High-Speed Lithium Ion Battery Charger

Relevant Devices

This application note applies to the following device: C8051F300.

Introduction

Driven by the need for untethered mobility and ease of use, many systems rely on rechargeable batteries as their primary power source. The battery charger is typically implemented using a fixed-function IC to control the charging current/voltage profile.

The C8051F300 family provides a flexible alternative to fixed-function linear battery chargers. This note discusses how to use the C8051F300 device in Li-Ion battery charger applications. The Li-Ion charging algorithms can be easily adapted to other battery chemistries.

Key Points

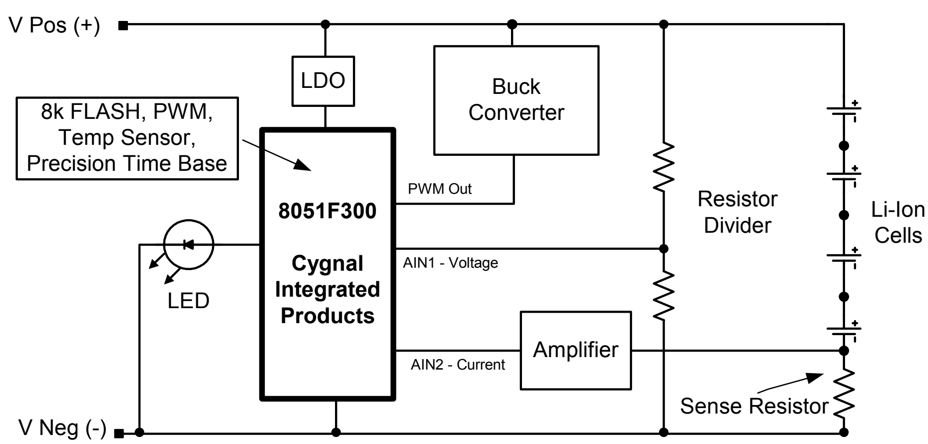
- On-chip high-speed ADC provides superior accuracy in monitoring charge voltage (critical to prevent overcharging in Li-Ion applications), maximizing charge effectiveness and battery life.
- On-chip comparator and PWM provides a means to implement a high speed buck converter with a small external inductor.
- On-chip temperature sensor provides an accurate and stable drive voltage for determining battery temperature. An external RTD (resistive temperature device) can also be accommodated.
- A single C8051F300 provides full product range for multi-chemistry chargers, expediting time to market and reducing inventory.

Charging Basics

Batteries are exhaustively characterized to determine safe yet time-efficient charging profiles. The optimum charging method for a battery is dependent on the battery's chemistry (Li-Ion, NiMH, NiCd, SLA, etc.). However, most charging strategies implement a 3-phase scheme:

1. Low-current conditioning phase
2. Constant-current phase
3. Constant-voltage phase/charge termination

Figure 1. Lithium Ion Battery Charger Block Diagram.



All batteries are charged by transferring electrical energy into them. The maximum charge current for a battery is dependent on the battery's rated capacity (C). For example, a battery with a cell capacity of 1000mAh is referred to as being charged at 1C (1 times the battery capacity) if the charge current is 1000mA. A battery can be charged at 1/50C (20 mA) or lower if desired. However, this is a common trickle-charge rate and is not practical in fast charge schemes where short charge-time is desired.

Most modern chargers utilize both trickle-charge and rated charge (also referred to as bulk charge) while charging a battery. The trickle-charge current is usually used in the initial phases of charging to minimize early self heating which can lead to premature charge termination. The bulk charge is usually used in the middle phase where the most of the battery's energy is restored.

During the final phase of battery charge, which generally takes the majority of the charge time, either the current or voltage or a combination of both are monitored to determine when charging is complete. Again, the termination scheme depends on the battery's chemistry. For instance, most Lithium Ion battery chargers hold the battery voltage constant, and monitor for minimum current. NiCd batteries use a rate of change in voltage or temperature to determine when to terminate.

While charging, some of the electrical energy is converted to thermal energy, until the battery reaches full charge, at which time all the electrical energy is converted to thermal energy. If charging isn't terminated, the battery can be damaged or destroyed. Fast chargers (chargers that charge batteries fully in less than two hours) compound this issue, as these chargers use a high charge current to minimize charge time. Therefore, monitoring a battery's temperature is critical especially for Li-Ion batteries which may explode if overcharged. Temperature is monitored during all phases and charge is terminated immediately if the temperature exceeds a preset maximum limit.

Hardware Description

Li-Ion batteries are currently the battery chemistry of choice for most applications due to their high energy/space and energy/weight characteristics when compared to other chemistries. Most modern linear Li-Ion chargers use the tapered charge termination, minimum current (see Figure 2) method to ensure the battery is fully charged, as does the example code provided at the end of this application note.

Buck Converter

The most economical way to create a tapered termination linear charger is to use a buck converter. A buck converter is a switching regulator that uses an inductor and/or a transformer (if isolation is desired), as an energy storage element to transfer energy from the input to the output in discrete packets. Feedback circuitry regulates the energy transfer via the transistor, also referred to as the pass switch, to maintain a constant voltage or constant current within the load limits of the circuit.

Figure 2. Lithium Ion Charge Profile.

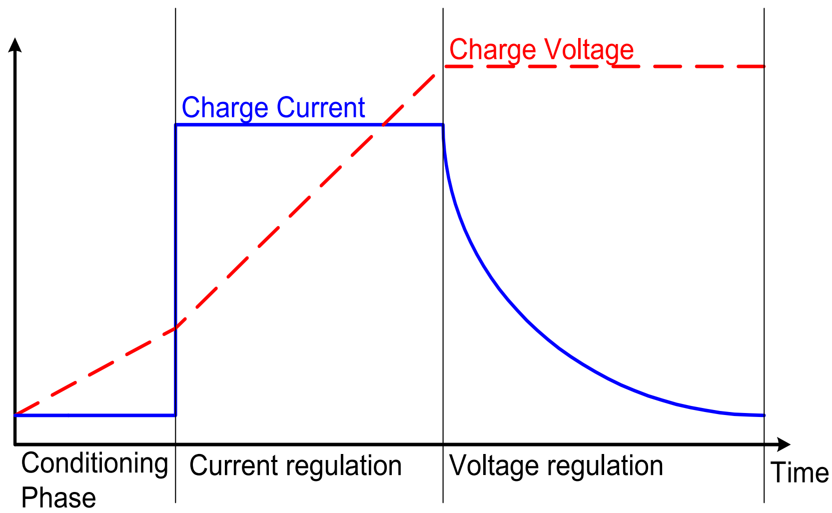
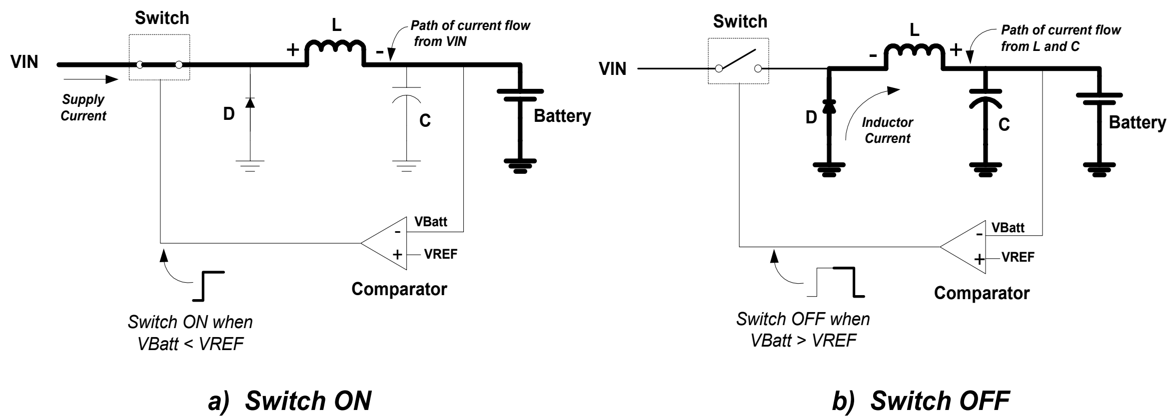


Figure 3. Buck Converter.



Buck Regulator Operation

The buck regulator operates by controlling the duty cycle of a transistor switch. The duty cycle is automatically increased to dispense more current into the battery. A comparator closes the switch when $V_{BATT} < V_{REF}$. As shown in Figure 3a, current flows into the battery and capacitor C. This current is also stored in inductor L. V_{BATT} rises until it exceeds V_{REF} at which time the comparator turns the switch off (Figure 3b). The current stored in the inductor rapidly decreases until diode D is forward biased, causing inductor current to flow into the battery at a decreasing rate. Capacitor C begins discharging after the inductor current has decayed and eventually V_{BATT} begins to fall. When V_{BATT} falls below V_{REF}, the comparator again turns the switch on and another cycle begins. On a larger scale, if the duty cycle is decreased (shorter "on" time), the average voltage decreases and vice versa. Therefore, controlling the duty cycle allows one to regulate the voltage or the current to within desired limits.

Selecting the Buck Converter Inductor

To size the inductor in the buck converter, one first assumes a 50 percent duty cycle, as this is where the converter operates most efficiently.

Duty cycle is given by Equation 1, where T is the period of the PWM (in our example T = 10.5µS).

$$\text{DutyCycle} = \frac{ton}{T}$$

Equation 1. Duty Cycle.

With this established, select a PWM switching frequency. As Equation 2 shows, the larger the PWM switching frequency, the smaller (and more cost effective) the inductor. Our example code configures the 'F300's hardware to generate a 510kHz switch rate.

$$L = \frac{(Vi - Vsat - Vo)ton}{2Iomax}$$

Equation 2. Inductor Size.

Now we can calculate the inductor's size. Assuming V_i, the charging voltage, is 15V, V_{sat}, the saturation voltage, is 0.5V, the desired output voltage, V_o, is 4.2V, and I_{0MAX}, the maximum output current, is 1500 mA, the inductor should be at least 4µH.

Note that the capacitor in this circuit is simply a ripple reducer. The larger it is the better as ripple is inversely proportional to the size of the capacitor.

High Speed Charger

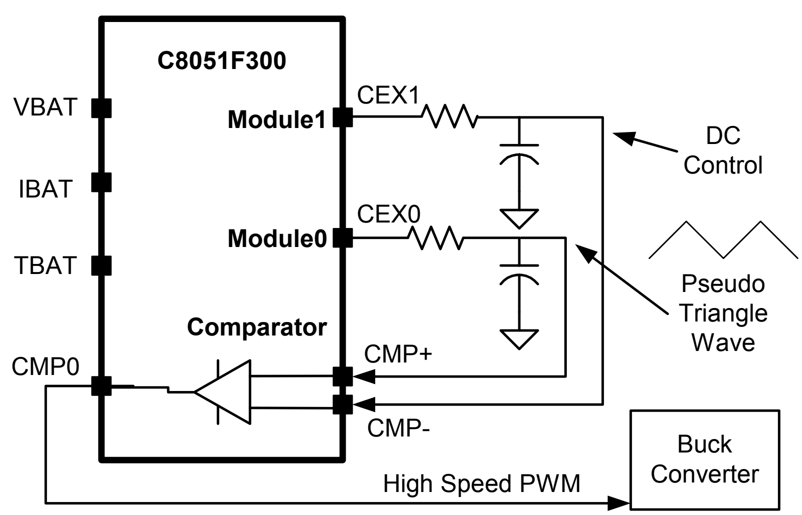
As AN037, *Lithium Ion Battery Charger Using C8051F300*, illustrates, the F300's 8-bit PWM can be configured to generate a 96kHz PWM with no external components. This PWM output can be used to drive the pass switch in a buck converter and charge a battery. However, a 96kHz frequency requires a buck converter to utilize a relatively large 18µH inductor. For some applications, this is too large and costs too much. To reduce the size and cost of this inductor requires that the switch rate of the buck converter increase. The beauty of the F300 lies in its flexible feature set. As mentioned, included in the device is a PCA (Programmable Counter Array) that has three 16-bit capture/compare modules with corresponding output drives that can be configured to provide numerous functions. We can use two of the PCA's modules, along with two external single-pole low-pass filters, and the on-chip comparator to generate an 8-bit, 510kHz PWM (refer to Figure 4 for details). By setting the switch rate to 510kHz, the inductor required to satisfy the buck converter equations is reduced by a factor of five to approximately 4µH.

