



## **Application Note**

# **Z8 Encore! XP<sup>®</sup>-Based NiMH Battery Charger**

AN022202-0707



**Warning:** DO NOT USE IN LIFE SUPPORT

### **LIFE SUPPORT POLICY**

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

### **As used herein**

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

### **Document Disclaimer**

©2007 by Zilog, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

Z8, Z8 Encore!, and Z8 Encore! XP are registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.

## Abstract

This Application Note describes a Z8 Encore! XP<sup>®</sup>-based Nickel Metal Hydride (NiMH) battery charger. The battery charger application uses the internal clock of the Z8 Encore! XP MCU as the system clock. An internal reference voltage of 2 V is applied to the ADC peripheral of the Z8 Encore! XP MCU.

The source code file associated with this Application Note, AN0222-SC01.zip, is available on [zilong.com](http://zilong.com).

## Z8 Encore! XP 4K Series Flash Microcontrollers

Zilog's Z8 Encore!<sup>®</sup> products are based on the new eZ8<sup>®</sup> CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8 MCU. Z8 Encore! MCUs combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals.

The Z8 Encore! XP 4K Series of devices support up to 4 KB of Flash program memory and 1 KB register RAM. An on-chip temperature sensor allows temperature measurement over a range of -40 °C to +105 °C. These devices include two enhanced 16-bit timer blocks featuring Pulse Width Modulator (PWM), Capture, and Compare capabilities. An on-chip Internal Precision Oscillator (5 MHz/32 KHz) is used as a trimmable clock source requiring no external components. The Z8 Encore! XP devices include 128 bytes of Non Volatile Data Storage (NVDS) memory where individual bytes are written or read. The full-duplex UART, provides serial communications, Infrared Data Association (IrDA) encoding and decoding capability, and supports multidrop address processing in hardware.

The on-chip peripherals make the Z8 Encore! XP MCUs suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

## Discussion

This section discusses the functionality of the Z8 Encore! XP-based battery charger application in detail. For detailed information of the NiMH battery technology see [Appendix E—Battery Technology](#) on page 38.

## Theory of Operation

At the core of a battery charger is the DC-DC converter (also known as a buck converter) that acts as a regulated power source. The charger hardware is capable of regulating the charger output in a number of modes, such as constant voltage, constant current, or

constant voltage with a current limit. The charger is a control system in itself. The type and capacity of the battery determines the mode of operation of the battery controller—namely, a constant current source or a constant voltage source. The voltage ( $V_{SET}$ ) and current ( $I_{SET}$ ) set points are also determined by the type and capacity of the battery.

The parameters, current and voltage, are controlled using the PWM technique. In the PWM technique, the frequency of the signal is maintained constant, and the width of the pulse or the duty cycle of the signal is varied. This variation is reflected as a change in voltage and/or current at the output. The switching regulator reads the parameters through a feedback circuit, and the battery controller operates based on the control algorithm.

The PWM output is obtained by comparing the actual value of the parameter under control with the corresponding set point. In the constant voltage mode, the converter voltage is compared with the voltage set point. In contrast, in the constant current mode, the voltage developed by the charging current across a sense resistor is compared with the current set point. The feedback loop maintains the converter voltage or the converter current constant depending on the selected mode of operation.

Controllers are differentiated based on the method of regulation of parameters in accordance with the corresponding set points. For a detailed information about battery controllers, see related document provided under the Electronics topic in [References](#) on page 11. In a proportional controller, the actual value and the set value are compared, and the resulting error value is used. The drawback of a proportional controller is the possibility of a steady state error. Adding an integral component to the control algorithm eliminates this error.

The equation for a proportional plus integral (PI) controller is:

$$u(t) = (k1 \times e(t) + k2 \times \int e(t) dt)$$

To be useful for a microcontroller-based (discrete) system, the integral is approximated by a running sum of the error signal. Therefore, an equation in the differential form is expressed as follows:

$$U[k] = \left( C1 \times e[k] + C2 \times \sum_{j=0}^{k-1} e[j] \right) \dots\dots\dots \text{Equation 1}$$

where, C1 and C2 are constants.

Equation 1 is the position algorithm. A better representation of Equation 1 is described in Equation 2, as follows:

$$U[k-1] = \left( C1 \times e[k-1] + C2 \times \sum_{j=0}^{k-2} e[j] \right) \dots\dots\dots \text{Equation 2}$$

Subtracting Equation 2 from Equation 1 and rearranging the terms yields Equation 3, as follows:

$$U[k] - U[k-1] = (K_p \times e[k] + K_i \times e[k-1]) \dots\dots\dots \text{Equation 3}$$

where,  $K_p$  and  $K_i$  are the proportional and integral constants, respectively.

Equation 3 is the velocity algorithm, and is a convenient expression as only the incremental change in the manipulated variable is calculated.

The charging algorithms are designed based on the type of battery and the current state of charge for that battery. The basic charging methods are constant current and constant voltage charging. The NiCd and NiMH batteries are charged using the constant current method.

In the constant current mode, fast charging occurs when the charging current equals the rated battery capacity, C. Fast charging requires constant monitoring of battery parameters and precise termination techniques. It is therefore important to know when to terminate charging. In the NiMH battery charger application, the battery parameters are constantly monitored, and the negative  $\Delta V$  termination technique is used. For detailed information about termination techniques associated with Nickel Cadmium (NiCd) and NiMH batteries, see [Appendix E—Battery Technology](#) on page 38.

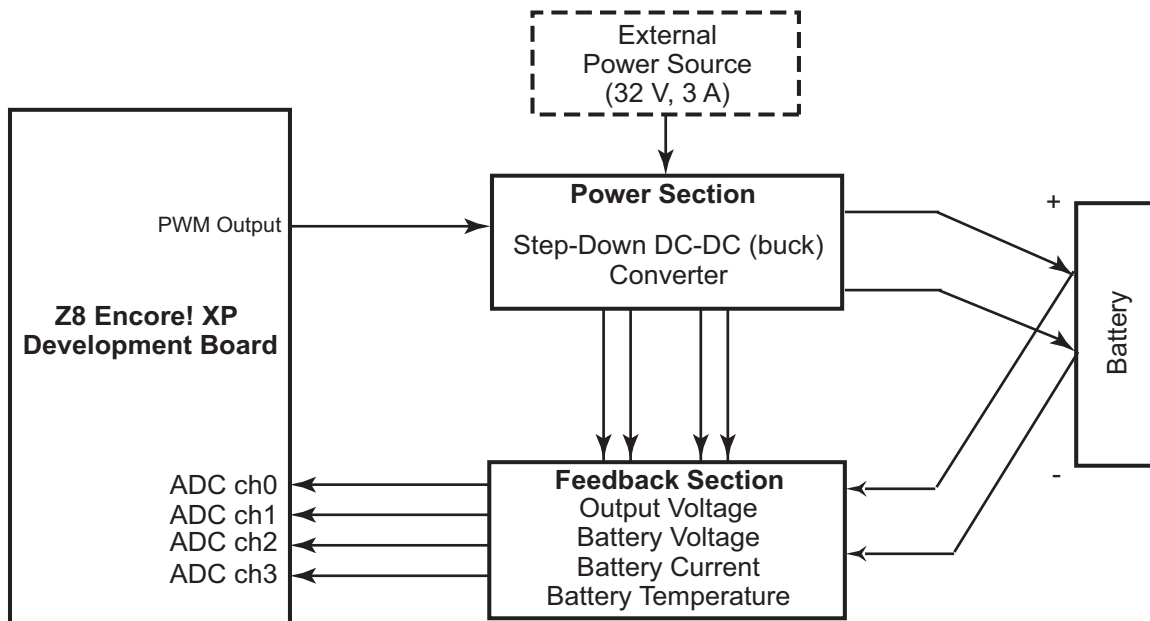
## Developing the Application with the Z8 Encore! XP MCU

This section provides an overview of the functional architecture of the NiMH battery charger implementation using the Z8 Encore! XP MCU.

