To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

H8S/2117

Smart Battery Communications

Introduction

This application note accompanies the Smart battery sample code developed for the H8S/2117. The sample program demonstrates how to communicate with a smart battery. Charging and discharging circuits and algorithms are not included in this application note. The software was developed and tested on the YTD08EV211701 and the HP HSTNN-CB69 standard laptop battery

Target Device

H8S/2117 (YTD08EV211701Common KBC Board)

Contents <contents header>

1. Overview

The Smart Battery sample code provides an example of Smart Battery communications. The program sends out commands and populates the Smart battery data structure with information about the smart battery. In a real application this information can then be used by higher layers to determine charging algorithms and other parameters.

2. Running the program

To run the program, download the code to the H8S2117. Connect a smart battery to the I2C channel 0. The program will then send out the series of commands and populate the smart battery data structure with responses received from the battery. Connecting a secondary battery to channel 2 and changing arguments to the battery interrogation functions will allow communication with the secondary battery.

3. The Smart Battery Standard

The Smart Battery Specification was written primarily with the intent to provide a communications interface between a battery and the user system. The specification was written to provide a better user experience when using such a system Aspects of the user experience including regular updates on battery status like remaining charge and estimated usage time, selection of proper algorithms for charging and discharging etc which are better served by this specification. Based on this information, the system is able to regulate performance to maximize battery life or maximize performance or any other user defined performance specification.

The Smart Battery Specification specifies a set of commands that the battery controller can decode and respond to. The spec also defines how to interpret battery responses to a particular command, as well as commands that let the system control certain aspects of battery operation.

3.1 Communication Protocol

At the communications layer, the Smart Battery Specification uses the SMBUS protocol. All smart batteries are equipped with microcontroller that operates as an SMBUS slave device. This device takes care of obtaining and keeping track of all the information regarding the battery like charge state, temperature, estimated charge time etc. In addition to this operational information, the microcontroller also holds static information about the battery like battery address (0x16), manufacturer information, serial number, battery chemistry etc.

The specification [0] is available for download on the smartbattery.org website.

4. Program Description

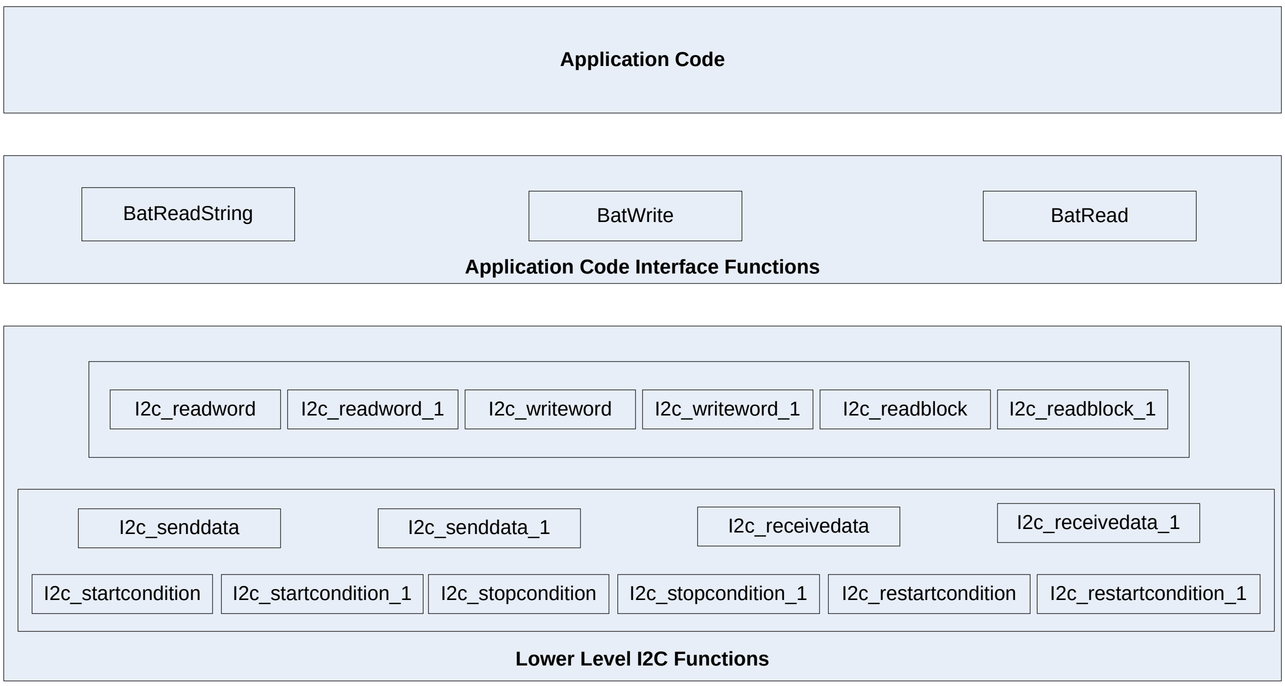

Figure 1: Sample Program Layout

The Main layers in the sample program are

a. Main (Application code)

b. Application Interface Functions: These functions take as arguments Battery-number, command and pointers to response buffers

```
uchar BatReadString (uchar command, uchar* ResponseString,uchar PolledBattery)
uchar BatRead (uchar command, uint* Resp, uchar PolledBattery)
uchar BatWrite(uchar command, uint Mesg, uchar PolledBattery)
```

c. Lower Level I2C functions: These functions are written for individual I2C Channels and are invoked depending on the Battery-number parameter passed to higher level functions.

```
uchar i2c_senddata(uchar sendingdata)
uchar i2c_receivedata(uchar * receivedata)
uchar i2c_senddata_1(uchar sendingdata)
uchar i2c_receivedata_1(uchar * receivedata)
```

The program is configured to work with I2C channels 0 and 1, although it can be easily ported to work with channel 2 as well. The lower level functions use a timeout function for a non-blocking implementation. Timer0 is used as the system tick timer and generates a 1 millisecond system tick. The timeout routine alarm() uses this as its time base.

