PRELIMINARY

DALLAS SEMICONDUCTOR

Application Note 110
NiCD/NiMH Intelligent Battery System Reference Design Using the DS2437

This Application Note presents a reference design for the DS2437 Smart Battery Monitor contained within a NiCD or NiMH battery pack. The DS2437 provides several functions that are desirable to carry in a battery pack: a means of tagging a battery pack with a unique serial number, a direct–to–digital temperature sensor which eliminates the need for thermistors in the battery pack, an A/D Converter which measures the battery voltage and current, an integrated current accumulator, which keeps a running total of all current going into and out of the battery, a real–time–clock, and 40 bytes of nonvolatile EEPROM memory for storage of important parameters such as battery capacity, capacity remaining, and indication of battery cycling.

Information is sent to and from the DS2437 over a 1–Wire™ interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS2437. This means that battery packs need only have three output connectors: battery power, ground, and the 1–Wire interface.

Because each DS2437 contains a unique silicon serial number, multiple DS2437s can exist on the same 1–Wire bus. This allows multiple battery packs to be charged or used in the system simultaneously.

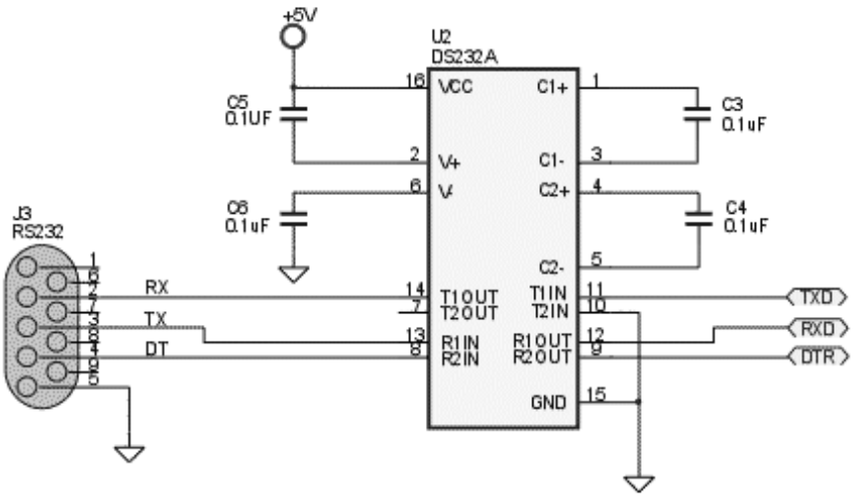
DS2437 Reference Design Hardware

The hardware for the DS2437 Reference Design was set up to allow the user to view the contents of the DS2437 inside the battery pack. The contents include the basic battery information as well as the real–time measurements that are being made during the current charge/discharge cycle of the battery. The reference design is relatively simple in terms of hardware and can be broken into three main sections consisting of a microcontroller, a LCD display and a battery charger.

Microcontroller

A DS5000 Microcontroller provides the controlling function of the reference design as it contains the software that is described in the DS2437 Reference Design Software portion of this application note. The DS5000 is an 8051–compatible microprocessor; it was chosen as the example since many keyboard controllers are also 8051–compatible, and keyboard controllers are often pressed into use as battery management processors in notebook computer systems. Communication with an external computer is accomplished through a RS232 serial port which passes through a DS232A Dual RS232 Transmitter/Receiver (Figure 1). This feature allows the user to update the software programmed into the DS5000.

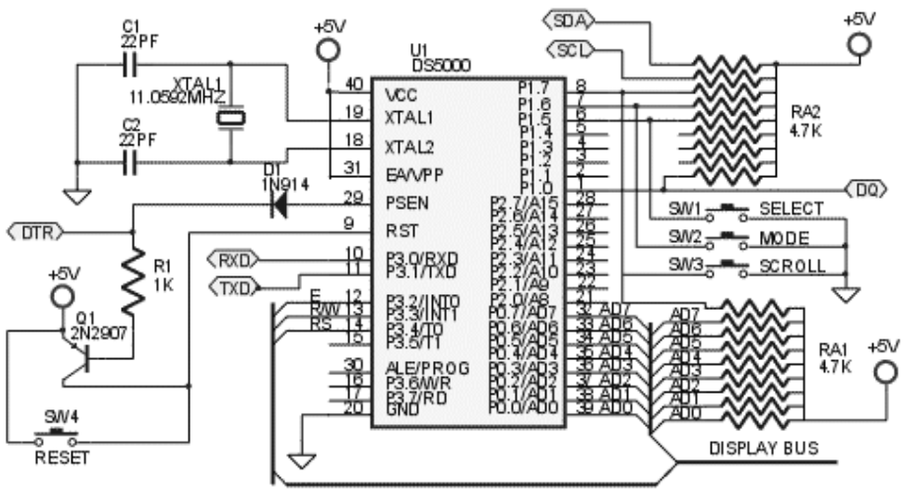
RS232 INTERFACE FOR CHARGER Figure 1



A series of push buttons are connected to the microcontroller to allow the user to choose what information is to be displayed on the LCD display. The RESET button serves the purpose of resetting the system to its initial state at any time. The SELECT button allows the user to select between three main menu screens: Information, Charge/Discharge Mode, and Graphics. The SCROLL button provides a more detailed look at the information contained inside each of the main menu items. The Charge/Discharge Mode selection presents the real–time measurements that are being made by the DS2437. Changing between the charge and discharge mode is done by using the MODE button and can only take place from the first screen within the Charge/Discharge Mode selection. Note that the default condition is for the charger to go into discharge mode upon initialization.

The microcontroller portion of this design is shown in Figure 2.

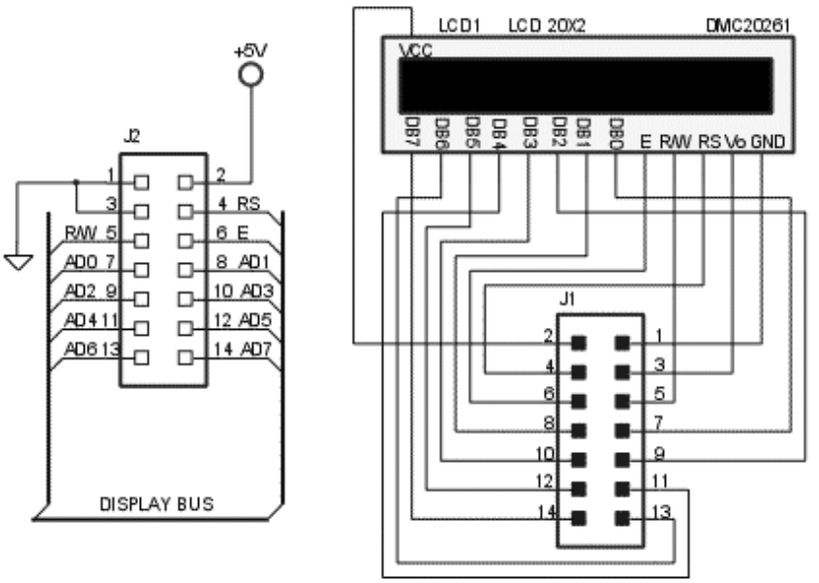
MICROCONTROLLER AND USER PUSHBUTTONS Figure 2



LCD Display

The information contained in the various screens described above is displayed on a DMC20261 20x2 LCD. This display provides the user with the ability to observe the status of the DS2437 on two, twenty character lines with no additional hardware required. The LCD display portion of the design is shown in Figure 3.

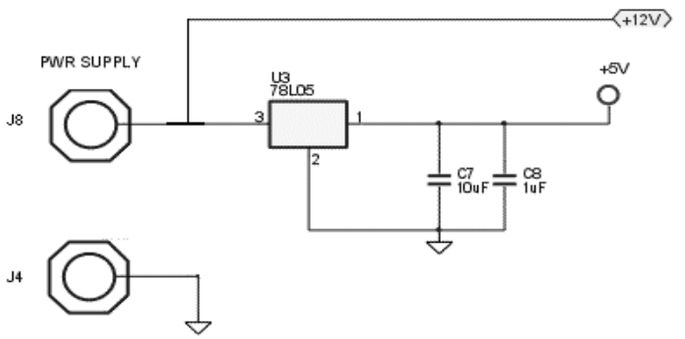
LCD DISPLAY Figure 3



Charger

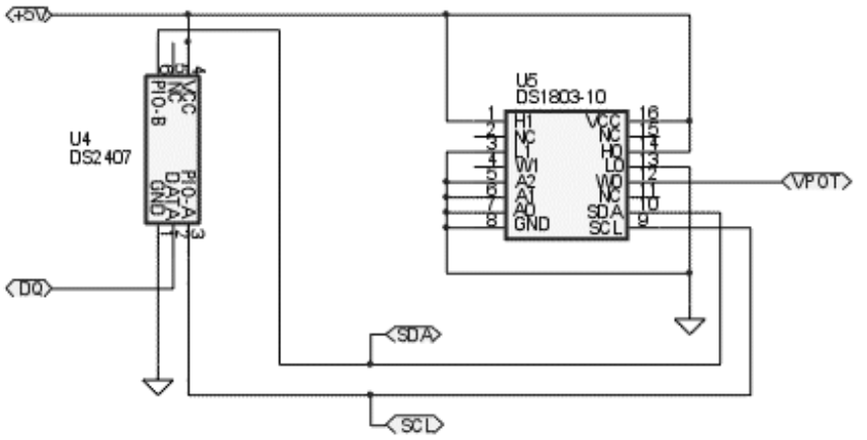
A 12V power supply provides charging current at voltages up to 12V. This supply also connects into a 78L05 Voltage Regulator in order to achieve the +5 volt levels required throughout the reference design. The power supply section of this design is shown in Figure 4.

POWER SUPPLY INPUT AND 5V REGULATOR Figure 4

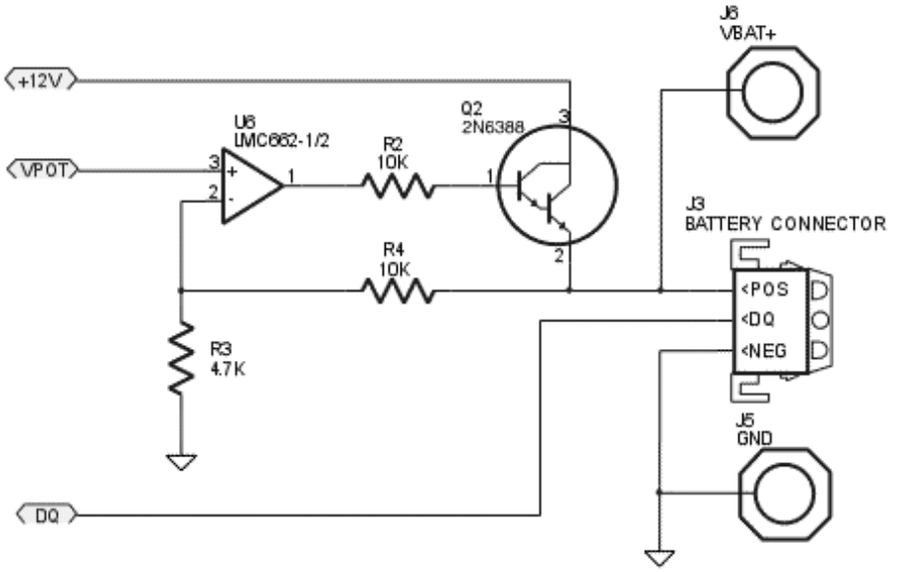


The charger (Figures 5 and 6) is a linear charger which controls current by the use of a DS1803 Digital Potentiometer. This device uses a two–wire interface, so a DS2407 is needed to allow the I/O to be done over the same one–wire which is used by the DS2437. The voltage, VPOT, which is set by the digital potentiometer, will vary according to the manipulations of the software, and then be amplified in order to provide for a sufficient current, up to several amps, to be input into the battery pack for charging purposes.

1–WIRE CONTROL OF 2–WIRE DIGITAL POTENTIOMETER Figure 5



POWER AMPLIFIER FOR CHARGER OUTPUT Figure 6



The charging current is obtained in the following fashion. First, the software measures the battery pack voltage and initializes the charger's output voltage to the battery pack voltage so that there will be no current flow at the beginning of the charge cycle. Note that the charger can only see 8–bit values, so the voltage reading is truncated to 8 bits.