### Hardware Architecture

[Figure 1](#) illustrates hardware block diagram for the battery charger application. The Z8 Encore! XP-based NiMH battery charger application features the following hardware blocks:

- Z8 Encore! XP Development Board
- External power source (32 V, 3 A)
- Step-down DC–DC (buck) converter
- Feedback section (analog design)
- NiMH battery



**Figure 1. Block Diagram of Battery Charger Hardware**

The battery charger application uses Port B of the Z8 Encore! XP MCU as ADC inputs. Timer 1 is used in PWM mode and the output is tapped at the PC1/Timer 1 output pin. The system clock is derived from the internal precision oscillator of the Z8 Encore! XP MCU. The reference voltage required for the ADC is generated internally by the Z8 Encore! XP MCU, hence the external component requirement and the Bill of Material (BOM) cost is reduced.

The step-down DC–DC (buck) converter provides a voltage or current appropriate to the NiMH battery. The buck converter modulates a higher voltage (from the external source) with a varying pulse width (PWM method) to generate a lower voltage. The pulse width is controlled by the control algorithm based on the values obtained from the feedback section.

The feedback section consists of four differential amplifiers/attenuators. The parameters controlled by the first three amplifiers are the converter voltage ( $V_{OUT}$ ), the battery voltage ( $V_{BATT}$ ), and the battery current ( $I_{BATT}$ ). The battery current and the converter current are same. The fourth differential amplifier is used for temperature measurement in the case of batteries featuring built-in temperature sensors.

For schematic diagrams associated with the battery charger application, see [Appendix B—Schematic Diagrams](#) on page 13.

## Software Implementation

All Z8 Encore! XP peripherals are initialized from their power-on state to the required mode of operation. After initialization, the battery parameters are loaded into the variables. These battery parameters are defined in the `charger.h` header file.

The safety and termination thresholds are calculated based on the battery parameters. The set points for the DC-DC step-down (buck) converter voltage, the current, and the current limit are calculated. After these one-time calculations are complete, the charger software enters into an infinite loop, which is broken only by a successful charge completion or a safety error.

Inside the infinite loop, the ADC reads the actual values for the converter output voltage, the battery voltage, the current, and the temperature (temperature is measured only if the battery features a temperature sensor). The ADC measures the output voltage and the output current of the DC-DC converter as feedback to the controller. The ADC also measures the voltage at the battery terminals as an input to determine the charge termination. Measurement of the output voltage, the output current, and the battery voltage are the basic measurements. The current across the battery terminals is same as the measured converter output current. For batteries featuring built-in temperature sensors, the charger reads the battery temperature in addition to the basic measurements. The temperature measurement is significant from the safety point of view.

After the actual values ( $V_{OUT}$ ,  $V_{BATT}$ , and  $I_{BATT}$ ) are determined, they are checked for safety limit compliance. The safety routine is responsible for the overall safety features associated with the battery charger. The charger ensures safety by comparing the actual converter voltage, the battery voltage, and the battery temperature with the calculated thresholds. Crossing these thresholds switches off the PWM output, which turns off the converter output and terminates charging functions. Such termination protects the batteries in the case of a device failure.

If all the actual values are within limits, the battery is tested for full charge. For NiMH batteries, the battery is considered to be completely charged if the battery voltage stops increasing (zero  $\Delta V$  termination). If the battery is not completely charged, the duty cycle required for maintaining the set points at the converter output is calculated by the control algorithm.

The control algorithm implements proportional plus integral (PI) control to derive the PWM output based on the equations presented in the [Theory of Operation](#) on page 3. The timer ISR is invoked every five milliseconds. The PWM value computed by the control algorithm is loaded into the PWM generators to be transmitted via the output pin. The 16-bit timer PWM mode offers a programmable switching frequency based on the reload value. This flexibility allows designers to trade off between accuracy and frequency of the PWM switching signal. A lower frequency results in a higher reload value and a higher resolution in the pulse width variation. The reverse is also true.

The timer ISR updates the charge termination variables every 10 s.

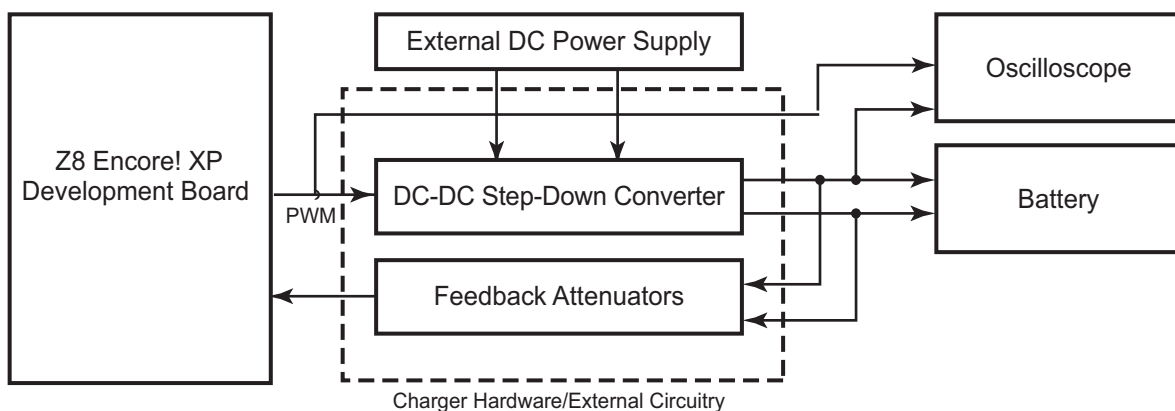
For the flowcharts related to the battery charger application, see [Appendix C—Flowcharts](#) on page 17. [Appendix D—Source Code](#) on page 19 lists the source code for the battery charger application.

## Testing

This section discusses the setup, equipment used, and procedure followed to test the Z8 Encore! XP-based NiMH battery charger application.

### Setup

The test setup for the Z8 Encore! XP-based NiMH battery charger application is as described in [Figure 2](#).



**Figure 2. Battery Charger Test Setup**

The test setup comprises of a Z8 Encore! XP Z8F042A Development Board, an NiMH battery that must be charged, an oscilloscope, an external DC power supply, and a DC-DC step-down (buck) converter. A feedback circuit comprised of differential amplifiers or attenuators forms a part of the test setup.

The external DC power supply provides two different voltages to the charger circuits — the DC-DC step-down converter and the feedback attenuators. The operational amplifier-based feedback attenuator circuits are fed with a 12 V supply. The DC-DC converter works on a 8 V to 12 V DC input for the batteries tested. The control algorithm provides the necessary line regulation to sustain the voltage variation at the input.