To create a 510kHz PWM with the F300, Module 0 of the PCA is configured to provide a 510kHz square wave via the Frequency Output Mode. This square wave is then filtered through a low-pass filter with a 500kHz corner frequency to provide approximately a 2 Volt peak-to-peak pseudo triangle wave to the positive input of the on-chip comparator. For the minus input of the comparator, Module 1 is configured as an 8-bit PWM at 96kHz switch rate. This PWM output is then low pass filtered with a corner frequency of approximately 15Hz to create a simple DC digital-to-analog converter. By comparing the pseudo triangle wave to a DC input, the output of the comparator's output becomes a 510kHz PWM.

The DC control path, Module1's output in this example, controls the duty cycle of the 510kHz PWM output from the comparator. By varying the duty cycle of the 8-bit 96kHz PWM, the minus input to the comparator can be varied from 0 volts to the supply, typically 3.3V. The accuracy of the DC control path is limited by the settling time of the external RC filter. For this example, components were selected to minimize errors contributed from this path. For more details on component selection for the DC path, refer to AN010, *16-Bit PWM Using an On-Chip Timer.*

The overall accuracy of the 510kHz PWM output from the comparator is mostly limited by the pseudo triangle path to the comparator. Assuming the DC path is error free, to create a true 8-bit 510kHz PWM output from the comparator requires that a perfectly linear full-scale triangle wave be input to the positive input of the comparator. A true full scale triangle wave refers to a triangle wave that linearly ramps from 0 volts to the positive supply and then returns back the negative supply in a similar fashion. However, the charge and discharge profile of a capacitor in the low pass configuration is not linear past a time constant. Moreover, it is desirable not to allow this capacitor to fully charge as the pseudo triangle wave becomes more nonlinear towards its peaks. Unfortunately, limiting the overall charge time/voltage limits the overall accuracy of the 510kHz PWM. For example, if we compare a triangle wave with a peak-to-peak voltage of 1/4 the supply to the DC control path's voltage, we could generate a 2-bit 510kHz PWM output from the comparator. In practical applications, a 2-bit 510kHz has very limited use. To improve the accuracy requires either one of two changes: 1) increase the voltage of the pseudo triangle wave or 2) increase the resolution of the DC path's PWM control. As mentioned earlier, increasing the peak-to-peak voltage of the pseudo triangle wave can be easily accomplished by adjusting its low pass filter components accordingly. Our example is designed

Figure 4. High Speed Charger.



to provide approximately a 2 volt peak-to-peak pseudo triangle wave. When compared to the 8-bit PWM dc control path, we achieve approximately 7.5-bits of performance at 510kHz switch rate.

The overall resolution of the high speed PWM output can be increased very easily by configuring Module 1 to a 16-bit PWM. On an aside, if a faster PWM output, greater than 510kHz, is desired to reduce the inductor size further, the user can reconfigure Module 0 to provide a faster square wave up to ½ the internal oscillator frequency or approximately 12MHz. If this is desired, the external low pass filter for the pseudo triangle wave path will have to be modified to accommodate the faster square wave. Other limitations, like comparator speed and voltage induced across the inductor due to the higher current transients will also have to be considered.

Software Description

The software example that follows demonstrates a Li-Ion battery charger using the C8051F300. The algorithms discussed are written entirely in "C" making them easily portable. Refer to the F300's datasheet for a full description of the device. Note that the software architecture for the low speed (96kHz) charger discussed in AN037 and the high speed charger (510kHz) are essentially the same (i.e. the flow charts that follow can be used for either hardware configuration). The main difference are the control mechanisms. For the slow speed charger in AN037, Module 0 (CEX0) is used to control the duty cycle. For the high speed charger Module 1 (CEX1) is used to control the duty cycle.

Calibration

To ensure accurate voltage and current measurements, the algorithms use a two-point system calibration scheme. In this scheme, the user is expected to apply two known voltages and two known currents, preferable, one point near ground and the other point near full-scale. The algorithm then takes these two points, calculates a slope and an offset for both the current and voltage channels, and stores the results in FLASH. All future conversions are scaled relative to these slope and offset calculations.

Temperature

To monitor the temperature, the algorithms use the on-chip temperature sensor. The sensor is left uncalibrated, but still provides a sufficiently accurate temperature measurement. For more accurate temperature measurement, one or two-point temperature calibration is required.

An external temperature sensor can be used if desired. The AMUX can be reconfigured to accommodate this additional input voltage.

Current

The current delivered to the battery cells is monitored by measuring the voltage across a sense resistor (typically tens of mohms; our example uses a 0.24 ohm resistor). To maximize current measurement accuracy, this reference design uses an external amplifier with a gain of 10. This provides about 11-bits of current measurement accuracy (8-bits from the ADC and 3-bits from the external gain amplifier). To further maximize current measurement accuracy, the raw current measurements are scaled via the slope and offset calibration coefficients every time a measurement conversion is taken.

To determine the minimum current resolution, recall that the output code of a ADC is given by Equation 3

$$Dout = \frac{(Ain)2^n}{Vref}$$

Equation 3. Digital Output Code.

Accounting for the external amplifier, Equation 4

$$Ain = (Iin \times Rs \times 10)$$

Equation 4. Input Current with 10x Gain.

states that Ain is

Iin is then given by Equation 5

$$Iin = \frac{(Dout \times Vref)}{2^n \times 10 \times Rs}$$

Equation 5. Input Current.

Assuming

- $Rs = 0.24\ \Omega$
- $V_{REF} = 3.3$ V
- $2^N = 256$, an 8-bit converter
- External Gain = 10
- No External Gain = 1

When Dout = 1, I_{MIN} is given by Equation 6.

$$Imin = (5.37)\frac{mA}{Code}$$

Equation 6. I_{MIN}.

When Dout=256, Imax is given by Equation 7.

$$Imax = (1.37)Amps$$

Equation 7. Imax.

It is important to note that if one chooses to modify the algorithm, the order of mathematical operations is important. To minimize truncation error effects, be sure to perform multiply operations first making the numerator as large as possible, before performing divide operations. Further recall that a *long* type variable, which is the limit for the F300's compiler, is limited to 2^{32}-1 or approximately 4 billion.

Voltage

The battery's voltages are divided down and monitored via external resistors. Note that this example uses the supply voltage as the ADC voltage reference. Any monitored voltage above the reference voltage must be divided down for accurate monitoring. If a more accurate reference is required, an external voltage reference can be used. Adjustment to the divide resistors must be made accordingly.

Charging - Phase1

In Phase 1, (for description purposes, we assume the battery is initially discharged), the 'F30x regulates the battery's current to $I_{LOWCURRENT}$ (typically 1/50 C) until the battery's voltage reaches $V_{MINVOLTBULK}$. Note that the battery's charge current is current limited to $I_{LOWCURRENT}$ to ensure safe initial charge and to minimize battery self-heating. If at any time the temperature increases out of limit, charging is halted.

Charging - Phase 2

Once the battery reaches $V_{MINVOLTBULK}$ the charger enters Phase 2, where the battery's algorithm controls the PWM pass switch to ensure the output voltage provides a constant charge-current I_{BULK} to the battery (rate or bulk current is usually 1C and is definable in the header file as is $I_{LOWCURRENT}$ and $V_{MINVOLTBULK}$).

Charging - Phase 3

After the battery reaches V_{Top} (typically 4.2 V in single cell charger), the charger algorithm enters Phase 3, where the PWM feeds back and regulates the battery's voltage. In Phase 3, the battery continues to charge until the battery's charge current reaches $I_{MINIBULK}$, after which, the battery is charged for an additional 30 minutes and then charge terminates. Phase 3 typically takes the majority of the charging time.

Note that in most practical applications, such as a portable PC, the batteries may be in any of the three phases when charging is activated. This doesn't really affect the charger as it simply monitors the battery's current condition and starts charging from that point.

Getting Started

The reference design that accompanies this application note is designed to charge a single cell 4.2 V lithium ion battery. To accommodate numerous power supplies and batteries, it charges at 250mA bulk current. To charge a battery, first connect power to the board by applying an 8V to 15V supply to JP1. The power supply should be able to supply a minimum of 500mA. Once the appropriate supply is connected, connect the battery to JP3. Connect the positive lead of the battery to pin 1 and connect the negative lead to pin 3. Terminal 2 of JP3 can be left unconnected as no code has been developed to monitor temperature via an external temperature sensor at this time. Finally, press the reset switch and the battery will begin charging. The PWM charge control signal can be monitored by probing pin 5 on the C8051F300.

Recommended Operating Conditions

- Supply: 8V to 15V
- Battery: One cell, 4.2V,
  with <1000mAh rating

Default Charging Parameters

- Trickle Current = 135mA
- Bulk Current = 250mA
- Regulation Voltage = 4.2V
- Termination Current = 125mA

Efficiency of Charger

- Switching Efficiency > 80%
- Voltage Accuracy > 1%
- Current Accuracy > 2%

Charging 2 Cells or More

To charge more than 1 battery cell, both the hardware and software will need to be modified. For example, to charge two 4.2 V batteries simultaneously, resistors R11 and R12 will need to be switched. Then, the header file will need these modifications:

CELLs = 2

RESB = 10

Once the header file is modified, recompile the software and down load the new source code to the charger board. A similar scheme can be used to modify the board for any number of cells.