The voltage output by the charger is given by:

Vout = VPOT * (1 + 100/47)

VPOT is calculated from :

VPOT = (tap/255) *5

where the tap is the digital potentiometer setting, 255 is the total number of taps available, and the voltage across the pot is 5 volts.

Thus, solving for the tap since Vout is known, provides:

tap = Vout * 255/((1 + 100/47)*5)

The tap or wiper setting on the DS1803 Digital Potentiometer is then incremented step by step and the current is measured at each step along the way. Once the current reaches the desired current level, the tap is no longer incremented and the charging will continue at a fairly constant level. As the battery voltage rises, the current will drop; the software will respond to this drop in current by further incrementing the tap until the desired current level is restored. This monitoring and adjustment process continues until a terminating condition is encountered.

A charging cycle will terminate if one of several charge termination schemes are met:

Negative Delta V

In this scheme, the battery pack voltage is constantly monitored. When the pack reaches a peak value, and then drops below the peak value by 10 mV per cell on a subsequent number of readings, the charging cycle is terminated.

Zero Delta V

This scheme is very similar to Negative Delta V except that instead of looking for the voltage to drop, the voltage is monitored to find out where it stops rising and "flattens out". When the voltage "flattens out" the charge cycle terminates.

Delta T/Delta t

This scheme is by far the best one and requires the battery pack temperature to be monitored over time. When the slope of the temperature with respect to time takes a "sharp turn up" (increases by 3.7 degrees C in a one minute period), then the pack has been charged to capacity and charging should terminate.

Absolute T

This scheme is implemented for protection of the parts involved in the battery pack. When the temperature, measured from inside the battery pack, becomes greater than 40 degrees C, charging will terminate.

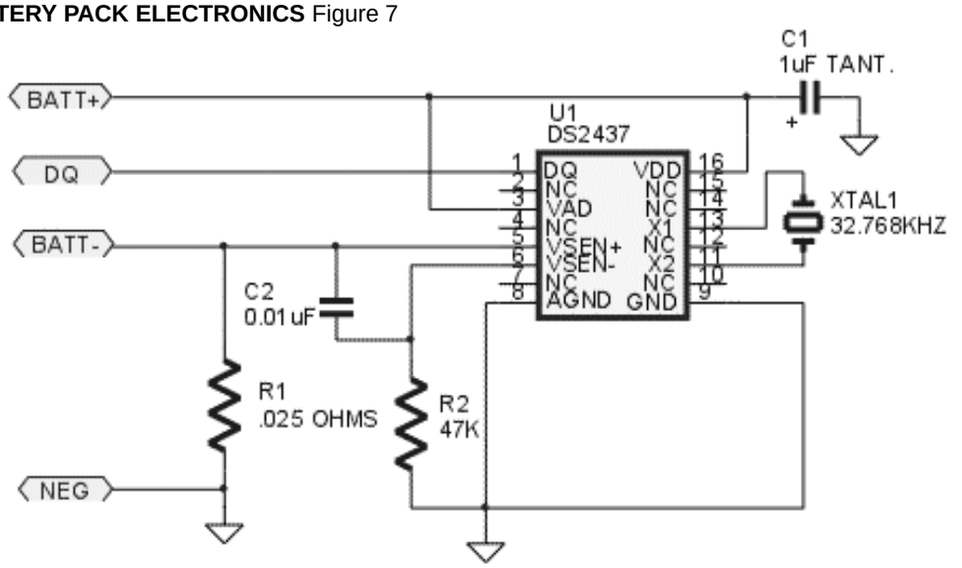
The details of the charging technique and these terminating conditions are located in the DS2437 Reference Design Software portion of this application note.

The battery pack is connected to the charger through a three terminal connector whose terminals are the positive battery voltage (POS), the data (DQ), and the negative battery voltage (NEG).

MEMORY MAP OF THE DS2437

The battery packs used with this reference design contain a DS2437 and support components are needed. These devices are mounted on a PCB inside the battery pack; this battery pack is referred to as an Intelligent Battery. A schematic of the PCB in the Intelligent Battery is shown in Figure 7.

BATTERY PACK ELECTRONICS Figure 7



The memory of the DS2437 is partitioned into eight pages, with Pages 0–2 being composed primarily of volatile SRAM and Pages 3–7 being composed of EEPROM. For a complete look at the memory map of the DS2437, refer to the DS2437 Data Sheet. The information programmed onto Pages 3–7 of the DS2437 memory are outlined below. It is recommended that intelligent battery designs based on the DS2437 use a similar scheme for data storage and data format.

Page 3:

Manufacturer ID

Byte 0 contains the manufacturer ID which is a code which identifies the manufacturer of the equipment with which the battery pack is intended to be used (i.e., an OEM); or, the ID of a battery pack manufacturer which is to be generic across many different OEM applications.

This provides a means of matching the battery pack and the end equipment; if the pack inserted into the equipment does not have a recognized ID code, the equipment may elect to reject that battery pack. Likewise, a charger into which a battery pack is inserted which does not recognize that pack's ID may choose not to charge that pack because the charger may not know how to safely charge the pack. Another option may be to charge a foreign or unrecognized battery pack at a known safe level, probably using a slow charge or trickle charge regime. This design merely reads the code and displays it, and makes no distinction between packs which have a different code.

The manufacturer ID chosen for this design is a hex 44, which is the ASCII character 'D'; this represents that Dallas Semiconductor is the manufacturer.

Chemistry

Byte 1 contains the chemistry of the battery pack which is coded to contain one of the various types of battery chemistries.

The coding scheme used is shown in the table below:

CODE	CHEMISTRY	ABBREVIATION
0000	Primary Cell	———
0001	Lead Acid	PbAc
0010	Lithium Ion	LION
0011	Nickel Cadmium	NiCd
0100	Nickel Metal Hydride	NiMH
0101	Nickel Zinc	NiZn
0110	Rechargeable Alkaline–Manganese	RAM
0111	Zinc Air	ZnAr

The coding scheme can also be located in the PRE_SCR.C section of code under the **info_2()** function.

The remaining codes not used are reserved for future use and definition to accommodate new cell chemistries.

The Primary Cell code (0000) indicates to the host system or battery charger that this pack is made up of primary cells and is therefore not rechargeable. A charger which detects this chemistry should NOT attempt to charge this battery pack at all, and if it is capable, should notify the user that the pack is not a rechargeable type.

This reference design is set up to handle chemistries of either the NiCD or NiMH type.

Number of Cells

Byte 2 contains the number of individual cells (numCells) that comprise the battery pack. This allows the system to describe the battery pack more completely, or to allow charging or power management systems to calculate, from the battery voltage, the average cell voltage for a cell within the battery pack.

This reference design is configured for packs of 6 cells.

Maximum Cell Voltage

Bytes 3 and 4 contain the maximum cell voltage LSB and MSB, respectively (maxCellVLSB and maxCellVMSB). The two bytes are formatted such that voltage is given in millivolts by the following equation.

maximum cell voltage = ((maxCellVMSB * 256) + maxCellVLSB) / 1000

The maximum cell voltage is the maximum voltage that an individual cell within the battery pack is designed or characterized to reach. Typically, voltages above this level will result in damage to the cell or indicate that a cell is damaged.

The maximum cell voltage can be used to determine if a battery is in an overcharge condition, and may be used as a primary or secondary termination limit for charging, depending upon the cell manufacturer's recommendations. It may also be used by a host system to determine the relative "health" of a battery after charging.

This reference design has the maximum cell voltage set at 1.6V.

Minimum Cell Voltage

Bytes 5 and 6 contain the minimum cell voltage LSB and MSB, respectively (minCellVLSB and minCellVMSB). The two bytes of the minimum cell voltage are formatted in the same manner as the bytes of the maximum cell voltage. The minimum cell voltage is the minimum voltage that an individual cell within the battery pack is designed or characterized to reach under normal charge/discharge conditions.

The minimum cell voltage can be used to determine if a battery is deeply discharged. It may be used by a charging system as a limit to prevent rapid charging of the battery pack, indicating instead to perform a slow or trickle charge until the actual battery cell voltage rises above this minimum level. It may be used as a limit below which the cell should not be discharged. It may also be used by a host system to determine the relative "health" of a battery after charging or discharging. Lower than normal cell voltages may indicate a damaged cell.

This reference design uses a minimum cell voltage of 0.9V.

Byte 7 is not used.

Page 4:

Designed Pack Voltage

Bytes 0 and 1 contain the designed pack voltage LSB and MSB, respectively (desPackVLSB and desPackVMSB). The designed pack voltage is the theoretical voltage of the new battery pack and is formatted in the same manner as Bytes 3 and 4 of Page 3 to give a value of millivolts.

This reference design assumes a designed pack voltage of 9.0V.

Minimum Battery Temperature

Byte 2 contains the minimum battery temperature (minBattTemp) which is the lowest temperature, in degrees C, at which the battery can be charged.

This may be used by a power management system to disconnect the battery from a load, or minimize the load on a battery pack, if the battery pack temperature is below this limit. For charging systems, this limit may be used to prevent charging of a battery or indicate that rapid charging should not take place while the battery is below this limit, switching the charger instead to a slow or trickle charge until the battery temperature rises above this limit.

This reference design uses a minimum battery temperature value of 10 degrees C.

Maximum Battery Temperature

Byte 3 contains the maximum battery temperature (maxBattTemp), in degrees C, under which the battery can be charged.

This may be used by a power management system to disconnect the battery from a load, or minimize the load on a battery pack, if the battery pack temperature is above this limit. For charging systems, this limit may be used to prevent charging of a battery or indicate that rapid charging should not take place while the battery is above this limit, switching the charger instead to a slow or trickle charge until the battery temperature falls below this limit.

This reference design uses a maximum battery temperature of 40 degrees C.

Maximum Charge Current

Byte 4 contains the maximum charge current (maxChargeCurr) , in tenths of amps, that the battery pack can sustain under charge. Currents over this limit may damage the pack.

This reference design uses a maximum charge current of 1.9A.

Date of Assembly

Bytes 5 and 6 contain the date of assembly LSB and MSB, respectively (assemDateLSB and assemDateMSB). The date is packed such that:

assembly date = (year – 1980)*512 + month* 32 +day where year is found in bits 9–15, month in bits 5–8 and day in bits 0–4. The following table outlines the format for date packing:

FIELD	BITS USED	FORMAT	ALLOWABLE VALUES
Day	0–4	5–bit binary value	1–31 corresponds to date
Month	5–8	4–bit binary value	1–12 corresponds to month
Year	9–15	7–bit binary value	0–127 corresponds to year biased by 1980

Charger Control Mode

Byte 7 contains the charger control mode (chgCtlMode) which defines the Intelligent Battery pack's capabilities for use with a Battery Charger.

This register is used to report to the Battery Charger how this pack should be charged. It may allow the Battery Charger to override the Intelligent Battery's charge parameter definitions, or it may restrict the charger to only using those parameters stored in the Intelligent Battery. Furthermore, it provides a host system or charger a method to determine the capabilities of the Intelligent Battery pack semiconductor content.

The flags within this register are defined as follows:

MSB							**LSB**
X	DIS	IC	CC	T	V	I	CAP

- X indicates don't care; an unused bit.
- Discharge before charge (DIS) bit is set if the battery pack should have a deep discharge cycle done on it before charging. This bit may be used with the Cycle Count parameter to allow a Battery Charger to determine if it should perform a deep discharge prior to charging the battery pack.
- Internal Charger (IC) bit set indicates that the Intelligent Battery pack contains its own charge controller.
- Charge Controller Enabled (CC) bit is set to enable the battery pack's internal charge controller. When this bit is cleared, the internal charge controller is disabled (default). This bit is active only if the Internal Charger bit is set.
- Temperature (T) bit set indicates that the Intelligent Battery pack can measure and report its own temperature.
- Voltage (V) bit set indicates that the Intelligent Battery pack can measure and report its own voltage.
- Current (I) bit set indicates that the Intelligent Battery pack can measure and report its own current in both charge and discharge modes.
- Capacity (CAP) bit set indicates that the Intelligent Battery pack can measure and report its own remaining capacity.

In this reference design, the packs used have a DS2437 present, but no internal charger. We also do not accomodate discharge–before–charge. The Charger Control Register is then set so that DIS is 0, IC is 0, CC is 0, and T, V, I and CAP are all set to 1, since the DS2437 can measure temperature, voltage, current, and capacity.