### Equipment Used

The equipment used to test the Z8 Encore! XP-based NiMH battery charger are listed in [Table 1](#) on page 9.

**Table 1. Battery Charger Test Equipment**

<b>Equipment to run the System</b>	
Z8 Encore! XP 4K Series Development Kit	(Z8F042A28100KIT)
External power supply	Make: Aplab, Model: LQ 6324
<b>Test Equipment</b>	
Oscilloscope	Make: Tektronix, Model: TDS 724D; 500 MHz / 1 GSps
Multimeter	Make: Motwane, Model: DM3750
<b>Batteries Used for testing</b>	
CP2010H T-014	Make: Panasonic, Type: NiMH, Ratings: 3.6 V, 150 mAh

### Procedure

To test the Z8 Encore! XP-based NiMH battery charger application, perform the following steps:

1. Download the AN0222-SC01.zip file from [www.zilog.com](http://www.zilog.com). Extract its contents to a folder on your PC.
2. Launch ZDSII for Z8 Encore!.
3. Make the hardware connections as shown in [Figure 2](#) and the schematics provided in [Appendix B—Schematic Diagrams](#) on page 13.
4. Connect the battery to be charged across the provided battery terminals (See [Appendix B—Schematic Diagrams](#) on page 13).
5. Apply the required voltages to the appropriate circuits as described in the section [Setup](#) on page 8.
6. Download the battery charger code to the Z8 Encore! XP Flash memory using the ZDSII-IDE.
7. Execute the battery charger code.
8. Observe the PWM waveforms on the oscilloscope.

### Results

The charging of the battery began in the constant current mode with the charging current equal to 1 C. The charging is terminated when the battery does not exhibit an increase in the terminal voltage (zero  $\Delta V$  termination).

## Summary

This Application Note discusses a NiMH battery charger implementation using the Z8 Encore! XP MCU. The battery charger software provides fast charging algorithms. Fast recharge is possible due to the accurate monitoring of the charging rendered by the 10-bit accuracy of the ADC. Monitoring the charging mechanism facilitates the accurate termination of charging. Therefore, overcharging is prevented, resulting in a longer battery life. Additionally, the PWM technique facilitates an accurate DC-DC step-down (buck) converter implementation with excellent line/load regulation.

## References

The documents associated with the Z8 Encore! XP products are listed in [Table 2](#).

**Table 2. List of References**

Topic	Document Name
eZ8 CPU	eZ8 CPU User Manual (UM0128)
Z8 Encore! XP 4K Series Microcontrollers	Z8 Encore! XP 4K Series Product Specification (PS0228) Z8 Encore! XP 4K Series Development Kit User Manual (UM0166)
Electronics	Power Electronics Design Handbook: Low Power Components and Applications; Author: Nihal Kularatna ISBN: 0-7506-7073-8 Publisher: Oxford: Newnes, 1998 High Frequency Switching Power Supplies: Theory and Design; Author: George Chryssis ISBN: 0-07-010949-4 Publisher: McGraw-Hill Book Company Digital Control Systems, Volume 1—Fundamentals, Deterministic Control; Author: Rolf Isermann ISBN: 0-387-50266-1 Publisher: Springer Verlag
ZDSII-IDE	Zilog Developer Studio II-Z8 Encore! User Manual (UM0130)

## Appendix A—Glossary

Definitions for terms and abbreviations used in this Application Note that are not commonly known are listed in Table 3.

**Table 3. Glossary**

<b>Term/Abbreviation</b>	<b>Definition</b>
1C	A charging current rate equal to the A-hr rating of the battery
ADC	Analog-to-Digital Converter
ISR	Interrupt Service Routine
mAh	milli-Ampere-hour: The unit of battery capacity
NiCd	Nickel Cadmium
NiMH	Nickel Metal Hydride
PI	Proportional and Integral
PWM	Pulse-Width Modulation
SLA	Sealed Lead Acid
Li-Ion	Lithium Ion

## Appendix B—Schematic Diagrams

The schematic diagram in Figure 3 illustrates the Z8 Encore! XP MCU pin diagram.

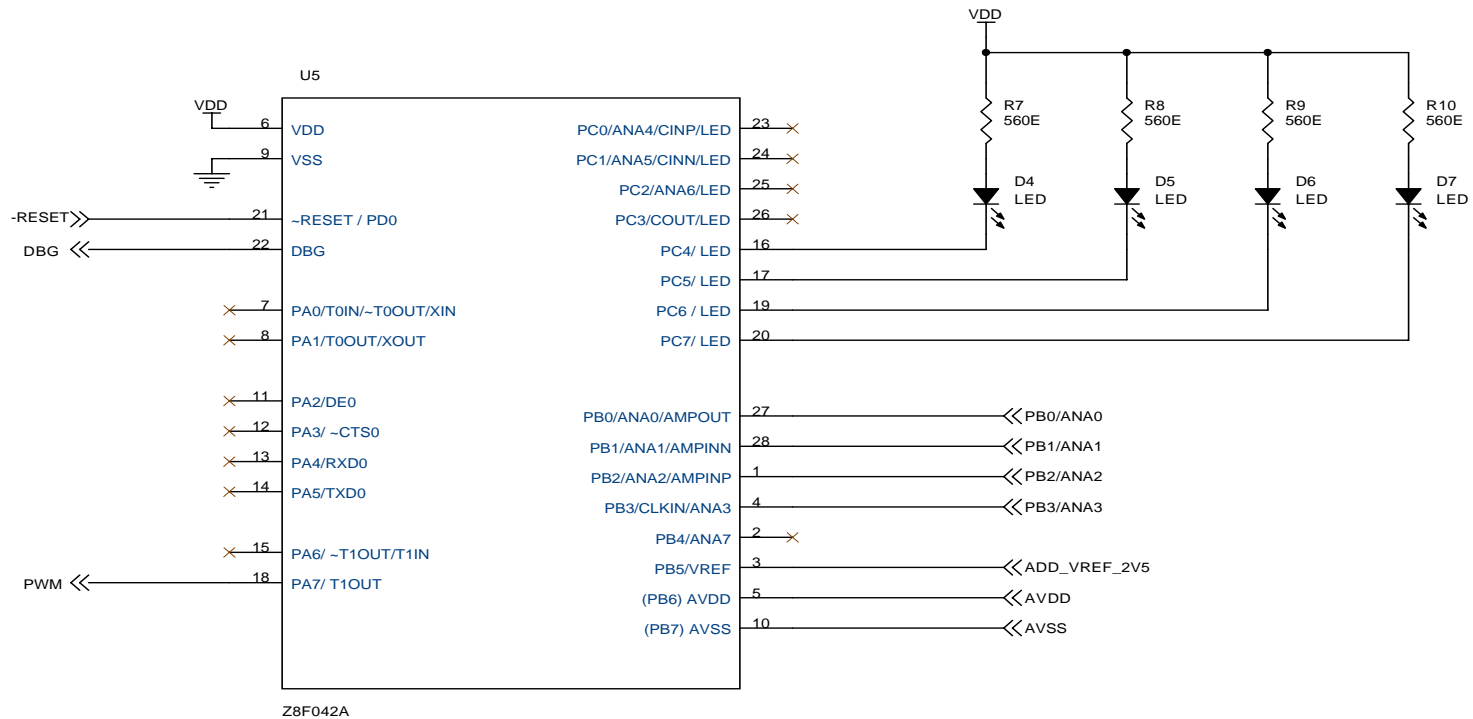
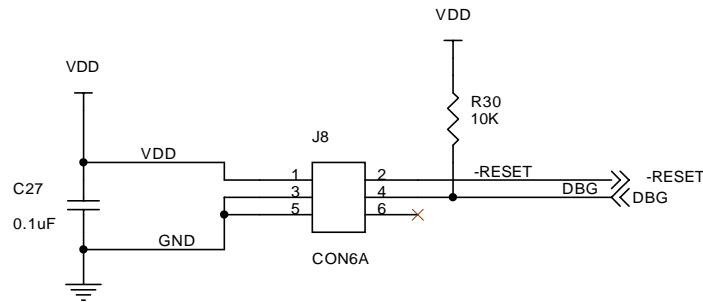
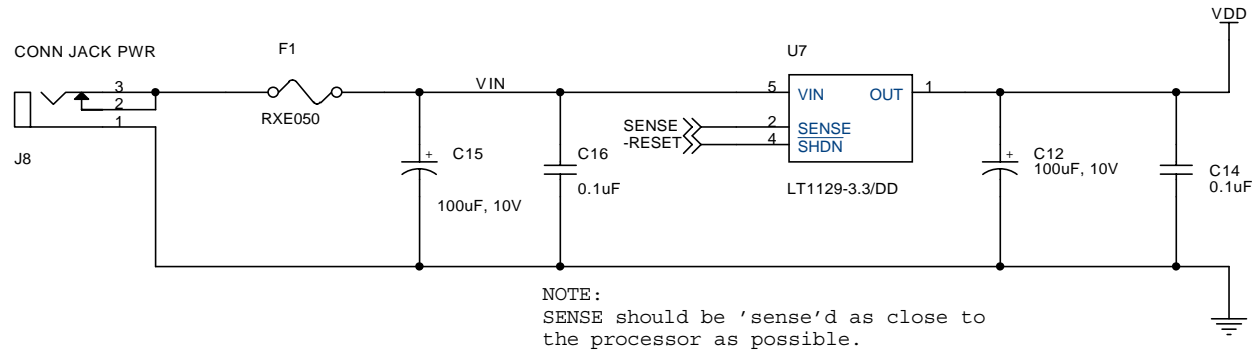