Conclusion

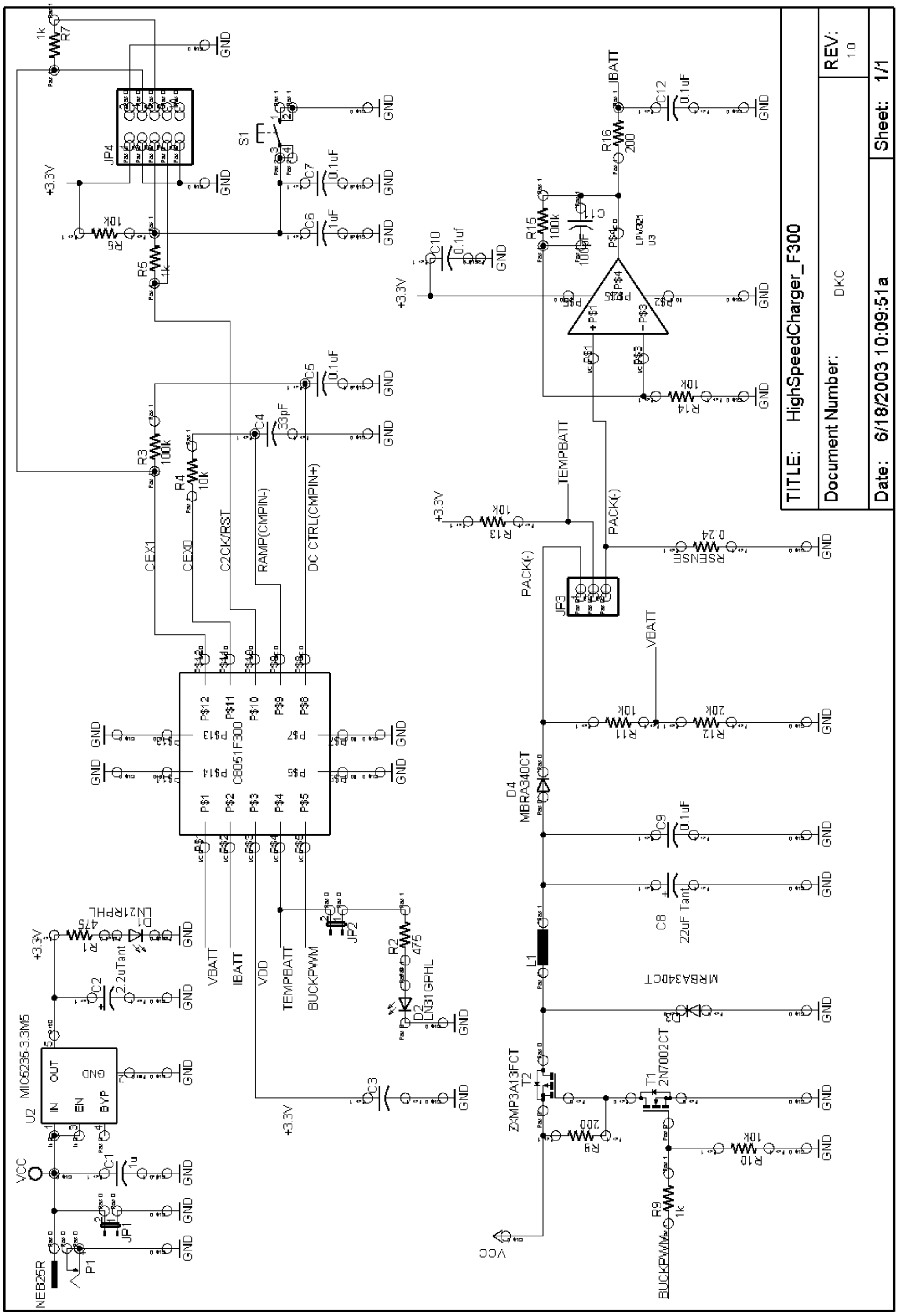
The C8051F300's high level of analog integration, small form-factor, integrated FLASH memory, and low power consumption make it ideal for flexible next generation battery charging applications. This note discussed how to use the C8051F300 in Lithium Ion battery charger applications at 510kHz switch rate. Example code is provided as well.

Reference

Applications of Linear Integrated Circuits. Eugene Hnatek, John Wiley and Sons, 1975.

Appendix A - Schematic

Figure 5. High Speed Charger Schematic.



Appendix B - Bill Of Materials

Figure 6. High Speed Charger Bill of Materials.

Item	QTY	Part	Value	Package	Notes
1	1	U1	C8051F300	MLP-11	Cygnal Integrated Products C8051F300
2	1	U3	LPV321M5	SOT-23-5	National (LPV321M5 or equivalent)
3	1	Q1	N-Channel	SOT-23	Zetex, N-Channel 30-V (D-S) MOSFET, (2N7002CT or equivalent)
4	1	Q2	P-Channel	SOT-23	Zetex, P-Channel 30-V (D-S) MOSFET, (ZXMP3A13FCT-ND or equivalent)
5	1	L1	22uH	SMD	Coil Craft Inductor, 22uH, 1.5 A, (DO3316P-223 or equivalent)
6	2	D3,4	Schottky	SMC	3A 40V Power Rectifier Diode (MBRS340CT or equivalent)
7	6	C3,5,7,9,10,12	0.1uF	0805	Cap X7R 50V 5% (Kemet C0805C104J5RACTU or equivalent)
8	1	C4	33pF	0805	Cap X7R, 50V 10% (Kemet C0805330J5GACTU or equivalent)
9	1	C6	1 uF	0805	Cap X7R, 10V 10% (Kemet C0805105K8RACTU or equivalent)
10	1	C8	22uF	EIA6032-28	Cap Tantalum, 16V, 10% (Kemet T491C226K016AS or equivalent)
11	1	C11	100pF	0805	Cap X7R 50V 5% (Kemet C0805C101K5GACTU or equivalent)
12	2	R3,15	100k	0805	Resistor 1/10W, 5% (Panasonic P100KCCT-ND or equivalent)
13	5	R4,10,11,13,14	10k	0805	Resistor 1/10W, 5% (Panasonic P10.0KCCT-ND or equivalent)
14	1	R9	1k	0805	Resistor 1/10W, 5% (Panasonic P1.0KCCT-ND or equivalent)
15	2	R8,16	200	0805	Resistor 1/10W, 5% (Panasonic P200CCT-ND or equivalent)
16	1	R12	20k	0805	Resistor 1/10W, 5% (Panasonic P20.0KCCT-ND or equivalent)
17	1	S1	Switch	6MM, SQ	Momentary switch (Panasonic P8007S-ND or equivalent)
18	1	RSENSE	0.24 ohm	0805	Resistor 1/4W, 2% (Panasonic RL12T0.24GCT or equivalent)
			BOM TOTAL		
Included on Demo Board, but NOT Part of Battery Charger BOM					
Item	QTY	Part	Value	Package	Notes
1	1	U2	MIC5235	SOT-23-5	Micrel Semiconductor (MIC5235-3.3M5)
2	1	C1	1 uF	0805	Cap X7R, 10V 10% (Kemet C0805105K8RACTU or equivalent)
3	1	C2	2.2 uF	EIA3216-18	Cap Tantalum, 16V, 10% (Kemet T491A225K016AS or equivalent)
4	2	R1,R2	475 ohm	0805	Resistor 1/10W, 5% (Panasonic P475CCT-ND or equivalent)
5	2	R5,7	1k	0805	Resistor 1/10W, 5% (Panasonic P1.0KCCT-ND or equivalent)
6	1	R6	10k	0805	Resistor 1/10W, 5% (Panasonic P10.0KCCT-ND or equivalent)
7	1	D1	LED, Red	0.1" thru hole	T-1 3/4 (Panasonic LN21RPHL or equivalent)
8	1	D2	LED, Green	0.1" thru hole	T-1 3/4 (Panasonic LN31GPHL or equivalent)
9	1	Shunt	Shunt	0.1"	Shunt (929957-08 or equivalent)
10	2	JP1,JP2	1x2 Header	0.1" thru hole	Sullins (S2105-02 or equivalent)
11	1	JP3	1x3 Header	0.1" thru hole	Sullins (S2105-03 or equivalent)
12	1	JP4	2x5 Header	0.1" thru hole	Protected with central polarizing key slot (3M 2510-6002UB or equiv.)
13	1	P1	RAPC722	2x5.5mm Jack	Switchcraft (SC1153-ND or equivalent)
14	1	Board	2-Layer	2"x1.75"	PCBEXPRESS (board manufacturing services)

Appendix C - PCB Layout

Figure 7. High Speed Charger Layout (Silk Screen).

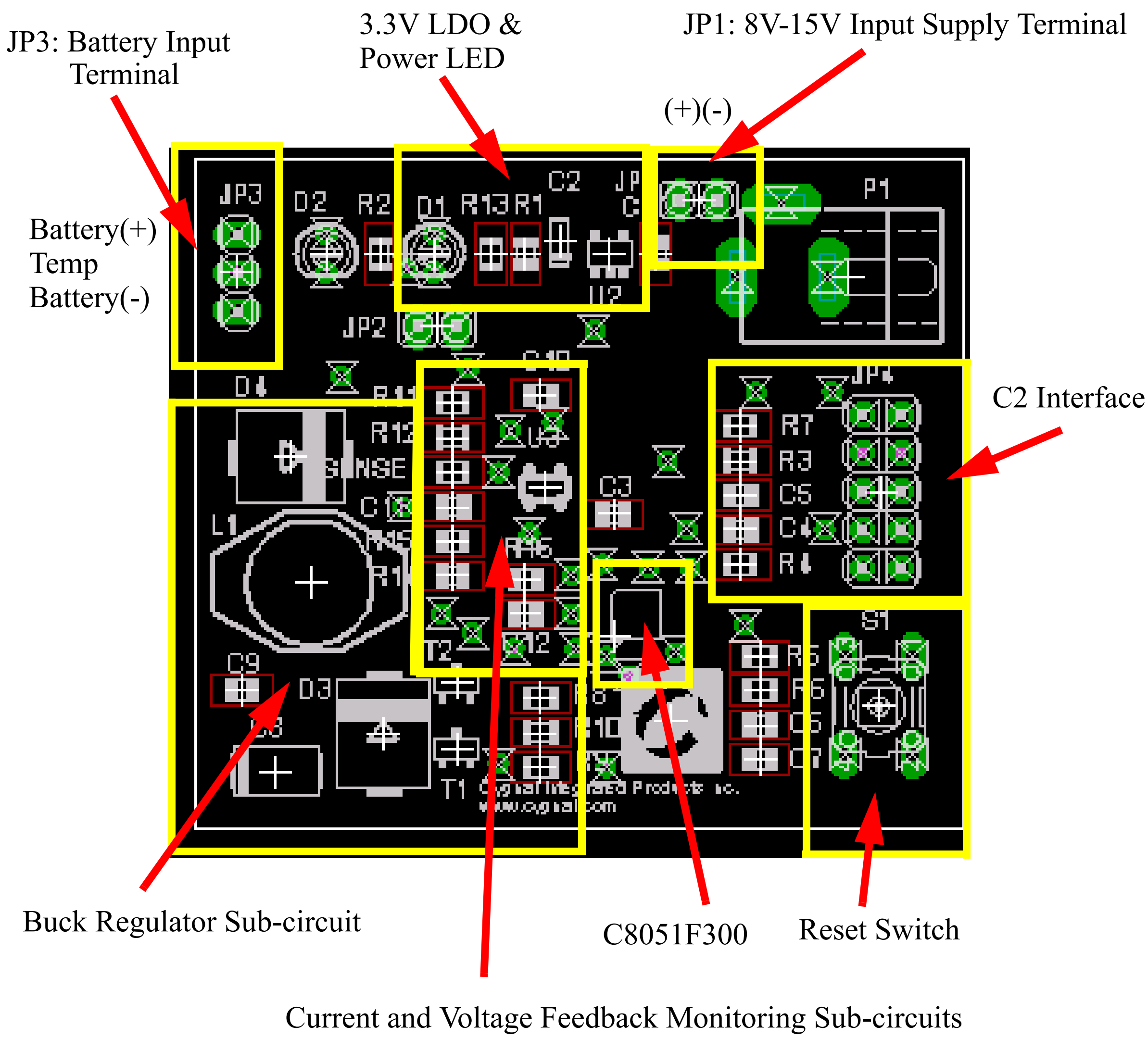


Figure 8. High Speed Charger Layout (Top Layer).