Page 5:

Full Charge Capacity

Byte 0 and 1 contain the full charge capacity LSB and MSB, respectively (fullChgCapLSB and fullChgCapMSB), which is the theoretical capacity of a fully charged pack. This is expressed in milliAmp Hours at a C/5 discharge rate and will typically be the 1C rate for a battery. The bytes are formatted using the same equation as Bytes 3 and 4 of Page 3.

The Full Charge Capacity may be used by a host system to determine power management settings for a particular battery pack and desired run time. It may also be used to inform the user about the rated capacity of the Intelligent Battery pack.

The full charge capacity for the batteries used in this reference design are 1.2 Ah for the NiCD pack and 1.9 Ah for the NiMH pack.

Negative Delta V

Byte 2 contains the limit for Negative Delta V per battery cell (deltaVcell) in millivolts. This data would likely only be used with NiCD or NiMH chemistries; for other chemistries to

which this parameter does not apply, this parameter should be 0.

A 10mV/cell Negative Delta V is used in this reference design for the NiCD pack; the NiMH pack uses Zero Delta V termination, so this parameter is set to zero for that pack.

Delta T

Byte 3 contains the limit for the change in temperature (deltaTemp), or the allowable difference between the battery temperature and the ambient temperature, in degrees C. This parameter is used to allow a Battery Charger to determine if a battery is going into an overcharge state by noting a certain temperature rise above ambient.

This reference design uses a value of 15 degrees C. This parameter is not used in this design, however, since no ambient temperature sensor is available.

Delta T/time

Byte 4 contains the allowable limit of temperature change over a one minute period (deltaT-time) given in tenths of a degree C.

This parameter is used to allow a Battery Charger to determine if a battery is going into an overcharge state by noting a certain temperature rise in a given period of time.

This parameter is set to 3.7 degrees C/minute in this reference design.

Battery Pack Manufacturer

Byte 5 and 6 contain information on the battery pack manufacturer (packManfByte1 and packManfByte2, respectively) of the cells used in the Intelligent Battery pack. This may be an identifying number or character, or string of characters which will uniquely identify the manufacturer.

This is used to identify, for reliability or traceability purposes, the manufacturer or assembler of the Intelligent Battery pack. A host system or charger may, if it supports the format used, display the manufacturer name as an identifier and advertisement for the cell manufacturer.

This reference design uses two ASCII characters to identify the pack manufacturer. The characters are “DS” to identify Dallas Semiconductor as the pack manufacturer.

Lot Code

Byte 7 contains the lot code (lotCode) for the battery pack as defined by the assembler of the pack, for traceability and reliability purposes.

Page 6:

Date of Purchase

Bytes 0 and 1 contain the date that the battery pack was purchased (purchaseDateLSB and purchaseDateMSB, respectively). The date is packed into the format as described for Bytes 5 and 6 of Page 4.

This parameter can be used to uniquely identify the pack and provide the system with information on the pack for reliability, traceability, and warranty purposes.

The date of purchase for the battery packs used in this design is set to 3/6/97.

Date of First Use

Bytes 2 and 3 contain the date the battery pack was first used (firstUseDateLSB and firstUseDateMSB, respectively). The date is packed into the format as described for Bytes 5 and 6 of Page 4.

This parameter can be used to uniquely identify the pack and provide the system with information on the pack for reliability, traceability, and warranty purposes.

The date of purchase for the battery packs used in this design is set to 3/7/97.

Assembler

Bytes 4, 5, 6 and 7 of Page 6 and Bytes 0,1 and 2 of Page 7 contains information that identifies the assembler of the Intelligent Battery pack. This may be an identifying number or character, or string of characters that is at the discretion of the assembler.

This parameter is used here as 6 ASCII charac-

ters – “DALLAS”, which identifies Dallas Semiconductor as the Assembler.

Page 7:

Bytes 0,1 and 2 are described with Bytes 4,5,6 and 7 of Page 6.

Termination Scheme

Byte 3 contains the coded termination scheme that is selected for this battery pack. The coding scheme can be viewed in the PRE_SCR.C code listing under the **charge_3()** function, and in the table below. The various terminating schemes are explained in the Charger section of the hardware portion of this application note.

CODE	TERMINATION SCHEME
0	Negative Delta V
1	Zero Delta V
2	Delta T
3	Delta T/time
4	Constant Voltage, Current Limited (this is for a future implementation for Li–Ion batteries)

For this reference design, the Negative Delta V termination is used for the NiCD pack, and the Zero Delta V termination is used for the NiMH pack.

Total Charge Accumulator (CCA)

Bytes 4 and 5 contain the charging current accumulator bytes (ccaByte0 and ccaByte1, respectively) that represents the total charging current the battery has encountered in its lifetime. The bytes are formatted to give charge in units of C, such that:

charge = (ccaByte1 * 256 + ccaByte0) * .32

Further information on the format of data in these registers can be found in the DS2437 data sheet.

Total Discharge Accumulator (DCA)

Bytes 6 and 7 contain the discharging current accumulator bytes (dcaByte0 and dcaByte1, respectively) that represents the total discharging current the battery has encountered in its lifetime. The bytes are formatted in the same manner as the charging current accumulator such that:

discharge = (dcaByte1 * 256 + dcaByte0) * .32

Further information on the format of data in these registers can be found in the DS2437 data sheet.

DS2437 Reference Design Software

The software for the DS2437 Reference Design was created to provide programmed and real–time data from the DS2437 in the battery pack and display it to the user in an efficient manner. This was realized through the implementation of a main event loop that checks to see if any changes were made in the system since the past loop and then makes those updates, which are in turn displayed to the LCD display.

Initialization

Upon starting the program for the first time, following a RESET call, or after battery has been removed and inserted or if a new battery has been introduced to the system, the system goes through an initialization routine.The first step is to set up communication with the DS5000 at a 9600 Baud Rate which is accomplished with the setting of SCON, TMOD, TCON, and TH1 to the values prescribed in the code. Then the variables are initialized to their respective values and the function **initialize()** is called. This function clears the display and positions the cursor in a manner to prepare it to receive the next screen command.

A battery check is accomplished by calling two functions, **Find Devices()** and **Identify Devices()**. These two functions take advantage of some of the 1–Wire software to determine what type, if any, of 1–Wire devices are connected to the system and identify the specific part numbers of the detected devices. Once the battery check determines if a battery is connected to the system, then the data currently on the DS2437 can be read with the **read_data()** function and the main event loop begins. Note the **create_data()** function can be used to change the data programmed into the DS2437, but it is omitted at this time along with the ‘#include setup’ statement that contains that code. A separate set of code is available that can write data to the DS2437 without having it be included in the main operating software.

Event Loop

The main event loop begins every sixth cycle by checking to see if indeed the battery has remained in the system since the last cycle. If it has, or if it is not time to check again, then the loop continues. The first action in this event loop is to obtain all the real–time measurements that the DS2437 has taken and to convert those measurements into a readable format to be placed on the LCD display when they are needed. The **read_real_time()** function accomplishes this task.

The system software then checks the status of the buttons that control the screen selection. The buttons are all initialized to a start up level that will begin at the 'Information' menu screen in the discharge mode. If a button is pressed, the new screen will be selected and displayed during the following event loop. The new screen is determined from the **pick_screen()** function which also accesses the **screen()** function; this provides the formatted output for the LCD display.

The next step is a check to see if the user has selected to enter the Charge Mode. This check is done once every 7 seconds. If the Charge Mode has been selected, then the **main_charge()** function is invoked to begin increasing the charging current as described in the Charger section of this application note. Additionally, the system checks once every fifth time through this charging loop if the specified terminating condition has been met and responds accordingly to the answer obtained.

The final step of the loop is to begin a charging cycle if the measured capacity of the battery falls below 10%. This is checked every thirty seconds. Note the timing involved in many of the checks and function calls which allow the loop to be repeated quickly and be able to detect changes in the system in an efficient manner. The different times reflect the priority of the changes involved and how quickly action needs to be taken if such an event occurs.

Battery Check

The **battery_in_system()** function performs the function of determining if a battery pack containing a DS2437 is in the system. To accomplish this, **FindDevices()** is accessed and another condition must be met. The **ow_reset()** function is called to check if there is indeed any kind of 1–Wire device found in the system, either a DS2407 or a DS2437, and then returns a 0 if a part has been found, or a 1 if no part is found. Then **First()** and **Next()** are called to read the ROM code of all parts found in the path. This will continue until all parts have been found and their ROM codes identified.Then **IdentifyDevices()** matches the ROM code of the parts found to the appropriate family of devices. If at least one DS2407 and one DS2437 are found, then no error will be presented and the battery check will return a true statement to the calling function.

Real–time Calculations

The real–time calculations are performed at the beginning of the main event loop on each occurrence of the loop. The real–time function begins by initiating a temperature and voltage reading by writing a byte to the DS2437 in the form of convert_T and convert_V, respectively. Note that prior to writing any information to a DS2437, **access 2437()** must be called to assure that the part is there and prepare the DS2437 to accept the information that is to be written.

Then each page within the DS2437 that contains real–time data is read and the calculations are made to put the data into a usable format. Temperature, voltage, current and C_rate are calculated from page 0. C_rate is a conversion factor that allows the current values obtained to be changed into terms of amps. Page 1 contains the real–time clock and the ICA, which is the capacity of the battery. The variable *now* is the current value of the real–time clock which has been running since the part was first put into operation in the battery. The LSB of the real–time clock is stored in *timer_now* and that is used in the event loop to control the timing of various processes described in the Event Loop section. Page 7 contains information on the charge and discharge activity over the life of theDS2437.

Note that more information on the manner in which these variables are calculated can be obtained from the DS2437 Data Sheet.

Push Buttons

As described in the Microcontroller section of the DS2437 Reference Design Hardware portion of this application note, there are four push buttons located on the demo board. The three of interest here are the SELECT,SCROLL, and MODE push buttons. Each of these contain a value of 0 when they are idle. Upon being pressed, a value of 1 is stored in the respective location.

When one of these buttons is pressed during the main event loop, it receives a value of 0. The *press_select*, *press_scroll*, or *press_mode* variables are changed from 0 to 1 when a 0 is found on the button associated with each one. Then the next time through the loop, whichever variable has the value of 1, if the respective button has been released and has a value of 1 again, then the *select*, *scroll* or *charging* value is updated accordingly. The *select*, *scroll* and *charging* are the variables that control which screen is to be displayed during each cycle.

Screens

During each loop, **pick_screen()** is called. This function performs the task of selecting which screen is to be displayed on the LCD display during that particular loop. The screen is updated once every 11 times through the loop, unless a condition has changed, such as a new screen being selected, at which time it updates the screen immediately.

Inside of **pick_screen()**, a switch is used to select the proper screen based on the state of the variables *select* and *scroll*. At each case, either the screen is called directly or a pre screen function is called that prepares any information that might need to be determined and passes that to the screen. In either event, eventually a screen is called that sends the appropriately formatted data to the LCD for display. The display will then maintain that screen for 10 cycles or until a change is requested by pushing a button or removing the battery. If no change is detected over the given interval, then the screen is simply refreshed and displays the same data. However, if the screen happens to be one of the real–time screens, then the updated measurements would be displayed at that time.

For printing on the LCD, generally the screen will first be initialized, using **init_display()**. Then the data to be printed will be put into the *display_string* string using the sprintf() function. Then the *display_string* will be printed to the display on either line 0 or line 1 using the **printd()** function.

The screens are set up in a simple menu fashion. The default selection is the 'Information' screen which is one of three main menu titles. The other two are 'Charge/ Discharge Mode' and 'Graphics'. Moving between these screens is accomplished by pressing the SELECT button. The screens will be stepped through, up to one screen per loop, and will cycle back to the 'Information' screen after the 'Graphics' screen. Moving between these main items can be done from any screen to which the user has scrolled. Each main menu item can be further explored using the SCROLL button which allows the user to view each of the screens in that menu. After the last item in the selected menu is viewed, the first screen after the menu title will again be displayed. On the first screen of the 'Charge/Discharge Mode' menu, the user has the choice of being in the charge or discharge mode by pressing the MODE button. The default setting is the discharge mode. All relavent screen information can be viewed whether the reference design is charging or discharging.

Charging

Any time a charge cycle is initiated, the **initial_charge()** function must be called in order to set the tap at the appropriate level so that no current flows initially, as described in the Charger section of this application note. The function **set_charge_level()** takes the value of tap that is calculated and sets the wiper position of the DS1803 Digital Potentiometer to the desired position. Also, the time that each charge cycle is started is time–stamped into the *start_of_charge* variable so that the duration of the charge cycle can be monitored relative to the real–time clock.