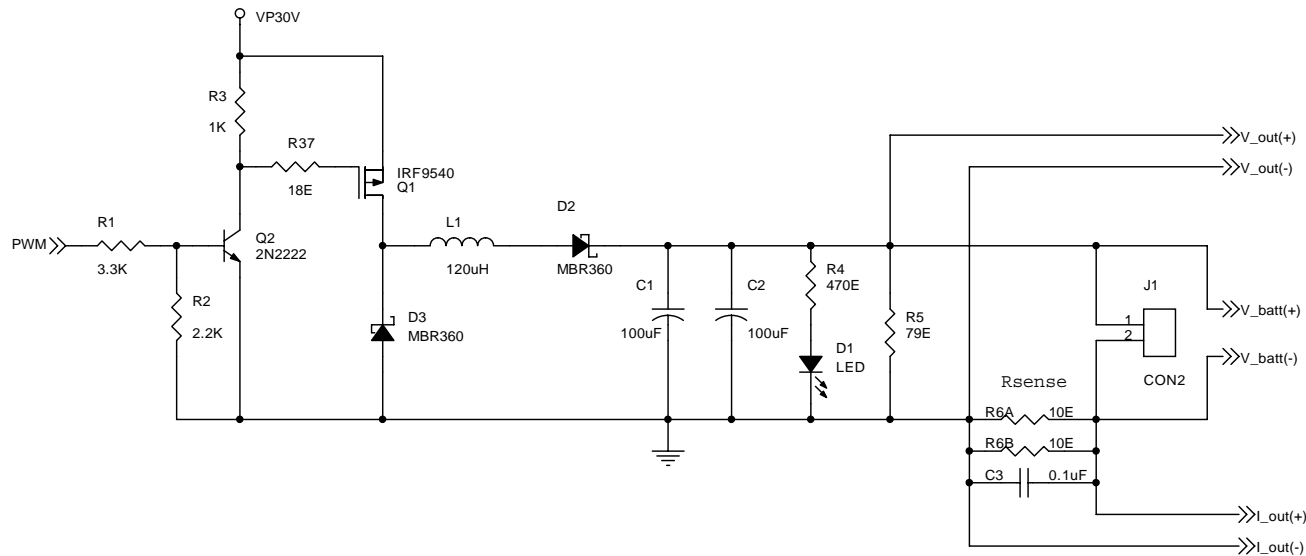
Figure 3. Schematic Illustrating the Z8 Encore! XP MCU Pin Diagram

The schematic diagram in [Figure 4](#) illustrates the external circuitry for the battery charger application.



**Figure 4. Schematic Illustrating the External Circuitry for the Battery Charger Application**

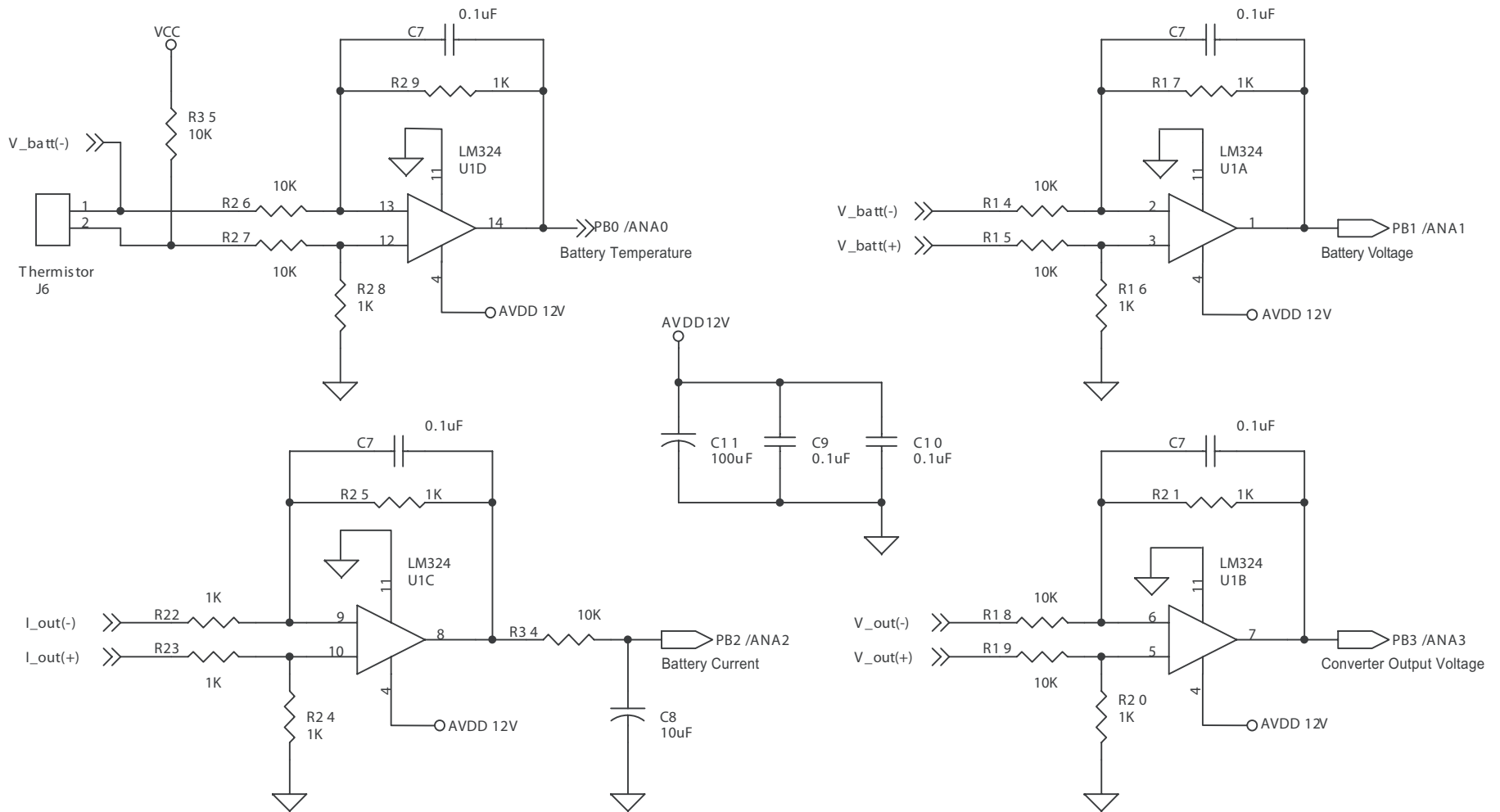
The schematic diagram in [Figure 5](#) illustrates the DC-DC step-down (buck) converter.