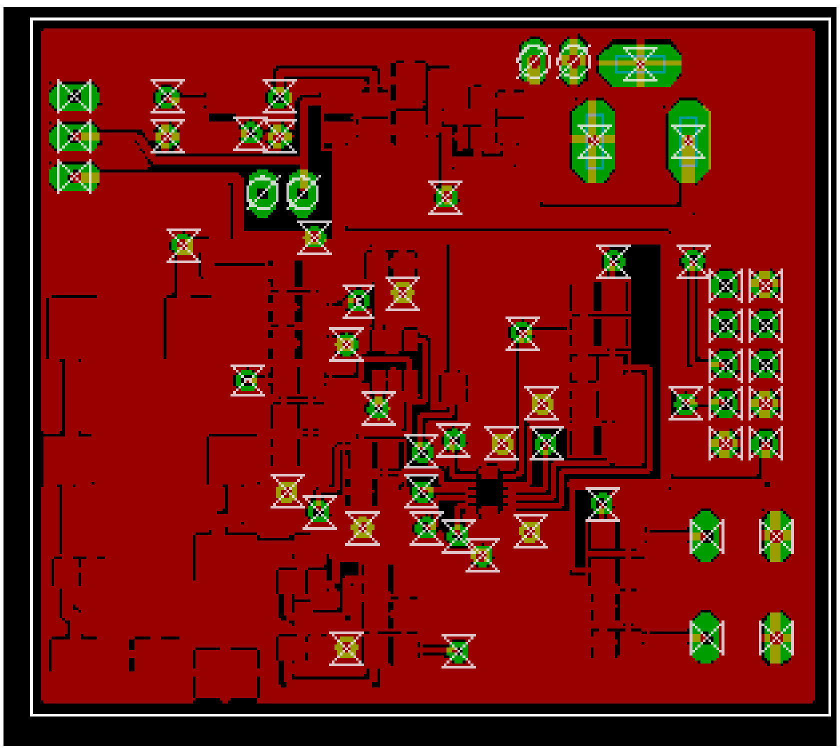


Figure 9. High Speed Charger Layout (Bottom Layer).

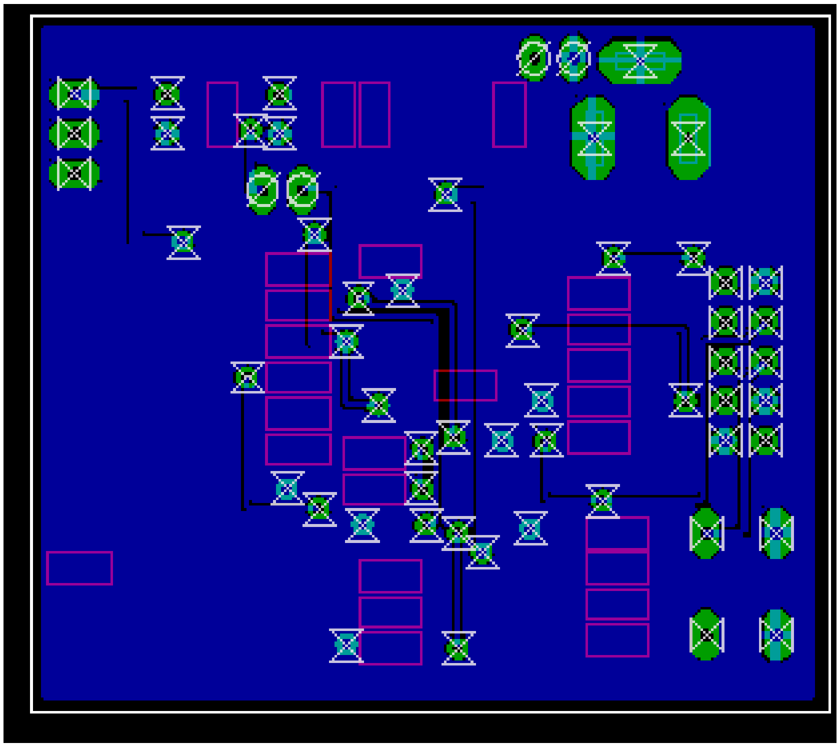


Figure 10. High Speed Charger Bill of Materials.

Item	QTY	Part	Value	Package	Notes
1	1	U1	C8051F300	MLP-11	Cygnal Integrated Products C8051F300
2	1	U3	LPV321M5	SOT-23-5	National (LPV321M5 or equivalent)
3	1	Q1	N-Channel	SOT-23	Zetex, N-Channel 30-V (D-S) MOSFET, (2N7002CT or equivalent)
4	1	Q2	P-Channel	SOT-23	Zetex, P-Channel 30-V (D-S) MOSFET, (ZXMP3A13FCT-ND or equivalent)
5	1	L1	22uH	SMD	Coil Craft Inductor, 22uH, 1.5 A, (DO3316P-223 or equivalent)
6	2	D3,4	Schottky	SMC	3A 40V Power Rectifier Diode (MBRS340CT or equivalent)
7	6	C3,5,7,9,10,12	0.1uF	0805	Cap X7R 50V 5% (Kemet C0805C104J5RACTU or equivalent)
8	1	C4	33pF	0805	Cap X7R, 50V 10% (Kemet C0805330J5GACTU or equivalent)
9	1	C6	1 uF	0805	Cap X7R, 10V 10% (Kemet C0805105K8RACTU or equivalent)
10	1	C8	22uF	EIA6032-28	Cap Tantalum, 16V, 10% (Kemet T491C226K016AS or equivalent)
11	1	C11	100pF	0805	Cap X7R 50V 5% (Kemet C0805C101K5GACTU or equivalent)
12	2	R3,15	100k	0805	Resistor 1/10W, 5% (Panasonic P100KCCT-ND or equivalent)
13	5	R4,10,11,13,14	10k	0805	Resistor 1/10W, 5% (Panasonic P10.0KCCT-ND or equivalent)
14	1	R9	1k	0805	Resistor 1/10W, 5% (Panasonic P1.0KCCT-ND or equivalent)
15	2	R8,16	200	0805	Resistor 1/10W, 5% (Panasonic P200CCT-ND or equivalent)
16	1	R12	20k	0805	Resistor 1/10W, 5% (Panasonic P20.0KCCT-ND or equivalent)
17	1	S1	Switch	6MM, SQ	Momentary switch (Panasonic P8007S-ND or equivalent)
18	1	RSENSE	0.24 ohm	0805	Resistor 1/4W, 2% (Panasonic RL12T0.24GCT or equivalent)
Included on Demo Board, but NOT Part of Battery Charger BOM					
Item	QTY	Part	Value	Package	Notes
1	1	U2	MIC5235	SOT-23-5	Micrel Semiconductor (MIC5235-3.3M5)
2	1	C1	1 uF	0805	Cap X7R, 10V 10% (Kemet C0805105K8RACTU or equivalent)
3	1	C2	2.2 uF	EIA3216-18	Cap Tantalum, 16V, 10% (Kemet T491A225K016AS or equivalent)
4	2	R1,R2	475 ohm	0805	Resistor 1/10W, 5% (Panasonic P475CCT-ND or equivalent)
5	2	R5,7	1k	0805	Resistor 1/10W, 5% (Panasonic P1.0KCCT-ND or equivalent)
6	1	R6	10k	0805	Resistor 1/10W, 5% (Panasonic P10.0KCCT-ND or equivalent)
7	1	D1	LED, Red	0.1" thru hole	T-1 3/4 (Panasonic LN21RPHL or equivalent)
8	1	D2	LED, Green	0.1" thru hole	T-1 3/4 (Panasonic LN31GPHL or equivalent)
9	1	Shunt	Shunt	0.1"	Shunt (929957-08 or equivalent)
10	2	JP1,JP2	1x2 Header	0.1" thru hole	Sullins (S2105-02 or equivalent)
11	1	JP3	1x3 Header	0.1" thru hole	Sullins (S2105-03 or equivalent)
12	1	JP4	2x5 Header	0.1" thru hole	Protected with central polarizing key slot (3M 2510-6002UB or equiv.)
13	1	P1	RAPC722	2x5.5mm Jack	Switchcraft (SC1153-ND or equivalent)
14	1	Board	2-Layer	2"x1.75"	PCBEXPRESS (board manufacturing services)

Figure 11. main() Flow Chart.