The **main_charge()** function will increment the tap value and set the charge level accordingly each time through as long as the measured current of the DS2437 is less than the current specified as the *maxChargeCurr* on the DS2437.

Once a charging cycle is completed, as determined by a terminating condition, the **write_full_charge()** function is invoked which changes the capacity to read 100%. At some times during a charging process a full charge could read 120% or could read 80%, therefore, it is reset to 100% following a completed charge cycle. Note that to write to any one byte on a page of the DS2437, each byte prior to the one that is to be written must also be written. Therefore, in this case, the real–time clock bits must be read and immediately written back so that the 100 can be written to the ICA bit.

Terminating a Charge Cycle

During a charge cycle, there are several factors that could cause the cycle to terminate. The meanings of these termination schemes is outlined in the Charger

section of the DS2437 Reference Design Hardware portion of this application note.

Once every thirty seconds measurements are made to see if a terminating condition is met. However, the checking of the termination condition does not start immediately after charging begins; there must be a period which allows the charging cycle to stabilize. Therefore, any terminating condition that occurs prior to one minute after the start_of_charge marker, will not affect the charging cycle. The only exception to this is if the battery pack temperature exceeds the *maxBattTemp* which is set at 40 degrees C. Any time that condition is exceeded, the charging cycle will terminate.

