcurrent_finish_hour = TIME.hour;
  while (lowcurrent_finish_min > 60)
  {
    lowcurrent_finish_min = lowcurrent_finish_min - 60;
    lowcurrent_finish_hour++;
  }

  // Enter Main Lowcurrent Loop.
  // Only exits are upon error and full charge
  while ((LOWCURRENT == 1) && (ERROR == 0))
  {
    temp = Monitor_Battery(TEMPERATURE);// Get Temperature Reading
                                         // Is TEMPERATURE within limits
    if ((temp > MIN_TEMP_ABS) && (temp < MAX_TEMP_ABS))
    {
      // Is Battery's Charge Voltage below max charge voltage
      temp = Monitor_Battery(VOLTAGE);  // Get Voltage Reading
      if (temp <= (VOLT_LOWCURRENT + VOLT_TOLERANCE))
      {
        if (CONST_C == 1)                // CONST_C ?, charge w/ constant current
          Regulate_Current(I_LOWCURRENT);

        if (CONST_V == 1)                // CONST_V?, charge w/ constant voltage
          Regulate_Voltage();

        if ((temp >= MIN_VOLT_BULK) && (DELAY == 0))// Bulk Threshold voltage met?
        { LOWCURRENT = 0;                // Exit LOWCURRENT mode
          BULK = 1;                      // Switch to Bulk Charge mode
        }
                                         // Check elapsed time
        if ((TIME.hour == lowcurrent_finish_hour) &&
```

```
        ( TIME.min == lowcurrent_finish_min))
        {
          TIME_MAX = 1;                 // Set Time MAX error flag
          ERROR    = 1;                 // Set general error flag
        }
      }
      else if(ERROR == 0)               // Voltage to high?
      {
        VOLT_MAX = 1;                   // Set Max voltage error flag
        ERROR    = 1;                   // Set general error flag
      }
    }
    else if(ERROR == 0)                 // Absolute temperature out of range?
    {
      if (temp < MIN_TEMP_ABS)
        TEMP_MIN = 1;                   // Set Temperature below minimum flag
      else
        TEMP_MAX = 1;                   // Set Temperature exceeds maximum flag

      ERROR = 1;                        // Set general error flag
    }
  }
}

//-----------------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------------
void main(void)
{
  EA = 0;                               // Disable All Interrupts
  Reset_Time_Base();
  Config_F300();                        // Config F300
  CalibrateADCforMeasurement();         // Calibrate F300

  EA = 1;                               // Enable All Active Interrupts

  while(1)
  {
    LED0 = 0;                           // Turn LED0 off

    TERMINATION = 0x00;                 // Reset Termination Flags
    CHARGE_STATUS = 0x00;               // Reset Charge Status Flags
    BULK = 1;                           // Start in LOWCURRENT Charge mode
    CONST_C = 1;

    while (SW0 == 1);                   // Wait until SW0 pushed
    Delay_Loop();                       // Wait for Switch Bounce

    while (ERROR == 0)
    {
      if (BULK == 1)
      {
        LED0 = 1;                       // Turn LED0,indicates Bulk Mode
        Bulk_Charge();                  // Enter Bulk Charge Mode
     }
      if (LOWCURRENT == 1)
        Lowcurrent_Charge();            // Enter Lowcurrent_Charge function
```

```
                                        // Toggle LED0 at 1 Hz rate via ISR
    }

    if (ERROR == 1)
    {
      Turn_PWM_Off();;                  // Turn PWM Off
      LED0 = 0;                         // Turn OFF LED0 to indicate "ERROR".
      EA = 0;                           // Disable All Interrupts
      while (1);                        // Enter a eternal loop
                                        // No recovery except "reset-button"
    }
  }
}

//-----------------------------------------------------------------------------
// PCA_ISR
//-----------------------------------------------------------------------------
// This routine counts the elapsed time in seconds, minutes, hours.
// It also toggles LED0 every second when in Lowcurrent Charge Mode.
// This routine interrupts every time the PCA counter overflows, every 256
//  SYSCLK cycles. After SYSCLK/256 interrupts, one second has elapsed.
//
void PCA_OVERFLOW_ISR (void) interrupt 9
{
  PCA0CN = 0x40;                        // Reset all PCA Interrupt Flags

  PCA0H = 0x00;                         // Reset High Byte of PCA Counter
                                        //   of 8-bit PWM we are using Module1

  if (0x0000 == --TIME.t_count)
  {
    TIME.t_count = PWM_CLOCK;           // Reset 1 Second Clock
    if ( 60 == ++TIME.sec )             // Account for elapsed seconds
    {                                   // Reset second counter every minute
      TIME.sec = 0x00;
      if ( 60 == ++TIME.min )           // Account for elapsed minutes
      {                                 // Reset minute counter every hour
        TIME.min = 0x00;
        if ( 24 == ++TIME.hour )        // Account for elapsed hours
          TIME.hour = 0x00;             // Reset hour counter every day
      }
    }

    if ((LOWCURRENT == 1) && (ERROR == 0))
    {                                   // Blink LED0 at 1 Hz if in Lowcurrent
      if (TIME.sec % 2)
        LED0 = 0;                       // Turn on LED every odd second
      else
        LED0 = 1;                       // Turn on LED every even second
    }
  }
}

// END OF FILE
```