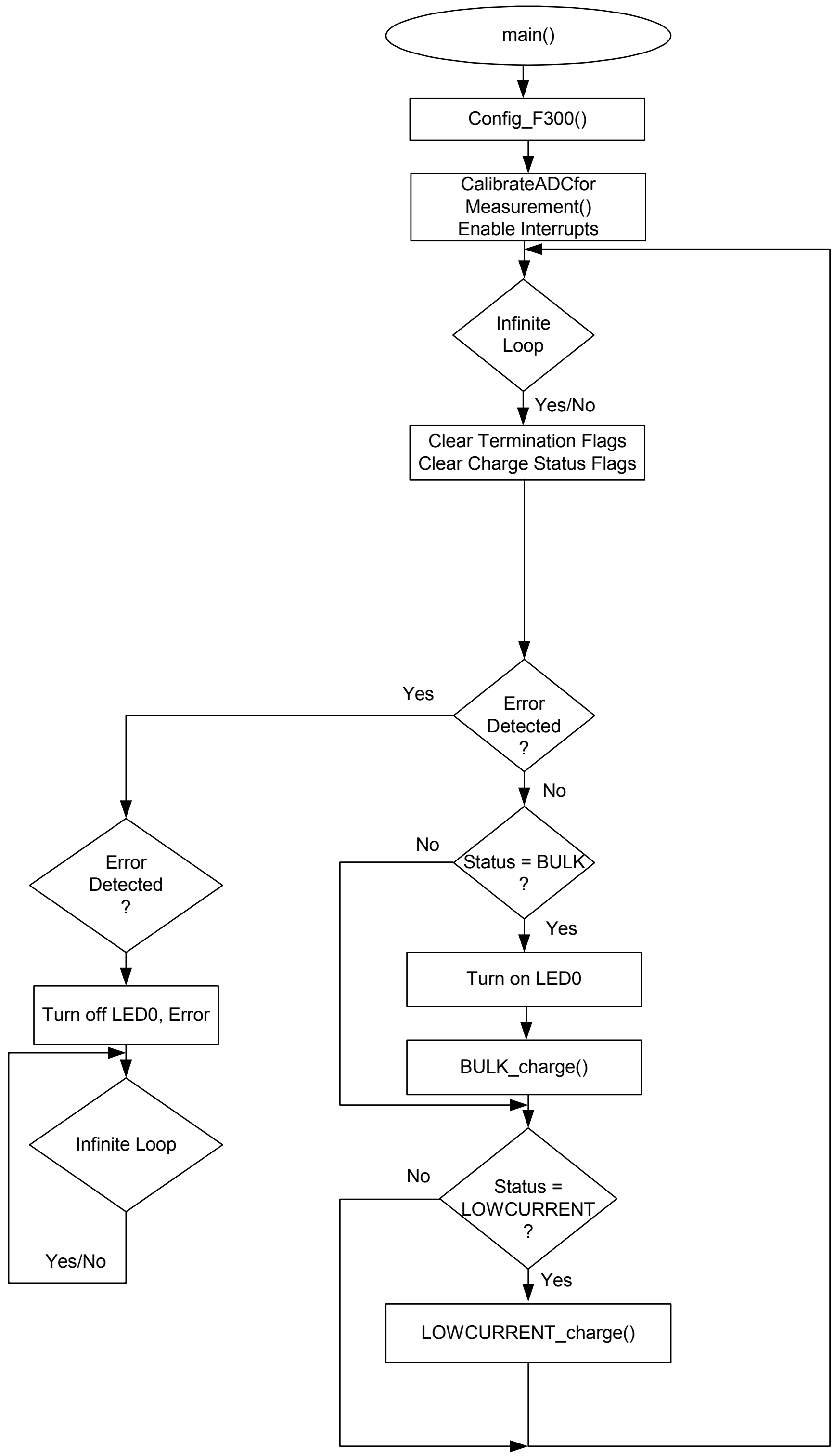
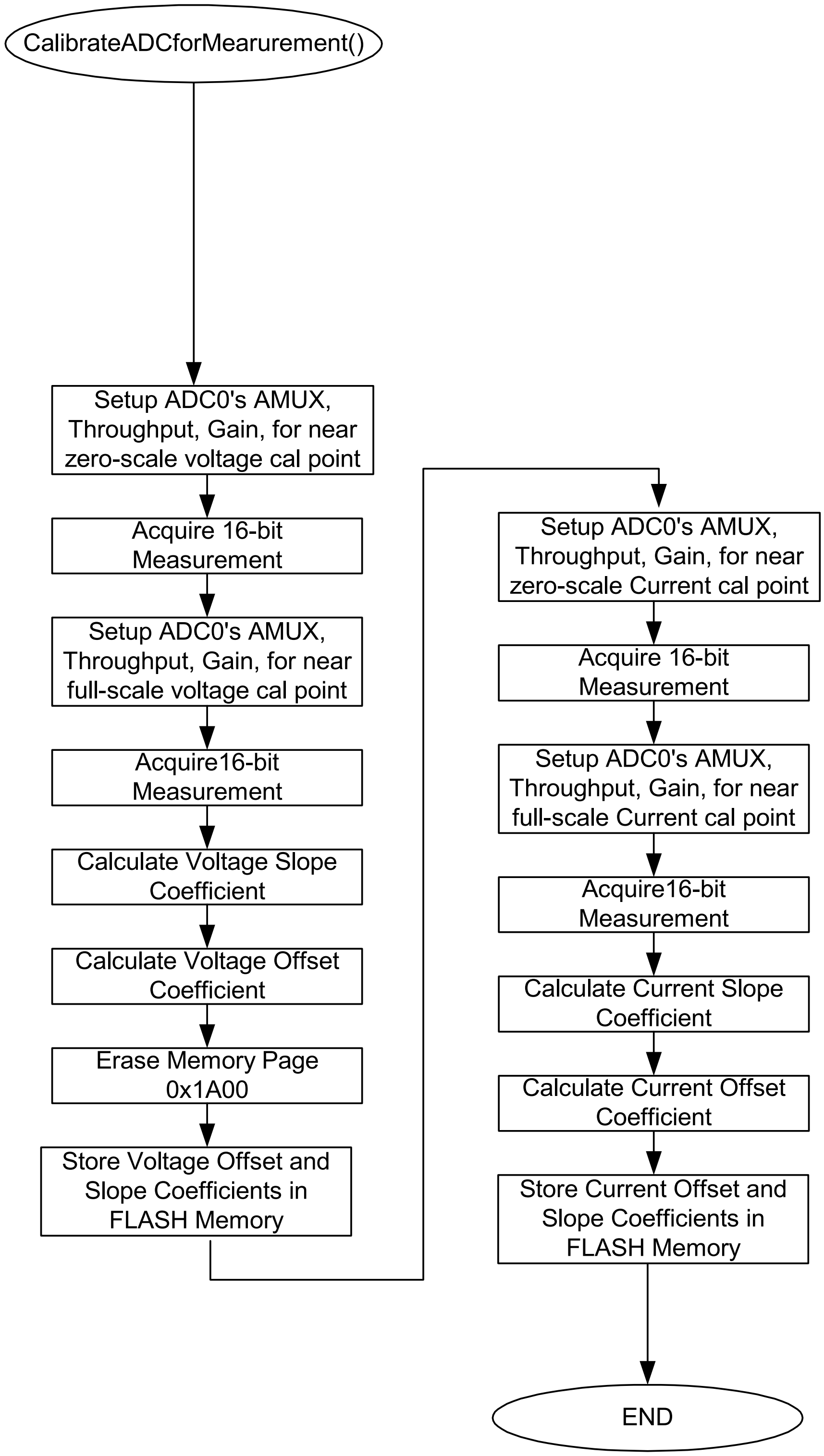


Figure 12. CalibrateADCforMeasurement() Flow Chart.



Appendix D - Software Flow Charts

Figure 13. Monitor_Battery() Flow Chart.

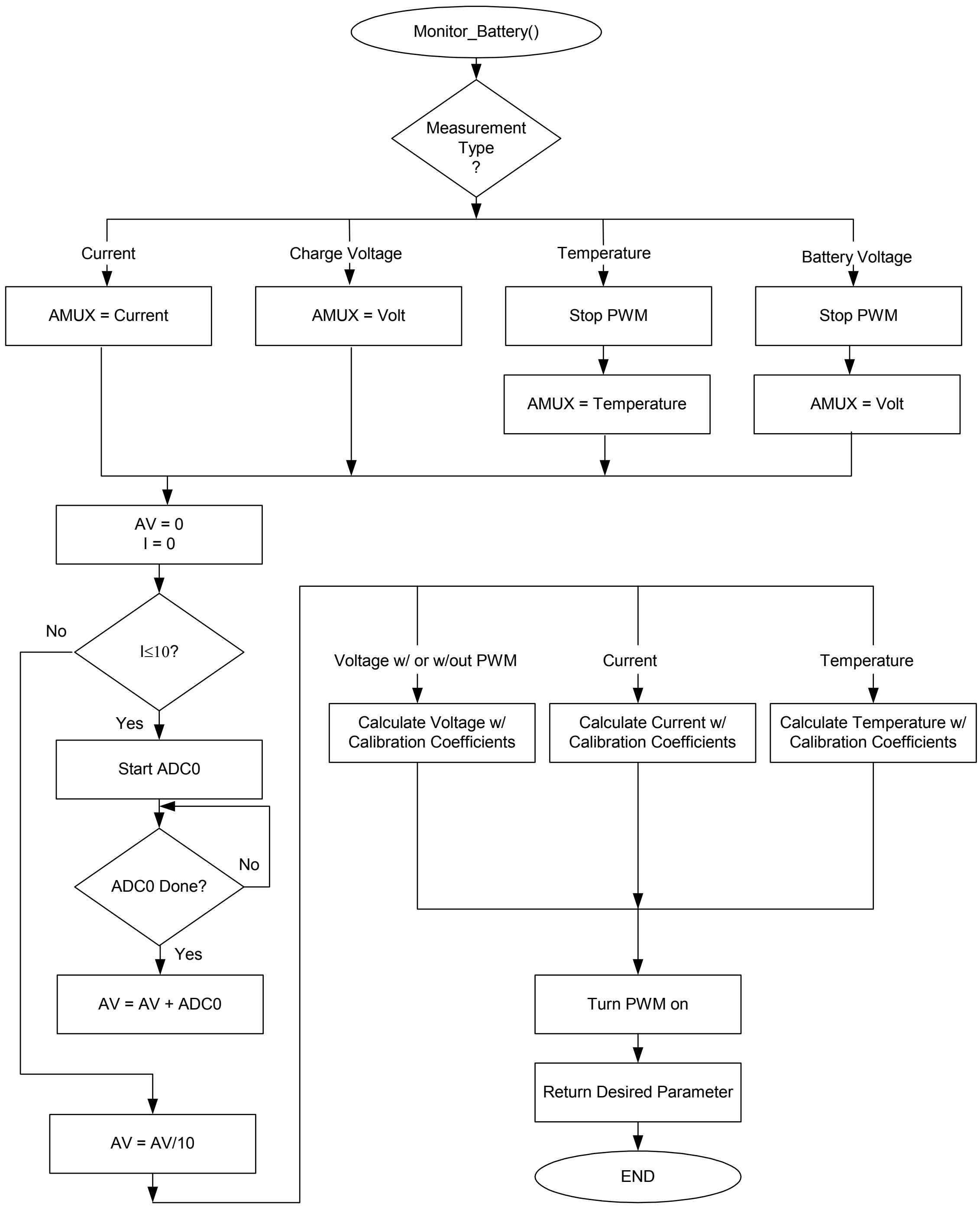


Figure 14. Bulk_Charge() Flow Chart (Part 1).

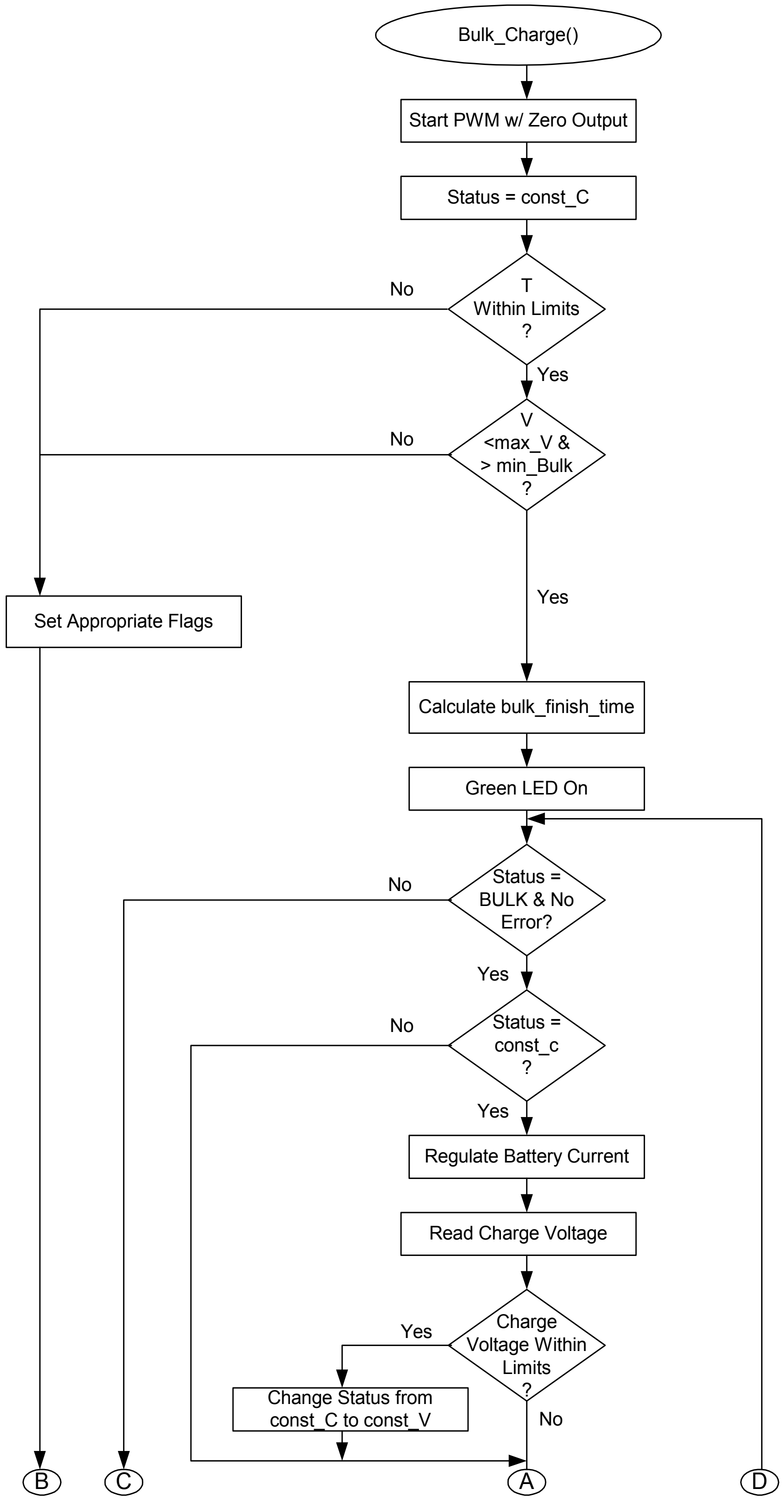


Figure 15. BULKCurrent() Flow Chart (Part 2).