The sample application uses the Application Interface functions to interrogate a smart battery and populate the battery structure based on the responses received from the battery. The program currently supports the following Smart Battery commands although more can be added easily.

```
#define TEMPERATURE            0x08 // Get cell temperature (word 0.1 degK)
#define VOLTAGE                0x09 // Get cell pack voltage (word mV)
#define CURRENT                0X0A // Get cell pack current (word mA)
#define AVERAGE_CURRENT        0x0B // Get cell avg  current (word mA)
#define PERCENT_FULL           0x0D // Get cell relative state of charge
#define REMAINING_CAPACITY     0x0F // Get remaining capacity (word mAh)
#define RUNTIME_TO_EMPTY       0x11 // Runtime remaining to drain battery (word minutes)
#define BATTERY_STATUS         0x16 // Battery status flags (word)
#define DESIGN_CAPACITY        0x18 // Designed capacity (word mAh)
#define DESIGN_VOLTAGE         0x19 // Designed voltage (word mV)
#define MFG_DATE               0x1B // Manufacture date
#define SERIAL_NUMBER          0x1C // Serial number
#define MFG_NAME               0x20 // Manufacturer name
#define DEVICE_NAME            0x21 // Device name
#define CHEMISTRY              0x22 // Device chemistry (4 character string)
```

Figure 2: Supported Smart Battery commands

The smart battery structure listed below is not an exhaustive one, only a sample of data that can be obtained from the battery. The I2C layer uses the i2c_data structure listed below to store data sent and received during a particular transaction. The higher layers transfer this data to the battery structure.

```
struct i2c_data {
  unsigned char  Slave_ID;     // Slave address
  unsigned char  Byte_count;      // Byte count
  unsigned char   buffer[I2C_BUFFSIZE];     // Received data or send data
    unsigned char   PolledBattery;  // Battery number (Primary or Secondary)
} I2C_DATA;
```

Figure 3: i2c_data data structure

```
Battery_struct
{
 uint     Temperature;           // Battery temperature (command 0x08)
 uint     Current;               // Current (abs value command 0x0A)
 uint     Voltage;               // Battery voltage (command 0x09)
 uint     AverageCurrent;        // Average current (abs value command 0x0B)
 uint     PercentFull;           // Percent full (command 0x0D)
 uint     RemainingCapacity;     // Battery capacity (command 0x0F)
 uint     RunTimetoEmpty;        // Time remaining until empty (command 0x11)
        // Status word read from battery (command 0x16)
 union
  {
      uint    Word;
      struct
      {   // Smart Battery Status bits
       uint ErrorCode       :4; // Error code
       uint FullyDischarged :1; // Battery discharged
       uint FullyCharged    :1; // Battery charged
       uint Discharging     :1; // Battery currently discharging
       uint Initialized     :1; // 1=Battery calibrated and working
       uint TimeAlarm       :1; // 1=Time to Empty less than Remaining Time
       uint CapacityAlarm   :1; // 1=Capacity less than Capacity alarm
       uint Reserve10       :1; // Reserved in SBS spec
       uint DischargeAlarm  :1; // 1=Battery is depleted: stop discharging
       uint OverTempAlarm   :1; // 1=High temperature: stop charging
       uint Reserve13       :1; // Reserved in SBS spec
       uint ChargeAlarm     :1; // 1=Charging must be suspended
       uint OverChargeAlarm :1; // 1=Battery is fully charged: stop charging
      } Bit;
 } Status;

 uint     DesignCapacity;              // Design capacity (command 0x18)
 uint     DesignVoltage;               // Design voltage (command 0x19)
 uint     MfgDate;                     // Manufacture date (command 0x1B)
 uint     SerialNumber;                // Serial number (command 0x1C)
 uchar    MfgName[MFG_NAME_LENGTH];       // Manufacturer Name
 uchar    DeviceName[DEVICE_NAME_LENGTH];// Device Name
 uchar    Chemistry[CHEMISTRY_NAME_LENGTH];   // Device chemistry ('LION')

}BATTERY_STRUCT;
```

Figure 4: Battery_struct data structure

5. Using the Program in your application

To use the program in a separate application, include the I2C_functions.c and I2C_functions.h in the application. The tick timer ISR in intprg.c should be included as well.

Initialize the proper I2c channel by calling I2C_`init()` with the proper argument. Since the I2C function calls use timer0 to determine timeouts, the timer has to be initialized at this point as well by calling the `Init_timer()` function. The main program can be used as a template on how to use the provided functions. The `BatRead()`, `BatWrite()` and `BatReadString()` functions can now be called with proper arguments to retrieve/send data to the smart battery system. The provided main application can be used as a template.

To port the program to work with channel 2, simply reconfigure the corresponding registers from channel 0 or 1 to channel 2. The module stop bit for the individual channel needs to be cleared as well to enable configuring the I2C registers.

Reference

1. http://smartbattery.org/specs/sbdata10.pdf

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sept.10.09	—	First edition issued