The Delta T terminating condition is monitored on a sliding one minute window that is updated every thirty seconds. If the Delta T exceeds the specified limit in any one minute period, then the charging cycle will terminate. The Delta V terminating condition is monitored once a peak voltage is encountered. At that time, whenever the current voltage drops below the peak voltage by a value of Delta V, the charging cycle with terminate. A cycle is terminated when the **check_terminate()** function returns a true value to the calling function and then the **write_full_charge()** function is invoked as described in the Charging section above.

```
//charge.c
//   contains routines used in charging a battery

////////////////////////////////////////////////////////////////////////////
// 2407 ACCESS
//
unsigned char access2407(void)
{
     unsigned char i;


     if(ow_reset()) return false;
     write_byte(0x55);                       // match ROM
     for(i=0;i<8;i++)
     {
        write_byte(FoundROM[ds2407ROMNum][i]);       //send ROM code
     }

     return true;
}

////////////////////////////////////////////////////////////////////////////
//
// READS THE CONTENTS OF SRAM IN STATUS MEMORY
//
unsigned char read_status()
{

     if(!access2407()) return false;

     write_byte(0xAA);
     write_byte(0x07);
     write_byte(0x00);

     return read_byte();
}

////////////////////////////////////////////////////////////////////////////
// CHECKS THE WRITE STATUS OF THE 2407
//
unsigned char write_status(unsigned char byte)
{

     if(!access2407()) return false;

     write_byte(0x55);
     write_byte(0x07);
     write_byte(0x00);

     write_byte(byte);               // ..whatever the byte is.
```

```
    //two returned crc bytes are ignored. Take care of them later on.
    read_byte();
    read_byte();

    return true;
}

////////////////////////////////////////////////////////////////////////
// CHECKS IF LINE IS IDLE OR NOT
//
unsigned int Is_It_Idle(unsigned char stat)
{

    stat &= 0x60;
    if ((stat & 0x60)==0x60)
       return true;
    else
       return false;
}

////////////////////////////////////////////////////////////////////////
// IDLE STATE:  Both are high.
// PIO-A = Clock (SCL), PIO-B = Data (SDA)
//
unsigned char Idle_State(unsigned char stat)
{
    stat |=0x60;                    // This will put PIO A & B high.
    return write_status(stat);
}

////////////////////////////////////////////////////////////////////////
// START : Clock stays high, Data goes low.
// PIO-A = Clock (SCL), PIO-B = Data (SDA)
//
unsigned char Start(unsigned char stat)
{

    stat &= 0xBF;           // PIO-B will go low, but
                            // PIO-A will remain as it is.
    if(!write_status(stat))
       return false;

    stat &= 0xDF;
    return write_status(stat);
}

////////////////////////////////////////////////////////////////////////
// STOP  : First clock goes from low to high and then data goes from low to high.
// PIO-A = Clock (SCL), PIO-B = Data (SDA)
```

```
unsigned char Stop(unsigned char stat)
{

     stat  &=0xBF;                          // PIO–B is low.

     if(!write_status(stat))
        return false;

     stat |=0x20;                                    // PIO–A should go high first,

     if(!write_status(stat))                         // PIO–B should stay low.
        return false;

     stat |=0x40;                                    // Now PIO–B can also go high.
     return write_status(stat);
}

////////////////////////////////////////////////////////////////////////
// SEND BIT 1
//
unsigned char send_bit_1(unsigned char stat)
{

     stat |= 0x40;                  // PIO–B high.

     if(!write_status(stat))
        return false;

     stat |= 0x20;                  // clock, PIO–A goes high.

     if(!write_status(stat))
        return false;
     stat &= 0xDF;                  // clock. PIO–A returns to low.
     return write_status(stat);
}

////////////////////////////////////////////////////////////////////////
// SEND BIT 0
//
unsigned char send_bit_0(unsigned char stat)
{
     stat &= 0xBF;                  // PIO–B low.

     if(!write_status(stat))
        return false;

     stat |= 0x20;                  // clock, PIO–A goes high.

     if(!write_status(stat))
        return false;
```

```
    stat &= 0xDF;                           // clock. PIO–A returns to low.
    return write_status(stat);
}


////////////////////////////////////////////////////////////////////////
// SEND BYTE
//
unsigned char send_byte(unsigned char senddata,unsigned char stat)
{
    unsigned char temp;
    int i;

    for (i=0;i<8;i++)
    {
       temp = senddata & 0x80;              //Sends byte of data one bit at a
                                            //time
       if (temp)
       {
              if(!send_bit_1(stat))
                     return false;
       }
       else
       {
              if(!send_bit_0(stat))
                     return false;
       }
       senddata <<= 1;
    }

    temp  = stat;

    temp |= 0x20;                           //Clocking the ACK by making PIO A
                                            //high.

    if(!write_status(temp))
       return false;

    temp &= 0xDF;                           // PIO–A goes low and ACK is clocked.
    return write_status(temp);
}

////////////////////////////////////////////////////////////////////////////////
//
// This function writes a given pot0 value using above functions.
//

void set_charge_level(unsigned char val)
{
    unsigned char stats;
```

```
    stats= read_status();

    if (!(Is_It_Idle(stats)))
       Idle_State(stats);

    Start(stats);
    send_byte(0x50,stats);                  //Control byte.
    send_byte(0xA9,stats);                         //Command byte.
    send_byte(val,stats);                   //Any value
    Stop(stats);

}
////////////////////////////////////////////////////////////////////////////
// WRITE FULL CHARGE
//
void write_full_charge(void)
{

    read_page2437(1);                               //Copies value of real–time
                                                    //clock to scratchpad
       rtcByte0  =    page[0];
       rtcByte1  =    page[1];
       rtcByte2  =    page[2];
       rtcByte3  =    page[3];
    if(access2437())
    {
       write_byte(write_scratchpad);                //Writes real–time clock from
                                                    //scratch pad to page 1
       write_byte(1);
       write_byte(rtcByte0);
       write_byte(rtcByte1);
       write_byte(rtcByte2);
       write_byte(rtcByte3);
       write_byte(0x64);                            //Writes 100% to capacity
       ow_reset();
    }

    if(access2437())
    {
       write_byte(copy_scratchpad);                 //Saves changes made by copy
                                                    //ing scratchpad to page 1
       write_byte(1);
       ow_reset();
    }

    init_display();
    sprintf(display_string,” CHARGING COMPLETE”);
    printd(display_string,0);
    set_charge_level(0);
    select=1;                                       //Resets display to MODE:
                                                    //DISCHARGE screen
```

```
    scroll=12;
}
////////////////////////////////////////////////////////////////////////////
// TRICKLE CHARGE ROUNTINE
//      Charges at a lower current when battery is out of safe charging range
//
void trickle_charge(void)
{
    trickle_current = maxChargeCurr/100;

    set_charge_level(tap);          //Updates the charging level

    if(current>trickle_current)     //Decrements tap if current is greater
    {                               //than desired current
       if(tap>0)
              tap--;
       else
              charging=false;
    }

    else                    //Increments tap if current is less
    {                       //than desired current
       if(tap<256)
              tap++;
       else
              charging=false;
    }
}

////////////////////////////////////////////////////////////////////////////
// INITIAL CHARGING ROUNTINE
//
void initial_charge(void)
{
    start_of_charge = now;                  //Sets a timestamp of when the charge
                                            //was started

    if(chemistry!=3&&chemistry!=4)          //If battery is of other
                                            //chemistry than NiCD
    {                                       //or NiMH, charging will not take
                                            //place
       init_display();
       sprintf(display_string,"UNKNOWN CHEMISTRY...");
       printd(display_string,0);
       sprintf(display_string,"REMOVE BATTERY");
       printd(display_string,1);

       delay_second();
       delay_second();

       charging=false;
```

```
       return;
    }

    tap = voltage*255.0/((1.0+100.0/47.0)*5.0);
                                                //Sets the voltage level to
                                                //the desired tap setting
    set_charge_level(tap);                      //of the pot so that the
                                                //output voltage is equal to
                                                //the measured voltage so
                                                //there is no intial current
                                                //flow

    desired_current = maxChargeCurr/10;     //Sets the desired current to the
                                            //max Charge Current in Amps

}
////////////////////////////////////////////////////////////////////////////
// MAIN CHARGING ROUNTINE
//
void main_charge(void)
{
    if(voltCell<minCellVolt&&trickle==0)            //Cell voltage must be above
    {                                               //safe charging level to
       init_display();                              //charge normally, otherwise
       sprintf(display_string,"CELL VOLTAGE BELOW");//a trickle charge is invoked
       printd(display_string,0);
       sprintf(display_string,"SAFE CHARGING LEVEL");
       printd(display_string,1);

       delay_second();
       delay_second();

       init_display();
       sprintf(display_string,"      STARTING ");
       printd(display_string,0);
       sprintf(display_string,"   TRICKLE CHARGE");
       printd(display_string,1);

       delay_second();
       delay_second();

       trickle=true;

    }

    if(temperature<minBattTemp&&trickle==0)         //Battery temp must be above
    {                                               //safe charging level to
       init_display();                              //charge normally, otherwise
       sprintf(display_string,"BATTERY TEMP BELOW"); //a trickle charge is invoked
       printd(display_string,0);
       sprintf(display_string,"SAFE CHARGING LEVEL");
```

```
      printd(display_string,1);

      delay_second();
      delay_second();

      init_display();
      sprintf(display_string,"      STARTING ");
      printd(display_string,0);
      sprintf(display_string,"   TRICKLE CHARGE");
      printd(display_string,1);

      delay_second();
      delay_second();

        trickle=true;

      }

   if(temperature>maxBattTemp)                    //Will not charge if battery
   {                                              //temp is above maxBattTemp
      init_display();
      sprintf(display_string,"BATTERY TEMP ABOVE");
      printd(display_string,0);
      sprintf(display_string,"SAFE CHARGING LEVEL");
      printd(display_string,1);

      delay_second();
      delay_second();

      charging=false;
      return;
   }

   if(trickle==true)
   {
      if(voltCell<minCellVolt||temperature<minBattTemp)
      {
             trickle_charge();
             return;
      }
      else
             trickle=false;         //Stops trickle charge when conditions are
                                    //safely met
   }

   set_charge_level(tap);                  //Updates the charging level

   if(current>desired_current)      //Decrements tap if current is greater
   {                                //than desired current
      if(tap>0)
             tap--;
```

```
       else
              charging=false;
    }

    else                            //Increments tap if current is less
    {                               //than desired current
       if(tap<256)
              tap++;
       else
              charging=false;
    }
}
```

```
// display.c
//   LCD display control routines for LCDs with HD44780 controller
//
#include ”hardware.h”

// Display control commands
#define cleardisplay   0x01
#define home           0x02

// entry mode – usage is entry | dir_[option] | shift_[option]
#define entrymode      0x04
#define dir_minus      0x00
#define dir_plus       0x02
#define shift_off      0x00
#define shift_on       0x01

// Display elements on/off – usage is displayctl | display_[option]
//                       | cursor_[option] | blink_[option]
#define displayctl     0x08
#define display_off    0x00
#define display_on     0x04
#define cursor_off     0x00
#define cursor_on      0x02
#define blink_off      0x00
#define blink_on       0x01

// Display or Cursor Shift – usage is shift | [cursor/both] | dir_[option]
#define shift          0x10
#define cursor         0x00
#define both           0x08
#define dir_right      0x00
#define dir_left       0x04

//  Data  Interface  –  usage  is  interface  |  int_[option]  |  lines_[option]  |
//dots_[option]
#define interface      0x20
#define int_8          0x10
#define int_4          0x00
#define lines_one      0x00           // one line display
#define lines_two      0x08           // two line display
#define dots_7         0x00           // 5x7 dots
#define dots_10        0x04           // 5x10 dots

// Char Gen Addr – usage is chargen | [address]
#define chargen        0x40

// RAM Addr – usage is ramaddr | [address]
#define ramaddr        0x80

////////////////////////////////////////////////////////////////////////
// BUSY – waits until busy line of display is low
```

```
//
void busy(void)
{
     E = 0;            // set enable signal low
     DDP = 0xFF;       // make DDP an input
     RS = 0;           // drop RS = instruction
     RW = 1;           // read
     E = 1;            // strobe data in

     while (BUSY){};   // loop here until busy signal goes low

     E = 0;            // reset enable line
}
////////////////////////////////////////////////////////////////////////////
// WRITE_DISPLAY
// writes a command or data to the display – mode sets command (0) or data(1)
//
void write_display(char cmd, bit mode)
{
     busy();       // check if display is busy
     RW = 0;       // set to write mode
     RS = mode;    // set mode: command or data
     DDP = cmd;    // write command to DDP
     E = 1;        // set E high
     E = 0;        // then bring E low
}
////////////////////////////////////////////////////////////////////////////
// INIT_DISPLAY – sets the display up for 8 bit mode, sets controller to two
//               lines, 5x7 font, move increment and shift,
//               clears the display and turns cursor off
//
void init_display(void)
{
     write_display(interface|int_8|lines_two|dots_7,0);      // data length = 8
                                                             // bits,
                                                             // 2 lines, 5x7
     write_display(cleardisplay,0);                          // clear display
     write_display(entrymode|dir_plus|shift_off,0);          // auto increment
                                                             // cursor
     write_display(displayctl|display_on|cursor_off|blink_off,0);   // display on,
                                                                    // cursor off
}
////////////////////////////////////////////////////////////////////////////
// GOTOXY – sets the current cursor location to column (x) and row (y).
//

void gotoxy(unsigned char x, unsigned char y)
{
     unsigned char row;
     if(y==1) row = 0x40;
     else row = y;
```

```
    write_display((ramaddr|row|x),0);
}
////////////////////////////////////////////////////////////////////////
//write_chargen – sends command to write to the character generator at address
//
void write_chargen(unsigned char address)
{
    write_display(chargen|address,0);       //sends starting address for chargen
}

////////////////////////////////////////////////////////////////////////
// printd – print to display
// this print statement will accept a string (probably created by the calling
// routine using a sprintf() statement), and print it on the display
// on the line specified in line (0 or 1).
//
void printd(char* disp_string, unsigned char line)
{
    unsigned char i;

    gotoxy(0,line);
    for(i=0;i<strlen(disp_string);i++)
    {
       write_display(disp_string[i],1);
    }
}

////////////////////////////////////////////////////////////////////////
//CHARACTER GENERATION
//
//
//DELTA – This routine generates a delta that can be displayed on the LCD
//
//
void delta(void)
{
    write_chargen(0x08);
    write_display(0x00,1);
    write_display(0x00,1);
    write_display(0x00,1);
    write_display(0x04,1);
    write_display(0x0A,1);
    write_display(0x11,1);
    write_display(0x1F,1);
    write_display(0x00,1);
}

// BAR – this routine generates a bar to be used in the graphics section
//
//
void bar_graph(void)
```

```
{
    write_chargen(0x08);
    write_display(0xFF,1);
    write_display(0xFF,1);
    write_display(0xFF,1);
    write_display(0xFF,1);
    write_display(0xFF,1);
    write_display(0xFF,1);
    write_display(0xFF,1);
    write_display(0x00,1);