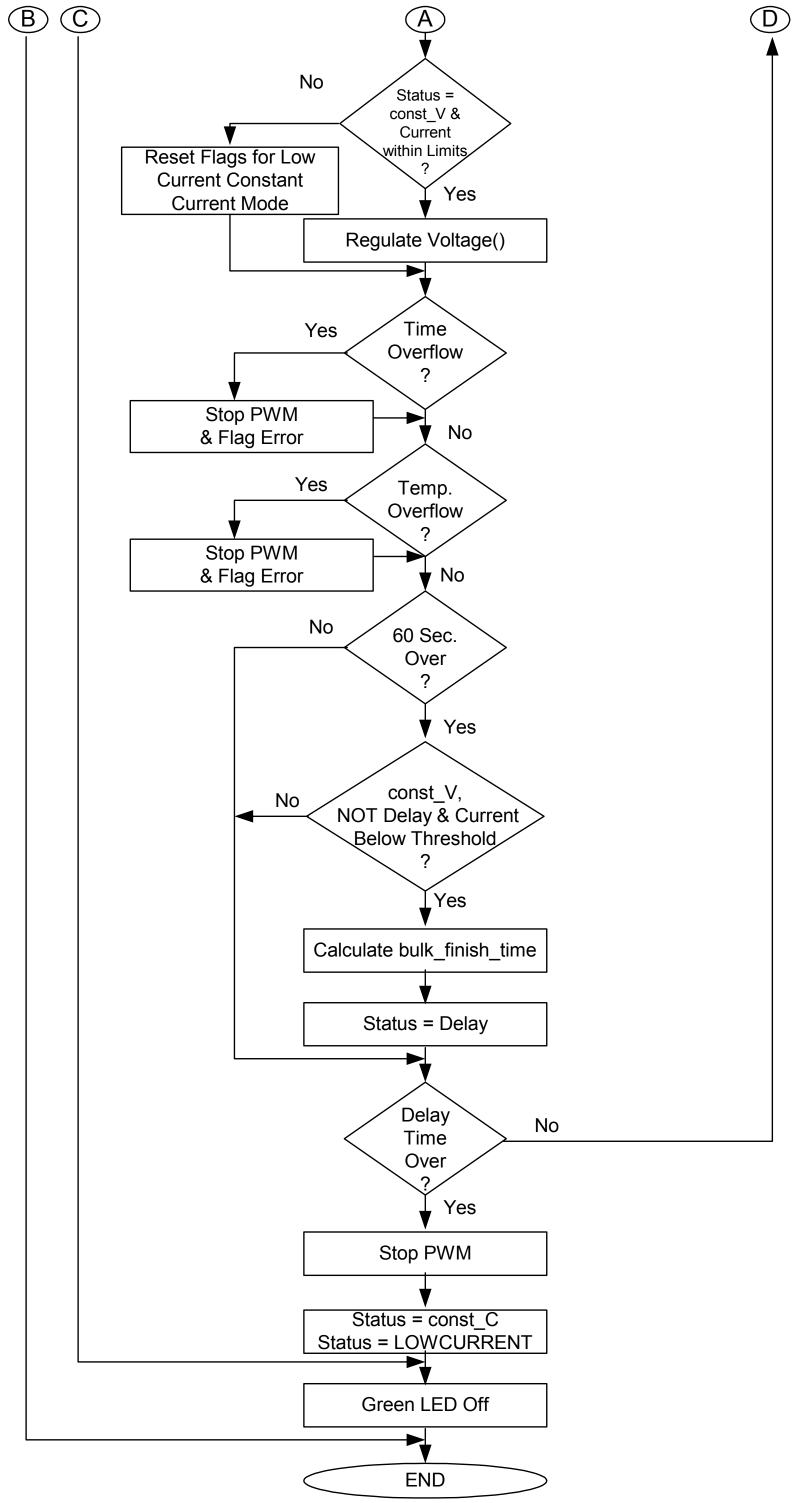


Figure 16. LowCurrent_Charge() Flow Chart.

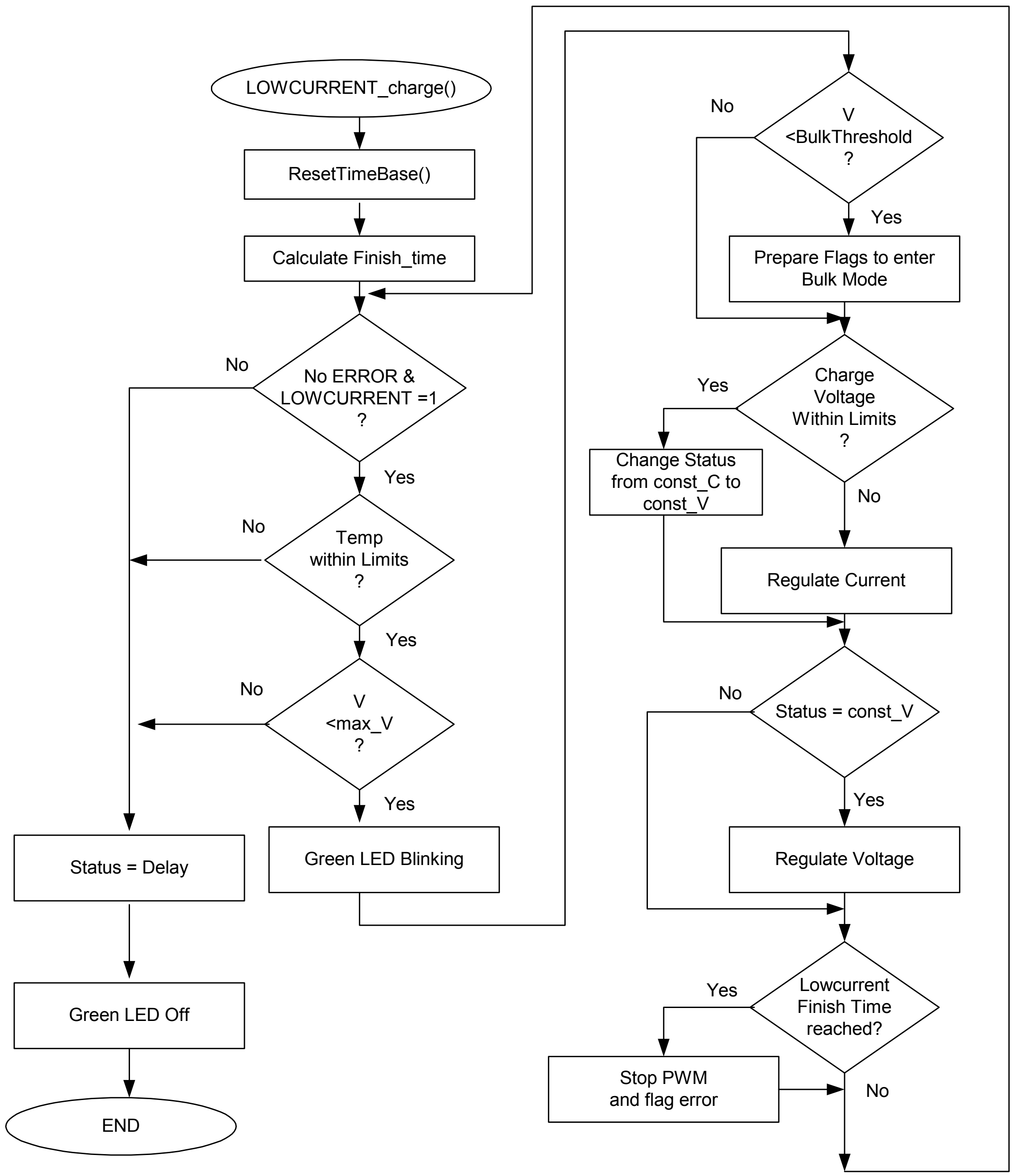


Figure 17. Turn_PWM_Off() Flow Chart.

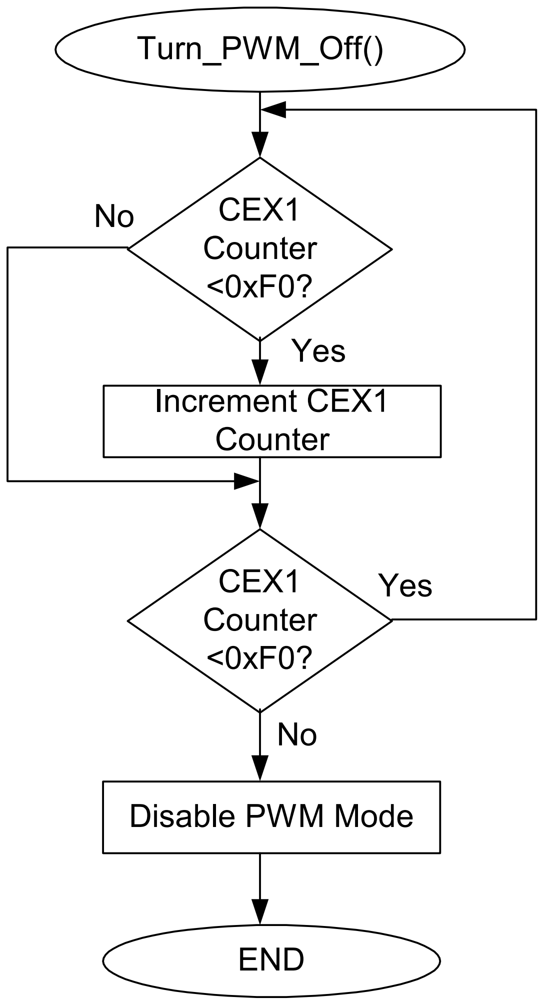


Figure 18. Measure() Flow Chart.