NOTE:  
For Testing the VP30V is obtained from  
External Power Source.

**Figure 5. Schematic Illustrating the DC-DC Step-Down (Buck) Converter**

The schematic diagram in [Figure 6](#) illustrates the feedback circuits for the battery charger application.

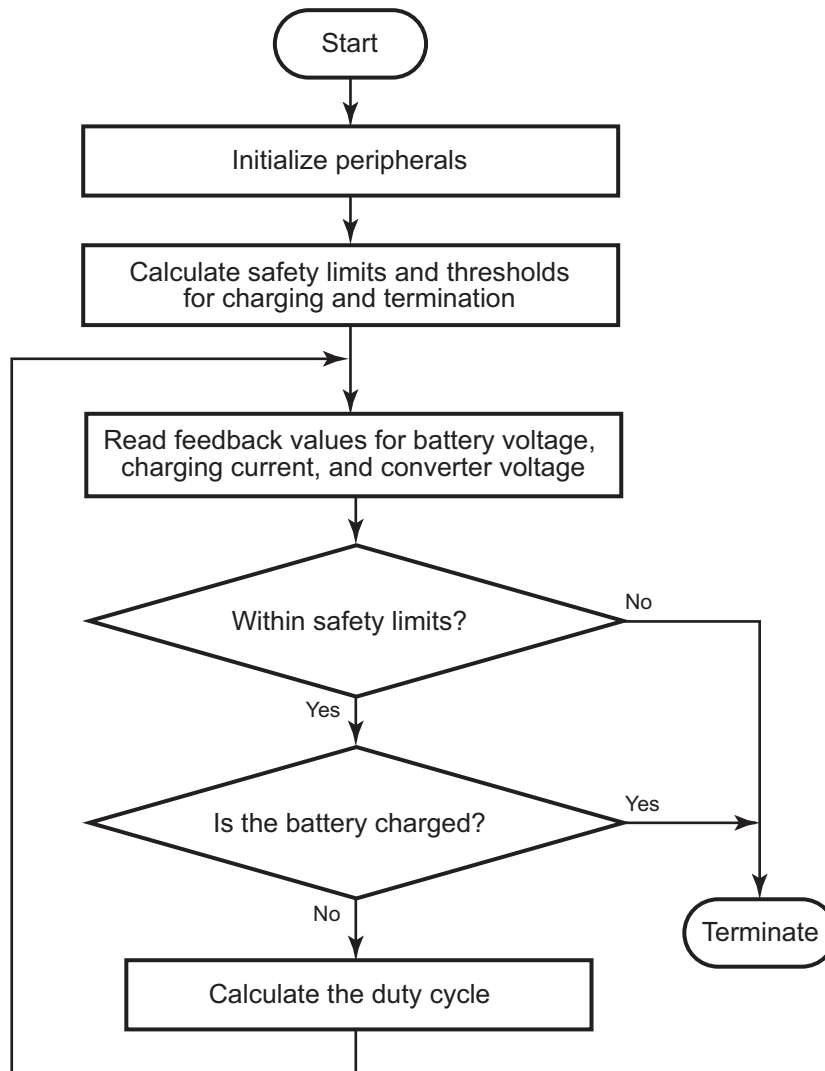


NOTE:  
For Testing the AVDD12V is obtained from External Power Source.

**Figure 6. Schematic Illustrating the Feedback Circuits**

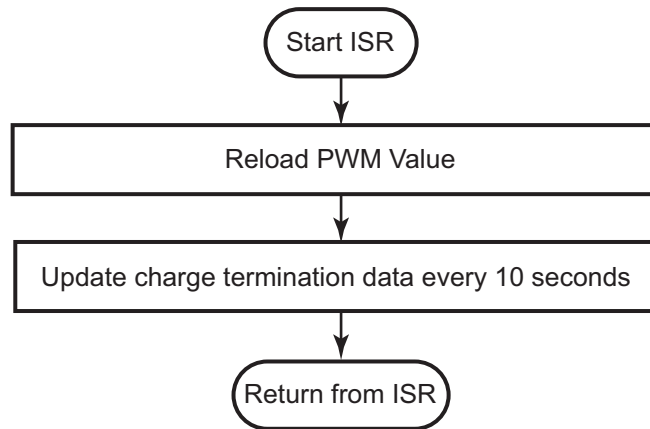
## Appendix C—Flowcharts

This appendix provides flowcharts for the battery charger application described in this document. [Figure 7](#) explains a flowchart of the main routine for the battery charger application. The main routine involves calculation of safety limits, thresholds, duty cycle, reading of feedback values for battery voltage, charging current, and converter voltage.



**Figure 7. The Main Routine**

Figure 8 explains a flowchart of the Interrupt Service Routine (ISR) for reloading the PWM value and updating the charge termination data.



**Figure 8. The ISR Return Routine**



## Appendix D—Source Code

This appendix provides a listing of the source code associated with the application described in this document.

The source code file, AN0222-SC01.zip, is available on [zillog.com](http://zillog.com).