}
```

```
/* DS2437 commands*/
#define        read_scratchpad        0xBE
#define write_scratchpad              0x4E
#define copy_scratchpad               0x48
#define recall_memory                 0xB8
#define convert_T                     0x44
#define convert_V                     0xB4

#define degree 0xDF
#define deltas 0x01
#define bar    0x01

// Page0

unsigned char xdata   statusReg;     //byte 0
unsigned char xdata   tempLSB;       //byte 1
signed char xdata     tempMSB;       //byte 2
unsigned char xdata   voltLSB;       //byte 3
unsigned char xdata   voltMSB;       //byte 4
unsigned char xdata   currLSB;       //byte 5
signed char xdata     currMSB;         //byte 6
                      //byte 7 is reserved

// Page1

unsigned char xdata   rtcByte0;      //byte 0
unsigned char xdata   rtcByte1;      //byte 1
unsigned char xdata   rtcByte2;      //byte 2
unsigned char xdata   rtcByte3;      //byte 3
unsigned char xdata   ica;           //byte 4
                      //byte 5 is reserved
                      //byte 6 is reserved
                      //byte 7 is reserved

// Page2

unsigned char xdata   disconnectByte0; //byte 0
unsigned char xdata   disconnectByte1; //byte 1
unsigned char xdata   disconnectByte2; //byte 2
unsigned char xdata   disconnectByte3; //byte 3
unsigned char xdata   endChargeByte0;  //byte 4
unsigned char xdata   endChargeByte1;  //byte 5
unsigned char xdata   endChargeByte2;  //byte 6
unsigned char xdata   endChargeByte3;  //byte 7

// Page3

unsigned char xdata   manufacturerID;  //byte 0
unsigned char xdata   chemistry;       //byte 1
unsigned char xdata   numCells;        //byte 2
unsigned char xdata   maxCellVLSB;     //byte 3
```

```
unsigned char xdata   maxCellVMSB;          //byte 4
unsigned char xdata   minCellVLSB;          //byte 5
unsigned char xdata   minCellVMSB;          //byte 6
                      //byte 7 is unused

// Page4

unsigned char xdata   desPackVLSB;          //byte 0
unsigned char xdata   desPackVMSB;          //byte 1
signed char  xdata    minBattTemp;          //byte 2
signed char  xdata    maxBattTemp;          //byte 3
unsigned char xdata   maxChargeCurr;        //byte 4
unsigned char xdata   assemDateLSB;         //byte 5
unsigned char xdata   assemDateMSB;         //byte 6
unsigned char xdata   chgCtlMode;           //byte 7

// Page5

unsigned char  xdata  fullChgCapLSB;        //byte 0
unsigned char xdata   fullChgCapMSB;        //byte 1
unsigned char xdata   deltaVcell;           //byte 2
unsigned char xdata   deltaTemp;            //byte 3
unsigned char xdata   deltaTtime;           //byte 4
unsigned char xdata   packManfByte1;        //byte 5
unsigned char xdata   packManfByte2;        //byte 6
unsigned char xdata   lotCode;              //byte 7

// Page6

unsigned char xdata   purchaseDateLSB;      //byte 0
unsigned char xdata   purchaseDateMSB;      //byte 1
unsigned char xdata   firstUseDateLSB;      //byte 2
unsigned char xdata   firstUseDateMSB;      //byte 3
unsigned char xdata   assemSiteByte0;       //byte 4
unsigned char xdata   assemSiteByte1;       //byte 5
unsigned char xdata   assemSiteByte2;       //byte 6
unsigned char xdata   assemSiteByte3;       //byte 7

// Page7

unsigned char xdata   assemSiteByte4;       //byte 0
unsigned char xdata   assemSiteByte5;       //byte 1
unsigned char xdata   assemSiteByte6;       //byte 2
unsigned char xdata   termination;          //byte 3
unsigned char xdata   ccaByte0;             //byte 4
unsigned char  xdata  ccaByte1;             //byte 5
unsigned char  xdata  dcaByte0;             //byte 6
unsigned char xdata   dcaByte1;             //byte 7

////////////////////////////////////////////////////////////////////////////
// GLOBAL VARIABLES
```

```
//
//screen variables
unsigned char xdata   select, scroll, charging;
unsigned char xdata   update_screen;
char xdata            display_string[21];
unsigned char xdata   changed,old_scroll,old_select, old_charge;

//real time variables
int                   month, day, year, timer_now, timer;
double                now;
float                 charge, discharge,voltage, temperature;
float                 capacity, current, maxCellVolt, minCellVolt,voltCell;
unsigned char xdata   c_rate;

//battery related variables
unsigned char xdata   trickle, batt_insert, batt_check;
unsigned char xdata   numROMs, ds2437ROMNum, ds2407ROMNum;
float                 desired_current, trickle_current, tap;
double                start_of_charge;

//2437 variables
unsigned char xdata   page[8], status, data_in[8];

//termination variables
float                 old_volt, new_volt, peak_volt, old_temp, new_temp;
float                 current_deltaTt, last_deltaTt;
unsigned char data    terminate,termination_check,peak;

////////////////////////////////////////////////////////////////////////////
// FUNCTION PROTOTYPES
//
void delta(void);
void bar_graph(void);
void date(unsigned char, unsigned char);
void delay_second(void);
void battery_check(void);
unsigned char access2437(void);
void read_page2437(unsigned char);
void read_data(void);
void pick_screen(void);
void write_status2437(unsigned char);
```

```
//mainbat.c
//This is the main program to test the battery charger
//

#include <ds5000.h>
#include <stdio.h>
#include <string.h>
#include ″ds2437.h″
#include ″display.c″
#include ″1wirewsr.c″
#include ″charge.c″
#include ″screens.c″
#include ″pre_scr.c″
#include ″termin8.c″
//#include     ″setup.c″

////////////////////////////////////////////////////////////////////////////
// WRITES TO STATUS REGISTER
//
//
void write_status2437(unsigned char status)
{
     if(access2437())
     {
        write_byte(write_scratchpad);               //Sends command
        write_byte(0);                              //Sends page number
        write_byte(status);                         //Sends data to be written to
                                                    //the first byte
                                                    //on the page

        ow_reset();
     }
}

////////////////////////////////////////////////////////////////////////////
// READS PAGE FROM DS2437
//
//
void read_page2437(unsigned char pageNum)
{
     unsigned char m;
     if(access2437())
     {
        write_byte(recall_memory);    //Reads page and writes to the scratchpad
        write_byte(pageNum);
        ow_reset();
     }

     if(access2437())
     {
```

```
        write_byte(read_scratchpad);  //Reads scratchpad and writes to the
                                      //variables
        write_byte(pageNum);

        for(m=0; m<8; m++)
        {
                page[m]=read_byte();  //Assigns each byte a page[number] that will
                                      //be assigned
        }                             //a meaningful name outside of this function
        ow_reset();
    }

}

////////////////////////////////////////////////////////////////////////////
//  DATE
//     Unpacks the packed date code
//     Date comes packed in a 16 bit character with:
//                     day           = bits 0–4
//                     month         = bits 5–8
//                     year          = bits 9–15
//
void date(unsigned char date_codeMSB, unsigned char date_codeLSB)
{
    year      =      (date_codeMSB/2) + 1980;
    month     =      (date_codeMSB–(date_codeMSB/2)*2)*8+date_codeLSB/32;
    day       =      date_codeLSB – (date_codeLSB/32)*32;

    if(year<2000)                 //Takes into account years before and after
                                  //the year 2000 to put in a two digit format,
                                  //such as 97, for 1997
    {

        year = year – 1900;
    }
    else
    {
        year = year – 2000;
    }
}

////////////////////////////////////////////////////////////////////////////
//  DELAY_SECOND
//     Enters a delay of one second
//
void delay_second(void)
{
    delay(32000);
    delay(32000);
}
```

```
////////////////////////////////////////////////////////////////////////////
//INITIALIZE DISPLAY
//

void initialize(void)
{
     init_display();   //Initializes display
     set_charge_level(0);  //Initializes charge level to zero to terminate any
                           //preexisting charge condition

}

////////////////////////////////////////////////////////////////////////////
// FIND DEVICES
//
void FindDevices(void)
{
     unsigned char m;

     if(!ow_reset())                         //Begins when a presence is detected
     {
        if(First())                  //Begins when at least one part is found
        {
                numROMs=0;
                do
                {
                        numROMs++;
                        for(m=0;m<8;m++)
                        {
                        FoundROM[numROMs][m]=ROM[m];  //Identifies ROM number
                                                      //on found device
                }
                }while (Next()&&(numROMs<10));       //Continues until no
                                                      //addi tional devices
                                                      //are found
        }
     }

}

////////////////////////////////////////////////////////////////////////////
//IDENTIFY DEVICES
//
void IdentifyDevices(void)
{
     unsigned char m, ROM_family;


     ds2437ROMNum = 0;                       //Number of DS2437's found in search
     ds2407ROMNum = 0;                       //Number of DS2407's found in search
```

```
    for(m=1;m<=numROMs;m++)
    {
       ROM_family = FoundROM[m][0];         //Determines what type of part has
                                            //been found
       switch(ROM_family)
       {
              case 0x1E:                    //ROM_family for DS2437 is 0x1E
              ds2437ROMNum = m;
              break;

              case 0x12:                    //ROM_family for Ds2407 is 0x12
              ds2407ROMNum = m;
              break;
       }
    }

    if (ds2437ROMNum ==0)
    {
       init_display();
       sprintf(display_string,"No DS2437 Found!");
       printd(display_string,0);
       delay_second();
       return;
    }


    if (ds2407ROMNum ==0)
    {
       init_display();
       sprintf(display_string,"No DS2407 Found!");
       printd(display_string,0);
       delay_second();
         return;
    }

}
////////////////////////////////////////////////////////////////////
// 2437 ACCESS
//      Detects if DS2437 is in the path and can be written to
//
unsigned char access2437(void)
{
    unsigned char m;

    if(ow_reset()) return false;
    write_byte(0x55);                       // match ROM
    for(m=0;m<8;m++)
    {
       write_byte(FoundROM[ds2437ROMNum][m]);       //send ROM code
    }
    return true;
```

```
}

////////////////////////////////////////////////////////////////////
//   BATTERY IN SYSTEM CHECK
//      Checks to see if battery is in system
//
unsigned char batt_in_system(void)
{
     batt_check++;
     if(batt_check>5)                //Only makes this check once every 6 times
     {                               //      through the main event loop
        batt_check=0;
        FindDevices();
        IdentifyDevices();

        if(ds2437ROMNum==0)          //If a DS2437 is not found but was at one
                                     //time then this action is taken
        {
                if(batt_insert==true) //i.e.  battery has been removed
                {
                        charging=false;
                }
                batt_insert=false;
                init_display();
                sprintf(display_string,"No DS2437 Found!");
                printd(display_string,0);
                delay_second();
        }

        else
                batt_insert=true;
     }

     return batt_insert;
}

////////////////////////////////////////////////////////////////////////
// REAL TIME CALCULATIONS
//      Reads all real time data and converts it into a usable format
//
void read_real_time(void)
{
     if(access2437())                //initiates a temperature reading
        write_byte(convert_T);
     if(access2437())                //initiates a voltage reading
        write_byte(convert_V);

    read_page2437(0);

        tempLSB =      page[1];  //Contents of Page 0
        tempMSB =        page[2];
```

```
       voltLSB =      page[3];
       voltMSB =      page[4];
       currLSB =      page[5];
       currMSB =      page[6];

    temperature = tempMSB + (tempLSB/256.0);

    voltage = ((voltMSB*2.56) + (voltLSB*.01))/2;

    voltCell = (voltage/numCells);   //Calculated voltage per cell

    c_rate = (fullChgCapMSB*.256)+(fullChgCapLSB*.001);
    current = (((currMSB*1.250048) + (currLSB*.004883))*c_rate);


    read_page2437(1);

       rtcByte0  =    page[0];       //Contents of Page 1
       rtcByte1  =    page[1];
       rtcByte2  =    page[2];
       rtcByte3  =    page[3];
       ica       =    page[4];

    timer_now = rtcByte0;
    now = (rtcByte3*16777216)+(rtcByte2*65536)+(rtcByte1*256)+rtcByte0;
    capacity = ica;

    read_page2437(7);

       ccaByte0 =     page[4];       //Contents of Page 7
       ccaByte1 =     page[5];
       dcaByte0 =     page[6];
       dcaByte1 =     page[7];

    charge = (ccaByte1*256 + ccaByte0)*.32;
    discharge = (dcaByte1*256 + dcaByte0)*.32;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// MAIN PROGRAM
//
void main(void)
{

    unsigned char charge_loop, press_select, press_scroll, press_mode;

    SCON = 0x52;                          //Initializes Baud Rate to 9600
    TMOD = 0x20;                          //
    TCON = 0x69;                          //
    TH1  = 0xFD;                          //
```

```
    scroll=select=charging=0;
    press_select=press_mode=press_scroll=0;
    changed=0;
    old_volt=old_temp=terminate=termination_check=0;
    last_deltaTt=peak=peak_volt=0;
    batt_check=5;
    batt_insert=trickle=false;
    initialize();

    while(!batt_in_system());        //Waits for battery to be inserted

    //create_data();                 //Adds data to unprogrammed parts
    write_status2437(0x07);
    read_data();                     //Reads all static data

////////////////////////////////Main Event Loop/////////////////////////////////////

    while(batt_in_system())                 //Loops when a battery is in system
    {
       read_real_time();                    //Reads all real-time data

       if(press_select==1)                  //Detects SELECT being released after
       {                                    //being pressedin order to select from
              if(SELECT==1)                 // one of the main menu items
          {
              select++;
              press_select=0;
          }
          if(select==1) scroll=11;          //Initializes each menu to its first
                                            //screen
          else if(select==2) scroll=18;
          else if(select==3)
          {
              select=0;//
              scroll=0;
          }

       }
    if(press_scroll==1)                     //Detects SCROLL being released after
                                            //being pressed in order to view
    {                                       //all the contents
          if(SCROLL==1)                     //of each main menu selection
          {
              scroll++;
              press_scroll=0;
          }
         if(select==0&&scroll>10) scroll=1;         //Creates a loop within
                                                    //each main menu
         else if(select==1&&scroll>16) scroll=12;   //
         else if(select==2&&scroll>21) scroll=19;   //
```

```
}

if(press_mode==1)                       //Detects MENU being released after
                                        //being pressed and toggles
{                                       //between charge and discharge
     if(MODE==1)
     {
          if(charging==false)
                 charging=true;
          else
                   charging=false;

          press_mode=0;
     }
}

if(SELECT==0)                           //Detects pressing the respective
                                        //button
          press_select=1;               //
if(SCROLL==0)                           //
          press_scroll=1;               //
if(select==1&&scroll==12&&MODE==0)      //
          press_mode=1;                 //
pick_screen();

   if(charging==true&&(timer_now%7==0))         //If in charging mode, charge
                                                //will be updated on
   {                                            //seven second intervals
   main_charge();
   charge_loop++;
   if(charge_loop>4)                    //Checks terminating conditions
                                        //every fifth time
      {                                 //through the charging loop

          charge_loop=0;
            if(check_terminate())
            {
                 charging=false;        //Ends charging cycle
                 write_full_charge();
                 delay_second();
                 delay_second();
             }
      }
   }

if(charging==false&&capacity<10&&(timer_now%30==0))     //If capacity falls
                                                        //below 10%, a
{                                                       //charging cycle
                                                        //will begin

init_display();                                 //This condition is
```

```
                                                    //checked every
    sprintf(display_string,”  STARTING CHARGE”);    //30 seconds
    printd(display_string,0);