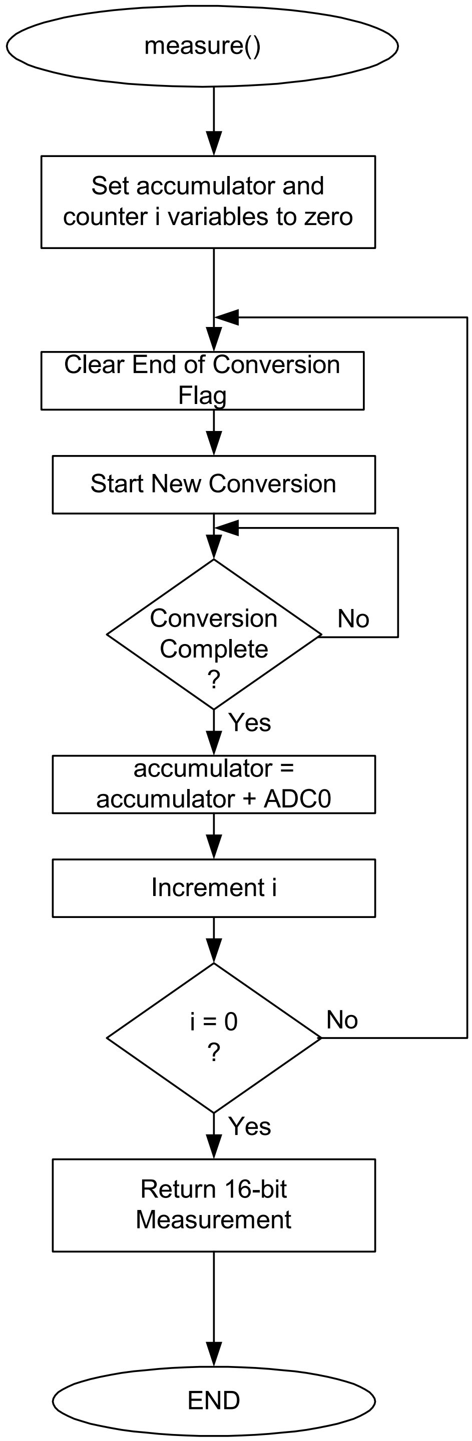


Figure 19. Regulate_Voltage() Flow Chart.

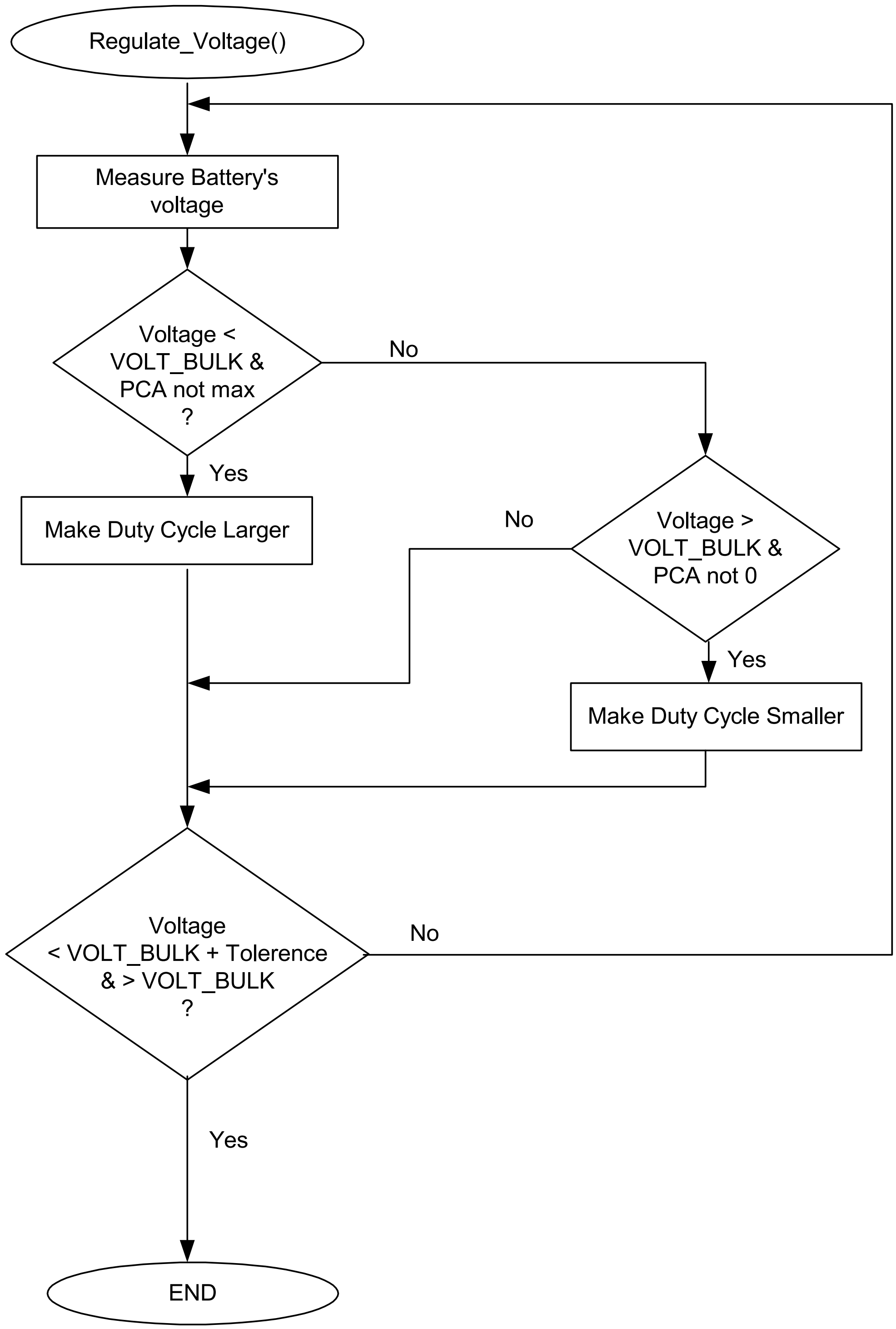


Figure 20. Regulate_Current() Flow Chart.

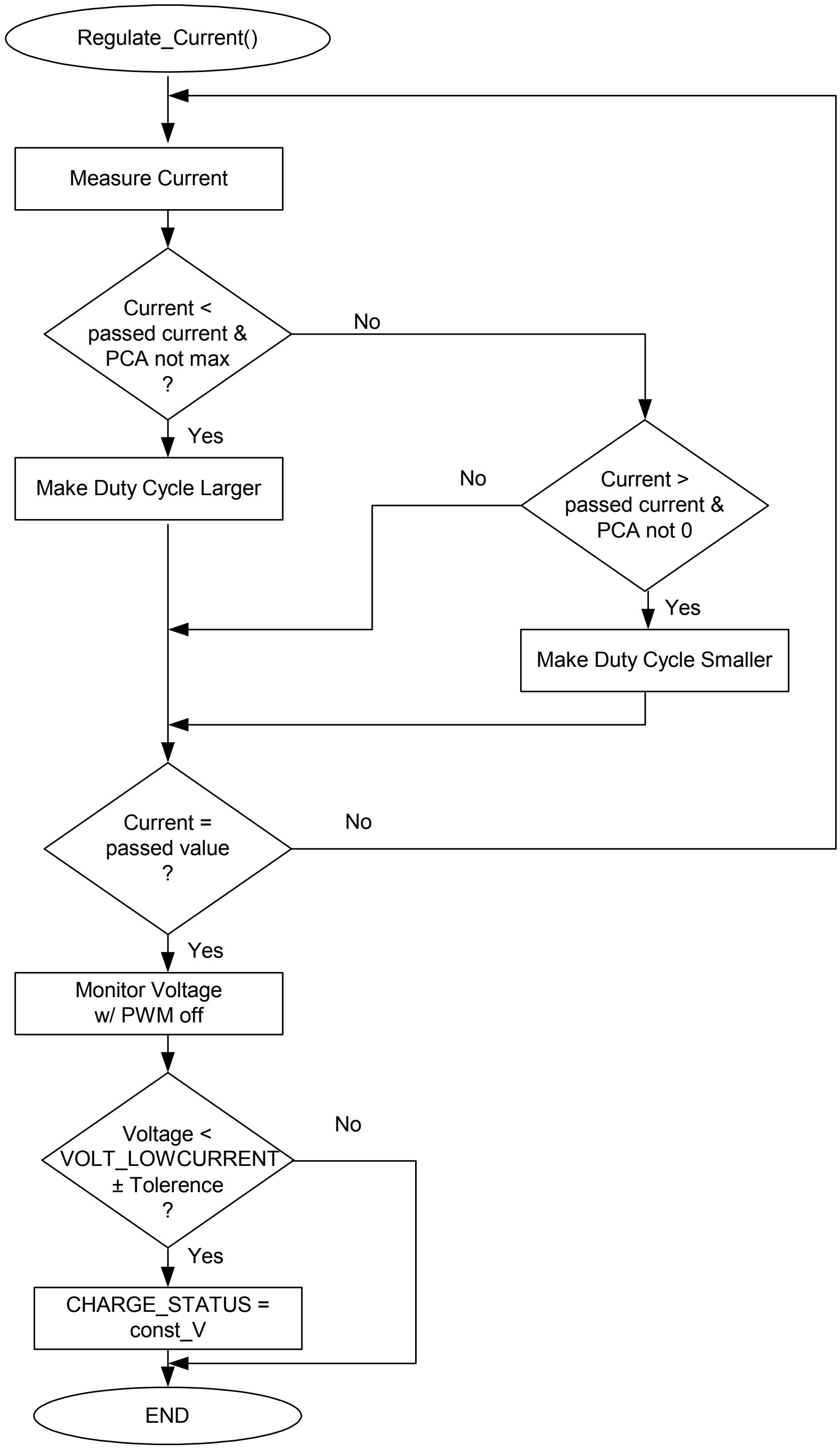
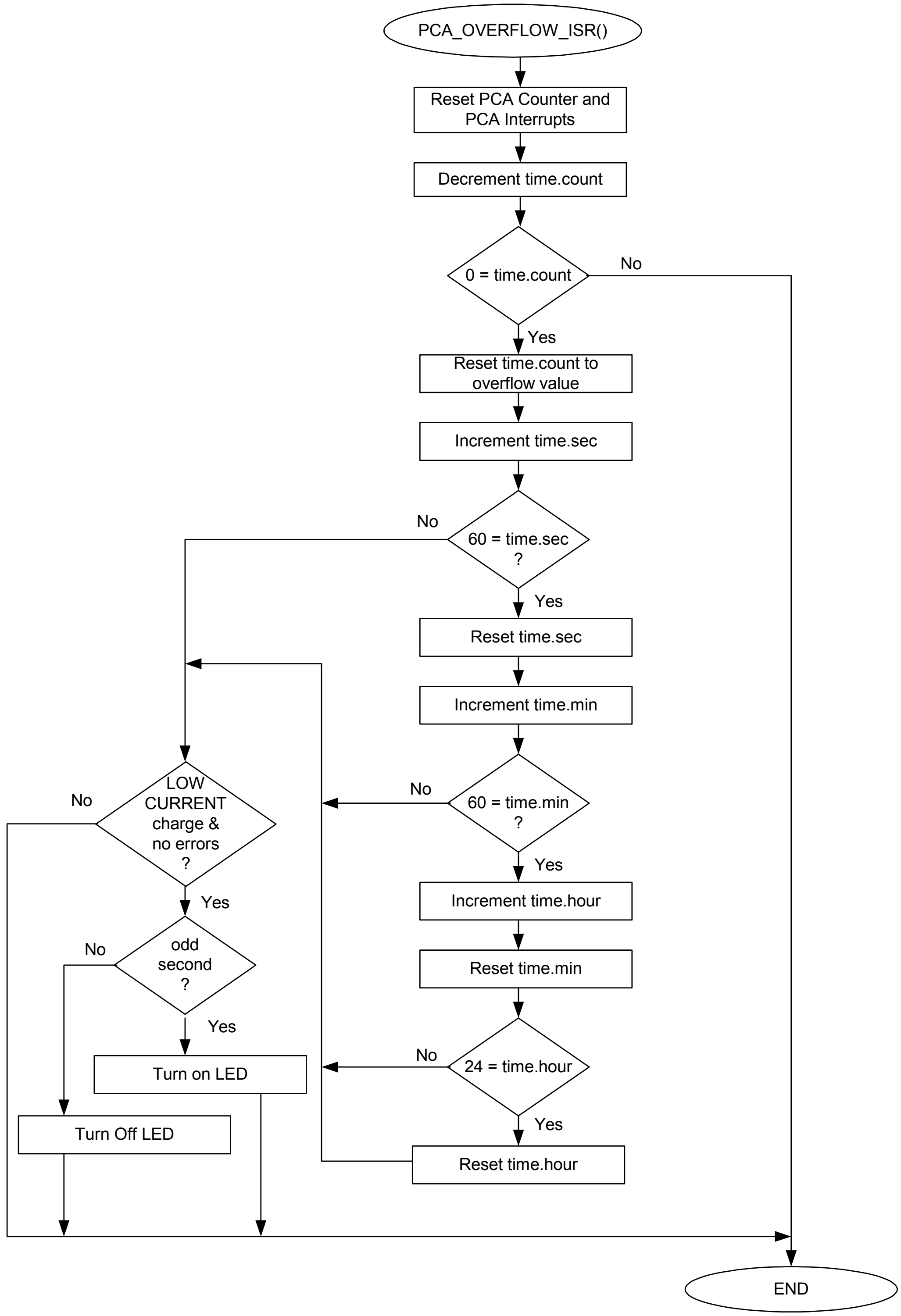


Figure 21. PCA_OVERFLOW_ISR() Flow Chart.