### C Files

The following C files are listed in this section:

- main.c
- initialize.c
- charger.c

```

/*
*****
*****
* File : main.c
* Description: The scope is to demonstrate how the Z8 Encore!
XP® can be * used as a NiMH battery charger. This main program calls
various *
* routines required for the charger to function.
*
* Copyright 2005 Zilog Inc. ALL RIGHTS RESERVED
*
* The source code in this file was written by an
* authorized Zilog employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZILOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*****/
/*

```

Changes in this version

-----

1. Design change: ISR frequency from 5ms to 10ms to give time for ADC

```
2. Changed T0_RELOAD values from 0x5A00 to 0x3600

*/

#include <stdio.h>
#include <ez8.h>
#include "charger.h"

void interrupt isr_timer0(void);
int display_timer = 0, pwm_update_timer = 0,
eoc_update_timer = 0;
int new_pwh = 0, new_pwl = 0;
int eoc_value = 0, last_eoc_value = 0;
int battery_type = 0;
extern int average_value[CHANNELS+1];
extern int new_pwl, new_pwh;

void main()
{
    unsigned int print_data1, print_data2, print_data3;
    int n = 0;
    int i;
    int new_pw;
    unsigned int temp = 0;

    initialize();           // Initialize the system.
    battery_type = NIMH;
    putchar('\n');         // Fresh Line on HyperTerminal.
    ADCCTL0 |= 0x80;       // Enable ADC.
    EI();                  // Enable Interrupts.

    while (n != 1)
    {
        read_feedback_values(); // Find actual values.
        check_for_safety(); // Ensure safe operation.
        n = check_for_termination(battery_type);
        // Find whether further charging is required.
        calculate_pwm_value(battery_type);
        // Find appropriate pwm value based on feedback received.
    }
}

/*
```

```

*****
*****
** Routine      : isr_timer0
** Parameters   : None
** Return       : int
**
** Purpose:
**   This is the ISR for the battery charger.
**   It updates the duty cycle every 10 milliseconds.
**   It updates the fullcharge test values every 10 seconds.
**
*****
*****
*/
#pragma interrupt
void isr_timer0()
{
    eoc_update_timer = eoc_update_timer + 1;
    // Condition for updating termination values
    // printf costly try printhex etc.

    T1CTL  &= 0x7F;          // Disable timer for modifications.
    T1PWH  = new_pwh;        // PWM High.
    T1PWL  = new_pwl;        // PWM Low.
    T1CTL  |= 0x80;          // Enable timer, rest same.

    if (eoc_update_timer == 2000)
        // Store end_of_charging values every 10 seconds.
        {
            last_eoc_value = eoc_value; // Update last value.
            eoc_value      = average_value[1]; // Update present value.
            eoc_update_timer = 0; // Reset every 10 seconds.
        }
}

/
*****
***** End of File < main.c >
*****
*****/

/*

```



```

*****
*****
* File : initialize.c
* Description: The scope is to initialize Z8 Encore! XP®
peripherals so
* that it functions as a NiMH battery charger.
* This program initializes -
* GPIO ports A, B, C
* Timer 0 and timer 1
* Internal Crystal oscillator
* ADC
* required for the charger function.
*
* Copyright 2005 Zilog Inc. ALL RIGHTS RESERVED
*
* The source code in this file was written by an
* authorized Zilog employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZILOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*****/

```

```

#include <stdio.h>

#include <ez8.h>
#include "charger.h"

extern void interrupt isr_timer0(void);
extern int average_value[CHANNELS + 1];
extern int adc_value[CHANNELS + 1];
extern int old[CHANNELS + 1];
extern int max;
extern int Vset, Iset, Ilim, Vbatt_low_th;

int print_data1, print_data2, print_data3;

void initialize(void)
{

```



```

    unsigned int RAM_Location;
    int i;

// Initialize GPIO ports.

PAADDR = ALT_FUN;           // Port A alternate function
-
    PACTL = 0xB0;           // - PA7 = Timer 1 output,
                            PA0, Pa1 //
                            selected thro' OSCCTL.
// PACTL = 0x80; // - PA7 = Timer 1 output, PA0, Pa1
// selected thro' OSCCTL.
// PAADDR = DATA_DIR; // Port A data direction -
// PACTL = 0x10; // - Decided by Alternate Functions
// wherever applicable - rest unused.

// High Drive Enable not initialized for Port A.
// Stop Mode Recovery Source not initialized for Port A.
// Pull Up not initialized for Port A.
// Timer1 output is PUSH-PULL. Same as RESET.
// AFS1, AFS2 not applicable for Port A.
// Data Input Register - No initialization.
// Data Output Register - No initialization.
//-----
-----

// Port B Initialization
PBADDR = ALT_FUN;           // Port B alternate
function -
    PBCTL = 0x2F;           // - PB0:3 ADC inputs,
PB5: VREF.
    PBADDR = AFS1;
    PBCTL = 0x2F; // High Drive Enable not initialized for
// Port B.

// Stop Mode Recovery Source not initialized for Port B.
// Pull Up not initialized for Port B.
// AFS2 not applicable for Port B.
// Data Input Register - No initialization.
// Data Output Register - No initialization.
// Port B data direction is all inputs. Same as RESET.
// Port B output control. Not applicable.

```



```

// -----

// Port C Initialization
PCADDR = ALT_FUN; // Port C alternate function -
PCCTL  = 0x08;    // - Selecting Alternate Function
                // for PC3.

PCADDR = AFS1;
PCCTL  = 0x08;    // Select LED Drive over.
                // Comparator o/p for PC3.

LEDEN  = 0x08;    // Enable LED Drive for PC3.
LEDLVLH = 0x00;
LEDLVLL = 0x08; // 7 mA current sink for PC3.
PCADDR = DATA_DIR; // Port C data direction -
PCCTL  = 0xF7;    // PC3 output, rest input.

// High Drive Enable not initialized for Port C.
// Stop Mode Recovery Source not initialized for Port C.
// Pull Up not initialized for Port C.
// AFS2 not applicable for Port C.
// Data Input Register - No initialization.

PCOUT  = 0xFF;    // Data Output Register - LED OFF.
// Port B data direction is all inputs. Same as RESET.
// Port B output control. Not applicable.
// Port D initialization - As at RESET.
// -----

// Initialize timer 0.
SET_VECTOR(TIMER0, isr_timer0); // Set interrupt service
routine.

// ISR for timer0.
    TOH= 0x00;    // Timer High.
    TOL = 0x00;    // Timer Low.
// TORH = 0x2B;    // Reload Compare High and Low -
//                // from 0x5A00 to 0x3600.
// TORL = 0x33;    // from 5ms to 8mS and from 20MHz
//                // to 5.5296MHz.
TORH = 0x36;    // Reload Compare High and
Low -
//                // from 0x5A00 to 0x3600.
TORL = 0x00;    // from 5ms to 10mS and from 20MHz
//                // to 5.5296MHz.