    scroll=12;                                      //Sets to go to Charge screen
    select=1;
    charging=true;

    delay_second();
    delay_second();
       }
    }
}
```

```
//pre_scr.c
//   Reads the initial data and performs the
//   calculations needed for each display screen
//

////////////////////////////////////////////////////////////////////////////
// READ DATA FROM 2437
//      Assigns values to each variable as each page is read from memory
//
void read_data(void)
{
    read_page2437(1);

       ica     =   page[4];

       capacity = ica;

    read_page2437(2);

       disconnectByte0 =    page[0];
       disconnectByte1 =    page[1];
       disconnectByte2 =    page[2];
       disconnectByte3 =    page[3];
       endChargeByte0  =    page[4];
       endChargeByte1  =    page[5];
       endChargeByte2  =    page[6];
       endChargeByte3  =    page[7];

    read_page2437(3);

       manufacturerID  =    page[0];
       chemistry       =    page[1];
       numCells        =    page[2];
       maxCellVLSB     =    page[3];
       maxCellVMSB     =    page[4];
       minCellVLSB     =    page[5];
       minCellVMSB     =    page[6];

       maxCellVolt      =    (maxCellVMSB*.256) + (maxCellVLSB*.001);
       minCellVolt      =    (minCellVMSB*.256) + (minCellVLSB*.001);

    read_page2437(4);

       desPackVLSB      =     page[0];
       desPackVMSB      =     page[1];
       minBattTemp      =     page[2];
       maxBattTemp      =     page[3];
       maxChargeCurr    =     page[4];
       assemDateLSB     =     page[5];
       assemDateMSB     =     page[6];
       chgCtlMode       =     page[7];
```

```
    read_page2437(5);

       fullChgCapLSB    =     page[0];
       fullChgCapMSB    =     page[1];
       deltaVcell       =     page[2];
       deltaTemp        =     page[3];
       deltaTtime       =     page[4];
       packManfByte1    =     page[5];
       packManfByte2    =     page[6];
       lotCode          =     page[7];

    read_page2437(6);

       purchaseDateLSB  =     page[0];
       purchaseDateMSB  =     page[1];
       firstUseDateLSB  =     page[2];
       firstUseDateMSB  =     page[3];
       assemSiteByte0   =     page[4];
       assemSiteByte1   =     page[5];
       assemSiteByte2   =     page[6];
       assemSiteByte3   =     page[7];

    read_page2437(7);

       assemSiteByte4   =     page[0];
       assemSiteByte5   =     page[1];
       assemSiteByte6   =     page[2];
       termination      =     page[3];
       ccaByte0         =     page[4];
       ccaByte1         =     page[5];
       dcaByte0         =     page[6];
       dcaByte1         =     page[7];
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// INFORMATION CALCULATIONS
//     Contains all the calculations needed to prepare the stored information
//             into a format that can be easily displayed on the LCD
//
////////////////////////////////////////////////////////////////////////////
void info_1(void)
{
    char display_stringB[21];      //Displays ROM Number of the DS2407
    sprintf(display_stringB, ”   %bX %bX %bX %bX %bX %bX %bX %bX”,
FoundROM[ds2437ROMNum][0],FoundROM[ds2437ROMNum][1],FoundROM[ds2437ROMNum][2],
FoundROM[ds2437ROMNum][3],FoundROM[ds2437ROMNum][4],FoundROM[ds2437ROMNum][5],
FoundROM[ds2437ROMNum][6],FoundROM[ds2437ROMNum][7]);

    screen_info_1(display_stringB);
```

```
}

////////////////////////////////////////////////////////////////////////////
void info_2(void)
{
    char manuID[12],chem[5];
    switch(manufacturerID)  //Determines the manufacturer of the part

    {
       case 'D':
              strcpy(manuID,"DALLAS SEMI");
              break;

       default :
              strcpy(manuID,"UNKNOWN    ");
    }

    switch(chemistry)       //Determines the type of battery chemistry
                            //involved
    {
       case 0x00:
              strcpy(chem, "----");
              break;

       case 0x01:
              strcpy(chem, "PbAc");
              break;

       case 0x02:
              strcpy(chem, "LION");
              break;

       case 0x03:
              strcpy(chem, "NiCd");
              break;

       case 0x04:
              strcpy(chem, "NiMH");
              break;

       case 0x05:
              strcpy(chem, "NiZn");
              break;

       case 0x06:
              strcpy(chem, "RAM");
              break;

       case 0x07:
              strcpy(chem, "ZnAr");
              break;
```

```
        default:
                strcpy(chem, ”Unk”);
    }

    screen_info_2(manuID,chem);
}

////////////////////////////////////////////////////////////////////////////
void info_3(void)
{
    float cells, nominal_volt, full_cap;

    cells = numCells;                                          //Number of
                                                               //cells in pack
    nominal_volt = ((desPackVMSB*256) + desPackVLSB)/1000.0;    //Nominal volt
                                                               //in mV
    full_cap = ((fullChgCapMSB*256) + fullChgCapLSB)/1000.0;    //Full Charge
                                                               //Capacity in mA
                                                               //hours
                                                               // /1000 to
                                                               //convert to V
                                                               //and A hrs
    screen_info_3(cells, nominal_volt, full_cap);
}

////////////////////////////////////////////////////////////////////////////
void info_4(void)
{
    int pur_month, pur_day, pur_year;
    int st_month, st_day, st_year;

    date(purchaseDateMSB,purchaseDateLSB);  //Sends purchase date to be unpacked
    pur_year  =       year;
    pur_month =       month;
    pur_day   =       day;

    date(firstUseDateMSB,firstUseDateLSB);  //Sends first use date to be unpacked
    st_year  = year;
    st_month = month;
    st_day   = day;

    screen_info_4(pur_month, pur_day, pur_year, st_month, st_day, st_year);
}

////////////////////////////////////////////////////////////////////////////
void info_5(void)
{
    char assembled_by[10];
    int asm_month,asm_day,asm_year;

    date(assemDateMSB, assemDateLSB);       //Sends assembly date to be unpacked
```

```
    asm_year  =       year;
    asm_month =       month;
    asm_day   =       day;

    switch(packManfByte2,packManfByte1)     //Determines the Package Manufacturer
    {
       case 'D','S':
              strcpy(assembled_by,"DALLAS ");
              break;

       default :
              strcpy(assembled_by,"UNKNOWN   ");
    }

    screen_info_5(assembled_by,asm_month,asm_day,asm_year);
}

////////////////////////////////////////////////////////////////////////////
void info_6(void)
{
    int lot_code;          //Displays the Lot Code
    lot_code = lotCode;

    screen_info_6(lot_code);
}

////////////////////////////////////////////////////////////////////////////
void info_7(void)
{
    float max_cell_volt,min_cell_volt;

    max_cell_volt = ((maxCellVMSB*256) + maxCellVLSB)/1000.0;  //Max and min volts
                                                               //per cell in mV
    min_cell_volt = ((minCellVMSB*256) + minCellVLSB)/1000.0;  //  .../1000 to
                                                               //convert to V

screen_info_7(max_cell_volt,min_cell_volt);

}

////////////////////////////////////////////////////////////////////////////
void info_8(void)
{
    float max_temp,min_temp;

    max_temp = maxBattTemp; //Displays the maximum and minimum temperatures

    min_temp = minBattTemp;

    if(max_temp>=128)      //Treats temperatures as 2's compliment numbers
```

```
       max_temp=max_temp – 256.0;

    if(min_temp>=128)
       min_temp=min_temp – 256.0;

    screen_info_8(max_temp,min_temp);
}

////////////////////////////////////////////////////////////////////////////
void info_10(void)
{
    double time_ago;

    time_ago = now/3600.0;  //Converts real–time clock time from seconds
                            //to hours

    screen_info_10(time_ago);
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// CHARGE CALCULATIONS
//
////////////////////////////////////////////////////////////////////////////
void charge_3(void)
{
    float maxCharge;
    char display_stringB[21];

    delta();

    maxCharge = maxChargeCurr/10.0;
    switch(termination)     //Determines termination scheme used in the pack

    {
       case 0:
              sprintf(display_stringB, ”CHG TERM:     –%cV”,deltas);
              break;

       case 1:
              sprintf(display_stringB, ”CHG TERM:     0%cV”,deltas);
              break;

       case 2:
              sprintf(display_stringB, ”CHG TERM:     %cT”,deltas);
              break;

       case 3:
              sprintf(display_stringB, ”CHG TERM:     dT/dt”);
              break;
```

```
       case 4:
              sprintf(display_stringB, ”CHG TERM:     CVCL”);
             break;                          //CVCL = ”constant voltage, current
                                             //limited”

    }

    screen_charge_3(maxCharge,display_stringB);
}

////////////////////////////////////////////////////////////////////////////
void charge_4(void)
{
    float delV, delT, delTt;

    delV      =       deltaVcell;                 //Displays the terminating
                                                  //levels
    delT      =       deltaTemp;
    delTt     =       deltaTtime/10.0;

    screen_charge_4(delV, delT, delTt);
}

////////////////////////////////////////////////////////////////////////////
void charge_5(void)
{
    double last_charge, time_ago;

    read_page2437(2);                //Records time when last charge ended

       endChargeByte0  =    page[4];
       endChargeByte1  =    page[5];
       endChargeByte2  =    page[6];
       endChargeByte3  =    page[7];

    last_charge = (endChargeByte3*16777216)+(endChargeByte2*65536)
+(endChargeByte1*256)+endChargeByte0;

    time_ago = (now – last_charge)/3600.0;  //Time since last charge ended, in
                                            //hours

    screen_charge_5(time_ago);
}

////////////////////////////////////////////////////////////////////////////
void charge_6(void)
{
    double time_charging;

    time_charging = (now – start_of_charge)/3600.0; //Time since current charging
                                                    //cycle began
```

```
    screen_charge_6(time_charging);
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
// GRAPHIC CALUCLATIONS
////////////////////////////////////////////////////////////////////////////
void graphics_1(void)
{
    bar_graph();
                        //Displays the voltage level in a graphical format

    if(voltage>=9.0)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=8.8)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=8.6)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=8.4)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=8.2)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=8.0)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=7.8)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=7.6)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=7.4)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(voltage>=7.2)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
```

```
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(voltage>=7.0)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
     else if(voltage>=6.8)
               sprintf(display_string, "%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(voltage>=6.6)
               sprintf(display_string, "%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar);

     else if(voltage>=6.4)
               sprintf(display_string,"%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar);

     else if(voltage>=6.2)
               sprintf(display_string, "%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar);
     else if(voltage>=6.0)
               sprintf(display_string, "%c%c%c%c%c",bar,bar,bar,bar,bar);
     else if(voltage>=5.8)
               sprintf(display_string, "%c%c%c%c",bar,bar,bar,bar);
     else if(voltage>=5.6)
               sprintf(display_string, "%c%c%c",bar,bar,bar);
     else if(voltage>=5.4)
               sprintf(display_string, "%c%c",bar,bar);
     else if(voltage>=5.2)
               sprintf(display_string, "%c",bar);
     else
               sprintf(display_string, " ");

     screen_graphics_1(display_string);
}

////////////////////////////////////////////////////////////////////////////
void graphics_2(void)
{
     char display_stringB[21];
     unsigned char xdata discharging;
     bar_graph();

     discharging=false;

     if(current<0)      //Makes the current a magnitude, so charge
                        //and discharge current can be displayed
     {

        current=current*(-1);
        discharging=true;
     }                          //Displays the current level in a graphical format
```

```
    if(current>=3.8)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=3.6)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=3.4)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=3.2)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=3.0)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=2.8)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=2.6)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=2.4)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=2.2)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=2.0)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=1.8)
       sprintf(display_string,
"%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=1.6)
       sprintf(display_string,
"%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=1.4)
       sprintf(display_string,
"%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=1.2)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=1.0)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=.8)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=.6)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
    else if(current>=.4)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);
```

```
    else if(current>=.2)
       sprintf(display_string, "%c%c",bar,bar);
    else
       sprintf(display_string, "%c",bar);

    if(discharging==true)                          //Displays Amps as either
                                                   //discharging or charging
       sprintf(display_stringB, "0  DISCHARGE AMPS  4");

    else
       sprintf(display_stringB, "0   CHARGE AMPS    4");

    screen_graphics_2(display_string,display_stringB);
}

////////////////////////////////////////////////////////////////////////////
void graphics_3(void)
{
    bar_graph();

                       //Displays the capacity in a graphical format
    if(capacity>=100)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(capacity>=95)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(capacity>=90)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

else if(capacity>=85)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(capacity>=80)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(capacity>=75)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

    else if(capacity>=70)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

else if(capacity>=65)
       sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
```

```
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=60)
        sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=55)
        sprintf(display_string, "%c%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=50)
        sprintf(display_string,"%c%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=45)
        sprintf(display_string, "%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=40)
        sprintf(display_string,"%c%c%c%c%c%c%c%c%c",
bar,bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=35)
        sprintf(display_string, "%c%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar,bar);

     else if(capacity>=30)
        sprintf(display_string, "%c%c%c%c%c%c",bar,bar,bar,bar,bar,bar);

     else if(capacity>=25)
        sprintf(display_string, "%c%c%c%c%c",bar,bar,bar,bar,bar);

     else if(capacity>=20)
        sprintf(display_string, "%c%c%c%c",bar,bar,bar,bar);

     else if(capacity>=15)
        sprintf(display_string, "%c%c%c",bar,bar,bar);

     else if(capacity>=10)
        sprintf(display_string, "%c%c",bar,bar);

     else if(capacity>=5)
        sprintf(display_string, "%c",bar);

     else
        sprintf(display_string, " ");

     screen_graphics_3(display_string);
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
```

```
//   PICK SCREEN
//      Determines which screen is to be displayed
//
void pick_screen(void)
{
    if(old_scroll!=scroll||old_select!=select||old_charge!=charging)
       changed=1;                           //If any of the possibilites has
                                            //changed since the last loop, update
                                            //the screen, otherwise,only update
                                            //the screen after 11 loops
    if(update_screen>10||changed==1)
    {
       update_screen=0;
       changed=0;

       init_display();
       switch(select,scroll)                //Switches to the screen based on
                                            //select and scroll
       {
       case 0,0:
         screen_information();
          break;

       case 0,1:
           info_1();
           break;

       case 0,2:
           info_2();