Appendix E - Firmware (Header File)

```
//-----------------------------------------------------------------------------
//
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// Filename:      F300_HighSpeed_BC.h
// Target Device: 8051F300
// Created:       1 MAR 2003
// Created By:    DKC
// Tool chain: KEIL Eval C51
//
// This header file is used to define all preprocessor directives, prototypes,
// and global variable for F300_HighSpeed_BC.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);

//-----------------------------------------------------------------------------
// 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
int Current = 0;                          // Most recent Current Measurement
int Voltage = 0;                          //   used to account for voltage drop
                                          //   across sense resistor

//-----------------------------------------------------------------------------
// 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 LED0      = P0 ^ 2;                  // bit 2 : LED0, Pin P0.2
sbit CMPOUT    = P0 ^ 3;                  // bit 3 : Comparator Output
sbit CMPIN1    = P0 ^ 4;                  // bit 4 : Comparator + Input
sbit CMPIN2    = P0 ^ 5;                  // bit 5 : Comparator - Input
sbit CEX0      = P0 ^ 6;                  // bit 6 : Frequency Output Mode.
sbit CEX1      = P0 ^ 7;                  // bit 7 : 8-bit PWM
                                          // AMUX Selections; Analog Inputs
#define VBAT   0xF0;                      // bit 0 : Voltage Ch.; Analog In
#define IBAT   0xF1;                      // bit 1 : Current Ch.; Analog In
#define TBAT   0xF8;                      // bit 2 : Temp.   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 INT_CURRENT_GAIN   1            // PGA gain setting
#define EXT_CURRENT_GAIN   10           // External gain setting
#define VREF               3300         // 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             33           // 1st cal point for 2 point cal.
#define I2_CAL             2800         // 2nd cal point for 2 point cal.
#define RSENSE             24           // RSENSE is default to 240mohm
#define RESB               20           // 20k Ohms,Voltage Divide Resistor
#define RESAB              30           // 30k Ohms,Sum of 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_Battery 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           250          // mAh, Battery Capacity (LiIon)
#define LiIon_CELL_VOLT    4200         // mV, Nominal Charge Voltage
#define I_BULK             (unsigned int)(CAPACITY)
#define I_LOWCURRENT       (unsigned int)(135)
#define VOLT_BULK       (unsigned int)(CELLS*LiIon_CELL_VOLT)

#define VOLT_LOWCURRENT    (unsigned int)(CELLS*LiIon_CELL_VOLT)

#define VOLT_TOLERANCE     (unsigned int)(CELLS*LiIon_CELL_VOLT/100)// 1 Percent Acc
#define CURRENT_TOLERENCE  (unsigned int)(CAPACITY/10)        // 10 Percent Acc
#define IMIN               100          // Minium Battery Charging is 100 mA
#define IMAX               1350         // Maximum Allowed Current to Protect Hardware

//-----------------------------------------------------------------------------
// Battery Characteristics: Charge TERMINATION Limits
//-----------------------------------------------------------------------------
#define  MIN_TEMP_ABS        26300      // Abs. min. TEMPERATURE = -10 C, 263K
#define  MAX_TEMP_ABS        35300      // Abs. max. TEMPERATURE = 70C, 323K:
#define  MIN_VOLT_BULK      (unsigned int)(CELLS*LiIon_CELL_VOLT*2/3) // Minimum BULK Voltage
#define  MAX_VOLT_ABS        (unsigned int)(CELLS * LiIon_CELL_VOLT)
#define  MIN_I_BULK          (unsigned int)(125)
#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
```

Appendix F - Firmware (Source File)

```
//-----------------------------------------------------------------------------
//
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// Filename:      F300_HighSpeed_BC.c
// Target Device: 8051F300
// Created:       1 March 2003
// 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.
// The High Frequency Output Mode is used to generate the switch rate.
// The default rate is 510 kHz.
//
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f300.h>
#include "F300_HighSpeed_BC.h"          // Battery Hearder File

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

void Config_F300(void)
{ RSTSRC   = 0x02;                      // Enable VDD Monitor
  XBR0     = 0x37;                      // Skip P0.0,1,2,4,5; they're analog In
  XBR1     = 0x90;                      // Enable P0.6, P0.7, as CEX0 and CEX1
  XBR2     = 0x40;                      // Make CEX0 an 8-Bit PWM
                                        // and CEX1 Frequency Output Mode
                                        // Also, Enable crossbar and weak pull-ups

  CMPIN2   = 1;                         // Make Comparator Output Initially low
  CMPIN1   = 0;                         //  to minimize current spikes on start-up

  P0MDOUT  = 0xC8;                      // Set P0.3,6,7 output to push-pull
  P0MDIN   = 0xC8;                      // Configure P0.0,1,2,4,5 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

//-------------------------------------------------------------------
// Comparator Register Configuration
//-------------------------------------------------------------------

  CPT0MX = 0x22;                        // Comparator 0 MUX Selection Register
                                        // P0.4,5 Input to Comparator
```

```
                                        // P0.3 Output of Comparator
  CPT0MD = 0x00;                        // Comparator 0 Mode Selection Register
  CPT0CN = 0x80;                        // Comparator 0 Control Register, Turn on

//-----------------------------------------------------------------------------
// 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 Frequency Output Mode
  PCA0CPL0 = 0x28;                      // Initialize PCA PWM to small duty cycle
  PCA0CPH0 = 0x28;                      // 0x18 makes output frequency ~510kHz
                                        // 0x28 makes output frequency ~306kHz

  //Module 1
  PCA0CPM1 = 0x42;                      // Configure CCM0 to 8-bit PWM mode
  PCA0CPL1 = 0xE0;                      // Initialize PCA PWM to small duty cycle
  PCA0CPH1 = 0xE0;                      // 0xB9 Ensures a Soft Initial Charge

  //Module 2
  PCA0CPM2 = 0x49;                      // Configure Module 1 as software timer
  PCA0CPL2 = 0xFF;                      // Initialize to 255 so that Interrupt
                                        //     is generated when PCA ends
                                        // 8-bit PWM Cycle
  PCA0CPH2 = 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;
}

//-----------------------------------------------------------------------------
// 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
  long i=0;
```

```
EA = 0;                              // Disable All Interrupts

// Wait until 1st calibration voltage is ready for cal
//while (SW0 == 1);                    // Wait until SW0 pushed
for (i=0;i<100000;i++);              // 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
//for (i=0;i<100000;i++);              // 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
  //for (i=0;i<100000;i++);              // 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
  //for (i=0;i<100000;i++);              // 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)INT_CURRENT_GAIN;// Account for Gain

  // Calculate the OFFSET
```

```
  temp_LONG_2.l =  (signed)(temp_INT_1.i/INT_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,delay_count = 0;
  do{
    temp = Monitor_Battery(CURRENT);    // Measure Current
    if (temp < passed_current)
    {     PCA0CPH1--;
          for(delay_count = 0;delay_count<2500;delay_count++);
   }
    if (temp > passed_current)
    {  PCA0CPH1++;
       for(delay_count = 0;delay_count<2500;delay_count++);
    }

  }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*2)) &&
   (temp <= (VOLT_LOWCURRENT + VOLT_TOLERANCE*2)))
  {
    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,delay_count = 0;
```

```
                                        // set VOLT_BULK (with “soft start”)
  do{
    temp = Monitor_Battery(VOLTAGE);

   if (temp < VOLT_BULK)
    {      PCA0CPH1--;
           for(delay_count = 0;delay_count<2500;delay_count++);
    }
    if (temp > VOLT_BULK)
    {  PCA0CPH1++;
       for(delay_count = 0;delay_count<2500;delay_count++);
    }

  }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 (PCA0CPH1 < 0xF0)
      PCA0CPH1++;

  }while (PCA0CPH1 < 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,delay_count=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 |= 0x01;                     // Set PGA gain = 1
    break;

}

//Compute average of next 10 A/D conversions
for(delay_count = 0;delay_count<2500;delay_count++);// Allow Settling Time
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 = 0x46;                        // 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 *= 100;                      // Account for Math Truncation Error
    result *= RESAB;                    // Account for Divide Resistors
   result /= VOLT_SLOPE.l;             // Convert to Voltage in Millivolts
    result /= RESB;
    result -= ((RSENSE*Current)/100); // Account for Sense Resistor Voltage Drop
 break;
  case CURRENT:
    result = (av - I_NOAMP_OFFSET.l); // Account for System Errors
    result *= 100;                      // Account for Math Truncation Error
    result *= 100;                      // Account for Sense Resistor
   result /= I_NOAMP_SLOPE.l;          // Convert to Milliamps
    result /= RSENSE;                   // Account for Sense Resistor
    result /= EXT_CURRENT_GAIN;         // Account for external Amplifier
   Current = (int) result;
   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?
    Voltage = temp;                    // for Debug

   if ((temp <= (MAX_VOLT_ABS + VOLT_TOLERANCE)) && (temp > MIN_VOLT_BULK))
    {
      PCA0CPM0 = 0x46;                 // Turn on PWM

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

        else if (CONST_V == 1)
        {  Current = Monitor_Battery(CURRENT);    // Measure Current
           if((Current < IMIN)||(Current > IMAX))
           { 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
           if (Current < IMIN)
                I_MIN = 1;             // Indicate Specific Error for Debug
```

```
        else
          I_MAX = 1;              // Indicate Specific Error for Debug
      }
    else if ((Current < IMAX) && (Current > IMIN))
    {  I_MAX = 0;                // Reset Error Flag
       I_MIN = 0;                 // Reset Error Flag
       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
//Turn_PWM_Off();                       // Turn PWM off before Calibration
//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
   LOWCURRENT = 0;
   BULK = 1;                          // Start in LOWCURRENT Charge Mode
    CONST_C = 1;

    while (ERROR == 0)
    {
      if (BULK == 1)
      {
        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
      EA = 0;                          // Disable All Interrupts
      while (1);                       // Enter a eternal loop
                                       // No recovery except "reset-button"
    }
  }
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// 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
{ int time_sec;
  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
      }
    }
   time_sec = TIME.sec;

    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
```

Contact Information

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: productinfo@silabs.com
Internet: www.silabs.com