```



```

TOCTL = 0x11;           // Timer Control#00010001b
                        // | ||_|_____ continuous mode of
                        // operation
                        // |_|_____ divide by 4
                        // Prescale
// Set Timer0 as Level2 (Nominal) priority interrupt
IRQ0ENH |= 0x20;      // IRQ0 Enable High for Timer0.
IRQ0ENL &= 0x00;     // IRQ0 Enable Low.
TOCTL |= 0x80;       // Enable timer0.
// -----

// Initialize timer 1.
// Initialize Timer 1 in continuous PWM mode with prescale =
1.

T1CTL   = 0x03;      // PWM mode, Prescale 1, not enabled.
T1PWMH  = 0x00;      // PWM High.
T1PWML  = 0x00;      // PWM Low.
T1RH    = 0x00;      // RELOAD High.
T1RL    = 0x6E;      // RELOAD Low 0x006E = 50KHz at
                        // 5.5296MHz sysclk.

// T1RH    = 0x01;      // RELOAD High.
// T1RL    = 0x70;      // RELOAD Low 0x0170 = 50KHz at
                        // 18.432MHz sysclk.
                        // Define destination address for
                        // ADC- DMA.
// -----

// Initialize Crystal Oscillator
OSCCTL  = OSC_UNLOCK_SEQ1; // Unlock sequence for OSCCTL
write.

OSCCTL  = OSC_UNLOCK_SEQ2;
OSCCTL  = 0x80;         // Internal 5.5MHz, till 5MHz crystal,
                        // RC is not available.

// Crystal oscillator is enabled.
// IPO is disabled, WDT oscillator enabled.
// Failure detection and recovery of primary oscillator is
enabled.

// Crystal or external RC oscillator as system clock.
// -----

```



```

// ADC initialization
//ADCCTL0 = 0x20;      // External Vref, Single Shot, Analog
                        // channel 0, not enabled.
//ADCCTL1 = 0x00;      // External Vref, Alarms disabled,
                        // unbuffered- single ended inputs.
ADCCTL0 = 0x20;        // Internal Vref, Single Shot, Analog
                        // channel 0, not enabled.
ADCCTL1 = 0x80;        // Internal Vref 2.0volts, Alarms
                        // disabled, unbuffered.
// Find Offset value for ADC
// -----

// Initialize UART for data collection, test only.
UOBRL = 0x06;          // Set Baudrate = 57600;
UOBRL = 0x06;          // Set Baudrate = 57600;
test only

                        // F_clk = 5.5296 MHz.
UOCTL0 = 0xC0;         // Transmit enable, Receive Enable,
                        // No Parity, 1 Stop.
// -----

// IBATT_NICD is battery capacity in mAh
// Iset = IBATT_NICD / 2; // Fast charge at the rate of k-times
                        // full capacity.
// Iset = 75;          // 75 corresponds to 200mA at Vin =
                        // 10.5 Volts.

Iset = 25;
Vset = 232;            // 234 corresponds to 4900mV output.
Vbatt_low_th = 232;   // Switch from CC to CV at 4900mV.
}

/
*****
***** End of File < initialize.c >
*****
*****/

/*
*****
*****
* File      : charger.c
* Description: The scope is to provide all of the core
routines for Z8

```



```

* Encore XP so that it performs as a NiMH battery charger.
* This program implements -
* PI controller
*ADC
* Safety routines
* Termination algorithm
* required for the charger function.
*
* Copyright 2005 Zilog Inc. ALL RIGHTS RESERVED
*
* The source code in this file was written by an
* authorized Zilog employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZILOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*****/

#include <ez8.h>
#include "charger.h"
#include <sio.h>

int Vbatt_max, Vout_max, Vbatt_low_th, flag_test, neg_th,
zero_th, abs_th;
int type, Vset, Iset, Ilim;
int average_value[CHANNELS + 1] = 0;
int adc_value[CHANNELS + 1] = 0;
int old[CHANNELS + 1] = 0;
int error[2] = {0, 0};
int Vpwm = 0;
char data_ready;

extern int eoc_value, last_eoc_value;
extern int new_pwl, new_pwh;
extern int battery_type;

/*
*****
*****

```



```

**
** Routine      :   read_feedback_values
** Parameters   :   None
** Return       :   Nothing
**
** Purpose:
** In the read-feedback_values routine:
** Four Analog channels are converted to Digital data.
** The ADC data is stored arbitrarily from RAM location
0xDE0.
** The RAM location must exclude 0xF00-0xFFF address space.
** The ADC data is passed through a digital low pass filter.
** The final data is stored in the array average_value.
**
*****
*****
*/

void read_feedback_values()
{
long i;
static int channel;

// Check if the data is ready.
// read data at appropriate location.
// if data for last channel is read.
// then set fresh_data flag.
// start conversion for next channel.

if (!(ADCCTL0 & 0x80)) // Check if the ADC data is available
{
// ADC data available.
adc_value[channel] = new_getvalue() // Read ADC data -
adjusted.
average_value[channel] = ((6 * adc_value[channel]) +
(4 * old[channel]))/10; // Low pass filter.
old[channel] = adc_value[channel]; // Preparing for low
pass.
// filter.

// Implement ADC channel sequence 0 - 1 - 2 - 3 - 0.

if (channel == CHANNELS)
{
// This is the last channel.

```



```

data_ready = 1;           // All channels are having a new
                           // value.
channel = 0;             // Start from the beginning channel.
}
else
{
channel++;               // Next Channel.
}
// Proper channel is selected
ADCCTL0 &= ~(0x0F);      // Clear the lower nibble for loading
                           // the new channel.
ADCCTL0 |= (char)channel; // Load the channel in the ADC
                           // control register.
ADCCTL0 |= 0x80;        // Start Conversion for the new
                           // channel.
}
else
{
// Conversion in progress, skip
}
}

/*
*****
*****
**
** Routine      :      check_for_safety
** Parameters   :      None
** Return       :      Nothing
**
** Purpose:
** This routine checks whether the battery voltage and
converter
** output voltage are within the safety limits.
**
*****
*****
*/
void check_for_safety()
{
    if (average_value[1] > Vbatt_max) // Compare actual
        battery voltage           //
        with max limit for Vbatt.
    {

```



```

        T1CTL &= 0x7F;        // Actual battery voltage higher
                               // than threshold - disable PWM
                               // output.
        DI();                // Disable further action.
    }

    if (average_value[3] > Vout_max) // Compare converter
output voltage                // with max limit of Vout.
    {
        T1CTL &= 0x7F;        // Actual output voltage higher
                               // than threshold - disable PWM
                               // output.
        DI();                // Disable further action.
    }
    // Code for thermal safety can be added here.
}

/*
*****
*****
**
** Routine      :      check_for_termination
** Parameters   :      None
** Return      :      int
**
** Purpose:
** This routine contains code for terminating different
** types of batteries. The algorithms available are -
** Negative delta V algorithm for NiCd batteries,
** Zero delta V algorithm for NiMH batteries.
**
*****
*****
*/
int check_for_termination(int type)
{
    static unsigned char count = 0;        // For NiMH zero
delta V                                // termination.
        static int max = 0;                // For NiCd neg
delta V                                // threshold.

```



```

int diff = 0; // For NiCd neg
delta V // threshold.
int difference;
int status = 0; // Initialize
default for // uncharged batteries.

if (average_value[1] > flag_test) // Check if Battery
    voltage is
    // near full charge.
    { // This avoids false early
    // termination.
    if ((eoc_value - last_eoc_value) < zero_th) // zero_th is
        2%
        // of battery rating.
    {
    count = count + 1; // Mark the occurrences.
    if (count > 2) // Avoiding spurious equalities.
    {
    T1CTL &= 0x7F; // terminate - disable PWM
    // output.
    DI(); // Disable further action.
    status = CHARGING_COMPLETE; // Termination flag set,
    // used in main()
    }
    }
    else
    {
    count = 0; // If not three consecutive times,
    // start again.
    }
    }
    return (status); // If charged helps in quitting the
    // main.c loop.
}

/*
*****
*****
**
** Routine : calculate_pwm_value
** Parameters : None
** Return : Nothing