           break;

       case 0,3:
           1info_3();
           break;

       case 0,4:
           info_4();
           break;

       case 0,5:
           info_5();
           break;

       case 0,6:
           info_6();
           break;

       case 0,7:
           info_7();
           break;
```

```
        case 0,8:
          info_8();
          break;

        case 0,9:
         screen_info_9();
          break;

        case 0,10:
          info_10();
          break;

////////////////////////////////////////
        case 1,11:
        screen_charge();
        break;

        case 1,12:                          //Charge/discharge status determined
               switch(charging)             //by charging value.
        {
               case 0:
               screen_mode_discharge();     //Charge level set to zero on the
                       if(old_charge==true) //first pass through the loop
                            set_charge_level(0); //following a charge cycle
                old_charge=false;
               break;

               case 1:
                       screen_mode_charge(); //Initial charge is called if this
                       if(old_charge==false) // is the first time through the
                          initial_charge();  // loop in charge mode
                          old_charge=true;
                       break;
        }
               break;

        case 1,13:
               screen_charge_2();
               break;

        case 1,14:
               charge_3();
               break;

        case 1,15:
               charge_4();
               break;

        case 1,16:
```

```
               if(charging==true)                    //This screen is displayed
                                                     //only while charging

               {
                 scroll=17;
                 charge_6();
                 break;
               }
               charge_5();
               break;

    case 1,17:
               if(charging==false)                   //This screen is displayed
                                                     //only while discharging

               {
                  scroll=12;
                 screen_mode_discharge();
                  break;
               }
       charge_6();
       break;

////////////////////////////////////
    case 2,18:
       screen_graphics();
       break;

       case 2,19:
       graphics_1();
       break;

    case 2,20:
       graphics_2();
       break;

    case 2,21:
       graphics_3();
       break;
    }

}
update_screen++;
old_scroll=scroll;
old_select=select;
old_charge=charging;
}
```

```
//screens.c
//   Contains all the information to be included on the display screen
//   for the DS2437 battery charger demo kit
//
//
////////////////////////////////////////////////////////////////////////////
// MAIN MENU SCREENS
//    Make selections with the SELECT button
//
void screen_information(void)
{
     sprintf(display_string, ”INFORMATION”);         //prints statement into string
     printd(display_string,0);                       //prints string to LCD
}

void screen_charge(void)
{
     sprintf(display_string, ”CHARGE/DISCHARGE”);
     printd(display_string,0);
}

void screen_graphics(void)
{
     sprintf(display_string, ”GRAPHICS”);
     printd(display_string,0);
}

////////////////////////////////////////////////////////////////////////////
// INFORMATION SCREENS
//    Contain information on the part and are accessed from
//    the INFORMATION screen using the SCROLL button
//
//
void screen_info_1(char display_stringB[21])
{
     sprintf(display_string, ”    PACK NUMBER”);
     printd(display_string,0);

     printd(display_stringB,1);
}

void screen_info_2(char manuID[12], char chem[5])//////////////////////////////
{
     sprintf(display_string, ”MFG           CHEM”);
     printd(display_string,0);
     sprintf(display_string, ”%s   %s”,manuID,chem);
     printd(display_string,1);
}

void screen_info_3(float cells,float nominal_volt, float full_cap)///////////////
{
```

```
    sprintf(display_string, ”CELLS  NOM   CAP”);
    printd(display_string,0);
    sprintf(display_string, ”  %1.0f    %2.1fV  %2.1fAH”,cells,nominal_volt,
full_cap);
    printd(display_string,1);
}

void screen_info_4(int pur_month,int pur_day,int pur_year, int st_month,
intst_day,int st_year)
{
    sprintf(display_string, ” PURCHASE   START”);
    printd(display_string,0);
    sprintf(display_string, ”%2d/%2d/%2d   %2d/%2d/%2d”,
pur_month,pur_day,pur_year,st_month,st_day,st_year);
    printd(display_string,1);
}

void screen_info_5(char assembled_by[10],int asm_month,int asm_day,int asm_year)//
{
    sprintf(display_string, ” ASSEMBLED   BY”);
    printd(display_string,0);
    sprintf(display_string, ”%2d/%2d/%2d     %s”,asm_month,asm_day,asm_year,
assem–bled_by);
    printd(display_string,1);
}

void screen_info_6(int lot_code)////////////////////////////////////////////////
{

       sprintf(display_string, ”LOT CODE    SITE”);
    printd(display_string,0);
    sprintf(display_string, ”  %3d       %c%c%c%c%c%c%c”,lot_code,assemSiteByte0,
assemSiteByte1,assemSiteByte2,assemSiteByte3,assemSiteByte4,assemSiteByte5,
assemSiteByte6);

    printd(display_string,1);
}

void screen_info_7(float max_cell_volt, float min_cell_volt)//////////////////////
{
    sprintf(display_string, ”MAX CELL VOLT: %3.2fV”,max_cell_volt);
    printd(display_string,0);
    sprintf(display_string, ”MIN CELL VOLT: %3.2fV”,min_cell_volt);
    printd(display_string,1);
}

void screen_info_8(float max_temp, float min_temp)////////////////////////////////
{
    sprintf(display_string, ”MAX TEMP:  %4.1f%cC”,max_temp, degree);
    printd(display_string,0);
    sprintf(display_string, ”MIN TEMP:  %4.1f%cC”,min_temp, degree);
```

```
    printd(display_string,1);
}

void screen_info_9(void)////////////////////////////////////////////////////////
{
    sprintf(display_string, "TOTAL CHARGE: %.1fC",charge);
    printd(display_string,0);
    sprintf(display_string, "TOTAL DISCHG: %.1fC",discharge);
    printd(display_string,1);
}

void screen_info_10(double time_ago)////////////////////////////////////////////
{
    sprintf(display_string, "     ASSEMBLED:");
    printd(display_string,0);
    sprintf(display_string, "  %.1f HOURS AGO",time_ago);
    printd(display_string,1);
}

////////////////////////////////////////////////////////////////////////////////
// MODE:  CHARGE/DISCHARGE MODE SCREENS
//    Allows the user to pick between charge and discharge modes
//    from the CHARGE/DISCHARGE screen by using the MODE button
//
void screen_mode_charge(void)
{
    sprintf(display_string,"       MODE:");
    printd(display_string,0);
    sprintf(display_string,"      CHARGE");
    printd(display_string,1);
}

void screen_mode_discharge(void)////////////////////////////////////////////////
{
    sprintf(display_string,"       MODE:");
    printd(display_string,0);
    sprintf(display_string,"     DISCHARGE");
    printd(display_string,1);
}

////////////////////////////////////////////////////////////////////////////////
// CHARGE/DISCHARGE SCREENS
//   Contain information on the part and are accessed from
//   the CHARGE/DISCHARGE screen using the SCROLL button
//
void screen_charge_2(void)//////////////////////////////////////////////////////
{
    sprintf(display_string, "V: %4.2f V Cap:%4.1f%%",voltage,capacity);
    printd(display_string,0);

    if(current<0)
```

```
        sprintf(display_string, ”C:%4.3fA T:%5.2f%cC”,current,temperature,degree);
    else
        sprintf(display_string, ”C:+%4.3fA T:%5.2f%cC”,current, temperature,
degree);
    printd(display_string,1);
}

void screen_charge_3(float maxCharge, char display_stringB[21])//////////////////
{
    sprintf(display_string, ”MAX CHARGE:   %2.1fA ”,maxCharge);
    printd(display_string,0);

    printd(display_stringB,1);
}


void screen_charge_4(float delV, float delT, float delTt) ///////////////////////
{
    delta();

    sprintf(display_string, ” %cV    %cT    %cT/t”,deltas,deltas,deltas);
    printd(display_string,0);
    sprintf(display_string, ”%3.0fmV %3.0f%cC  %2.1f%cC/m”,delv,   delT,degree,
delTt,degree );
    printd(display_string,1);
}


void screen_charge_5(double time_ago)//////////////////////////////////////////////
{
    sprintf(display_string, ”TIME SINCE LAST ”);
    printd(display_string,0);
    sprintf(display_string, ”CHARGE: %.1f HOURS”,time_ago);
    printd(display_string,1);
}

void screen_charge_6(float time_charging)//////////////////////////////////////////
{
    sprintf(display_string, ”CHARGING FULL:%4.1f%%”,capacity);
    printd(display_string,0);
    sprintf(display_string, ”CHARGE TIME:  %3.1f hr”,time_charging);
    printd(display_string,1);
}

////////////////////////////////////////////////////////////////////////////////
// GRAPHICS SCREENS
//   Contain information on the part and are accessed from
//   the GRAPHICS screen using the SCROLL button
//
void screen_graphics_1(char display_string[21])//////////voltage graph////////////
{
```

```
    printd(display_string,0);
    sprintf(display_string, ”5      VOLTS       9”);
    printd(display_string,1);
}

void screen_graphics_2(char display_string[21],char display_stringB[21])
                                          //currentgraph//////////////////////
{
    printd(display_string,0);

    printd(display_stringB,1);
}

void screen_graphics_3(char display_string[21])/////////capacity graph////////////
{
    printd(display_string,0);
    sprintf(display_string, ”0    CAPACITY   100%%”);
    printd(display_string,1);
}
```

```
//setup.c
//serves as a means to set up data into the ds 2437

////////////////////////////////////////////////////////////////////////////
//WRITE DATA TO 2437
//      Writes data to the DS2437
//
void write_data(unsigned char pageNum)
{
    unsigned char i;
    if(access2437())
    {
       write_byte(write_scratchpad);        //Writes data to the scratchpad
       write_byte(pageNum);                 //Writes data from scratchpad to
                                            //specified page

       for(i=0; i<8; i++)
       {
          write_byte(data_in[i]);    //Writes each byte of data to the
                                     //specified page
       }
       ow_reset();
    }

    if(access2437());
    {
       write_byte(copy_scratchpad);    //Saves by copying to the scratchpad
       write_byte(pageNum);               // to the specified page
       ow_reset();
    }

}

////////////////////////////////////////////////////////////////////
//CREATE TEMPORARY DATA IN 2437
//
void create_data(void)
{
    ////////////////Page 3 data ///////////////////////
                                                      //
       data_in[0] =   'D';                            //manufacturerID
       data_in[1] =   0x03;                           //chemistry
       data_in[2] =   0x06;                           //numCells
       data_in[3] =   0x40;                           //maxCellVLSB
       data_in[4] =   0x06;                           //maxCellVMSB
       data_in[5] =   0x84;                           //minCellVLSB
       data_in[6] =   0x03;                           //minCellVMSB
```

```
write_data(3);

   ////////////////Page 4 data/////////////////
                                              //
   data_in[0] =   0x28;                       // desPackVLSB
   data_in[1] =   0x23;                       // desPackVMSB
   data_in[2] =   0x0A;                       // minBattTemp
   data_in[3] =   0x28;                       // maxBattTemp
   data_in[4] =   0x14;                       // maxChargeCurr
   data_in[5] =   0x65;                       // assemDateLSB
   data_in[6] =   0x22;                       // assemDateMSB
   data_in[7] =   0x0F;                       // chgCtlMode

write_data(4);


////////////////Page 5 data/////////////////
                                              //
   data_in[0] =   0x6C;                       // fullChgCapLSB
   data_in[1] =   0x07;                       // fullChgCapMSB
   data_in[2] =   0x0A;                       // deltaVcell
   data_in[3] =   0x0F;                       // deltaTemp
   data_in[4] =   0x25;                       // deltaTtime ---will be
                                              // divided by 10
   data_in[5] =   'S';                        // packManfByte1
   data_in[6] =   'D';                        // packManfByte2
   data_in[7] =   0x7D;                       // lotCode

write_data(5);


////////////////Page 6 data/////////////////////
                                              //
   data_in[0] =   0x66;                       // purchaseDateLSB
   data_in[1] =   0x22;                       // purchaseDateMSB
   data_in[2] =   0x67;                       // firstUseDateLSB
   data_in[3] =   0x22;                       // firstUseDateMSB
   data_in[4] =   'D';                        // assemSiteByte0
   data_in[5] =   'A';                        // assemSiteByte1
   data_in[6] =   'L';                        // assemSiteByte2
   data_in[7] =   'L';                        // assemSiteByte3

write_data(6);

////////////////Page 7 data/////////////////////
                                              //
   data_in[0] =   'A';                        // assemSiteByte4
   data_in[1] =   'S';                        // assemSiteByte5
   data_in[2] =   0x00;                       // assemSiteByte6
   data_in[3] =   0x03;                       // termination
   data_in[4] =   0x00;                       // ccaByte0
```

```
        data_in[5] =   0x00;                        // ccaByte1
        data_in[6] =   0x00;                        // dcaByte0
        data_in[7] =   0x00;                        // dcaByte1

    write_data(7);

}
```

```
// termin8.c
//  Contains the subroutines for possible terminating conditions
//
////////////////////////////////////////////////////////////////////////////
//   termination is kind of type of termination:                           //
//     0=–delta V                                                          //
//     1=0delta V                                                          //
//     2=delta T                                                           //
//     3=deltaT/delta                                                      //
//     4=CVCL                                                              //
////////////////////////////////////////////////////////////////////////////
unsigned char check_terminate(void)
{
    float delta_v, deltaTt;

    if(now<start_of_charge+30)       //Allows charge cycle to stabalize before
                                     //termination checks are allowed
{
       terminate=false;
       return terminate;
    }

    if(voltCell>maxCellVolt)                //Full Charge when voltage per cell is
                                            //above maxCellVolt
    {                                       //
       terminate=true;
       return terminate;
    }
    new_volt=(voltage*1000)/numCells;       // voltage in mV
    new_temp=temperature;                   // temp in degrees C
    termination_check++;                    //Provides time for the terminating
                                            //scheme to set up current
                                            //and old voltages
    if(termination_check==1)
    {                                       //Reinitializes the peak voltage for
                                            //each new charge cycle
       peak=0;
       peak_volt=0;
    }

    if(new_volt<old_volt&&termination_check>2&&peak==0)
       if(termination==0||chemistry==3&&termination>4)
       {
              peak_volt=old_volt;           //Sets the peak voltage value once the
                                            //voltage begins to drop if
              peak=1;                       //negative delta V is scheme
       }

    else if(termination==1||chemistry==4&&termination>4)
```

```
   peak_volt=old_volt;          //The peak voltage for zero delta V is always
                                //updated so the flat level can be deteceted

delta_v=peak_volt–new_volt;                  //Delta_v from peak to current
                                             //voltage per cell

current_deltaTt=new_temp–old_temp;           //Represents change over last
                                             //30 second window
deltaTt=last_deltaTt+current_deltaTt;        //Represents change over the
                                             //last minute

////////Negative Delta V//////
if(termination==0||chemistry==3&&termination>4)
{
   if((delta_v>=(float)deltaVcell)&&termination_check>2)
   {
          terminate=true;
          termination_check=0;
          return terminate;
   }
}

////////Zero Delta V/////////
else if(termination==1||chemistry==4&&termination>4)
{
   if(delta_v==0&&termination_check>2)
   {
          terminate=true;
          termination_check=0;
          return terminate;
   }
}

else
   old_volt=new_volt;

/////////Delta T/delta t/////////
if(termination==3&&deltaTt>(deltaTtime/10)&&termination_check>2)
{
   terminate=true;                   //If temperature changes by more
                                     //than deltaTtime/10 in a
   termination_check=0;              //one minute interval,
                                     //charging will terminate
   return terminate;
}

else
{
   old_temp=new_temp;
   last_deltaTt=current_deltaTt;
   terminate=false;
```

```
    }
    return terminate;               //A terminate value of true will stop a
                                    //charging cycle
}
```