```

```

**
** Purpose: The calculate_pwm_value routine monitors the
battery
calculates the
**status and decides the charging type. It further
** pwm values required for next cycle in control loop.
**
*****
*/void calculate_pwm_value(int type)
{
    int new_pw;

    error[0] = Iset - average_value[2]; // Find present
error.

    // PI controller algorithm.
    Vpwm = Vpwm + ((8 * error[0]) / 10) - ((2 * error[1]) / 3);
    // Stable, best till now.
    error[1] = error[0]; // Store present error as last
    // error.

    // -----
    // Scale Vout to Pulse width.
    // Reload corresponds to Vin - 100% duty cycle.

    new_pw = (Vpwm * RELOAD) / VIN; // So, calculate new_pw to
    // correspond to Vout.

    // -----
    DI(); // Encapsulate so that half the
    // value is not loaded in ISR.
    new_pwl = new_pw & 0x00FF; // Convert the new_pw in PWM
    // register format - L.
    new_pwh = new_pw >> 8; // Convert the new_pw in PWM
    // register format - H.
    EI();
}

/*
*****
*****
**
** Routine : new_getvalue
** Parameters: None

```



```

** Return      :      int
**
** Purpose   : This function gets the value from the ADC HI-
Lo registers ** and concatenates it into a single integer value. This
value is
** returned to the calling function.
*****
*****
*/
int new_getvalue(void)
{
    int ADCvalue, valueH, valueL;
/*
    In single shot conversion the ADC is 11-bits and in 2's
complement
format.
    ADHR holds higher 8-bits and ADLR holds lower 3-bits in
bits 7-5.
*/

valueH = ADHR;
valueH &= 0x00FF; // valueH is an int - Is this required?
valueH = valueH << 3; // Make space for the three least
                    // significant bit.
valueH &= ~(0x07); // Mask all undefined bits.

valueL = ADLR; // Read LSbyte from ADC data Low register
valueL = valueL >> 5; // Position least significant bits
                    // at D0- // D2 positions.
    valueL &= 0x07; // Mask all undefined bits.
    ADCvalue = valueH | valueL; // Form the 10-bit ADC
value.
                    // This is raw ADC data
//ADCvalue = (ADCvalue - ADC_OFFSET) / 2;

return (ADCvalue);
}

/*
*****
*****
***** End of File < charger.c
>*****

```



```
*****  
*****  
* /
```

## Header Files

The following header file is listed in this section:

- charger.h

```

/*
*****
*****
* File : charger.h
* Description : The scope of this header file is to declare
all of the
* functions and constants required for the NiMH battery
charger.
* This header file declares all of the functions and
constants required * for the NiMH battery charger.
*
* Copyright 2005 Zilog Inc. ALL RIGHTS RESERVED
*
* The source code in this file was written by an
* authorized Zilog employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZILOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
******/

/*****
*Function prototypes for main.c
******/
void initialize(void);
void select_parameters(int);
void do_common_calculations(void);
void do_specific_calculations(int);
int read_battery_type(void);
int check_for_termination(int);
void calculate_pwm_value(int);

/*****
* Miscellaneous Definitions
******/

```



```

#define CHANNELS 3 // Four ADC channels 0-3.
#define VIN 360 // Conversion factor for PWM.
// #define VIN 1 // Conversion factor for

PWM

#define R_SENSE 5 // Current sensing resistor.
#define DATA_DIR0x01 // Assign PxCTL for Data direction.
#define ALT_FUN 0x02 // Assign PxCTL for Alternate function.
#define OUT_CTL 0x03 // Assign PxCTL for Output control.
#define RELOAD 0x006E // PWM reload value for 50KHz.
#define CC 0 // Constant current control.
#define CV 1 // Constant voltage control.
#define CHARGING_COMPLETE 0x01 // charge termination flag.
#define ADC_OFFSET22
#define ADC_VREF 2000 // Vref = 2volts.
#define ADC_STEPS 1023

#define OSC_UNLOCK_SEQ10xE7
#define OSC_UNLOCK_SEQ20x18
#define AFS1 0x07
#define AFS2 0x08

/*****
* LED codes
*****/
#define ALL_LEDS_OFF 0xFF
#define CHARGING_OVER 0x7F
#define SAFETY_ERROR 0xBF
#define NO_SELECTION 0xDF
#define LI_ION_BATTERY 0xEF
#define NIMH_BATTERY 0xF7
#define NICD_BATTERY 0xFB
#define SLA_BATTERY 0xFD

/*****
* Battery type definitions
*****/
#define NIMH 0x00

/*****
* Battery type-specific parameters
*****/
#define VBATT_NIMH 3600 // Voltage rating for the NiMH
battery. in mV
#define IBATT_NIMH 270 // Capacity rating for the

```

```

// NiMH battery in mAh

/*****
* ADC Routines and definitions
*****/
typedef unsigned char volatile far *IORegInt8;
#define IO_ADDR(addr) *((IORegInt8) addr)

void read_feedback_values(void);
int new_getvalue(void);

/*****
* Controller routines
*****/
void cv_control(void); // constant voltage control
void cc_control(void); // constant current control

/*****
* Safety Routines
*****/
void check_for_safety(void);

/*****
* Utilities
*****/
void print_hex(unsigned int);

/
*****
***** End of File < charger.h >
*****
*****/

```

## Appendix E—Battery Technology

The four mainstream battery chemistries (NiCd, NiMH, SLA, and Li-Ion) feature different charging and discharging characteristics. Long-term battery life and performance are critically dependent upon how batteries are charged. Therefore, it is important to charge batteries with a mechanism specific to their requirements.

It is also important to know when to terminate charging, because overcharging of a battery invariably results in poor performance and can damage the battery in extreme cases. Moreover, different battery types function differently near full charge condition, and therefore require specific charge termination techniques. During charging, all batteries exhibit a marked rise in voltage above the rated battery voltage.

The two major rechargeable battery types—NiCd and NiMH, are briefly discussed below. For more information, see [References](#) on page 11.

### Nickel Metal Hydride

NiMH batteries exhibit high power density compared to NiCd batteries. The voltage/cell ratio for a NiMH battery is 1.2 V. NiMH batteries are charged using the constant current charging method. If the voltage crosses the full-charge point during charging, the voltage drops but this drop in voltage is not as low as that in the case of NiCd batteries. Therefore, the negative  $\Delta V$  charge termination technique is not recommended for NiMH batteries. NiMH batteries exhibit plateau characteristics after a minimal drop in the voltage. This flat region of the battery characteristics represents the full-charge condition. Therefore, the charge termination technique used in NiMH batteries is termed as zero  $\Delta V$ . NiMH batteries do not exhibit memory effect, hence these batteries are used in consumer durables such as cell phones. NiMH batteries are more expensive compared to NiCd batteries as they are lighter in weight and are not prone to memory effect.

### Nickel Cadmium

NiCd batteries are used in camcorders, walkmans, and other portable consumer electronic equipment. The voltage/cell ratio for a NiCd battery is 1.2 V. NiCd batteries are charged using the constant current charging method. If the voltage crosses the full-charge point during charging, the charging current drops to 15 mV/cell. This voltage drop represents the full-charge condition. Charging is terminated when the battery is in full-charge condition. NiCd batteries use the negative  $\Delta V$  charge termination technique. During charging, the battery voltage rises to 1.65 voltage/cell. Therefore, the battery must be discharged periodically to ensure that the battery functions efficiently. This is memory effect.