Application Report
SLUA543–March 2011

WinCE/Linux Drivers for bq275xx Fuel Gauge

Charles Herder *BMS Handheld*

ABSTRACT

The bq275xx gas gauge integrated circuits (IC) can use an I^2C line for communication with the host system. It is necessary to allow application-layer software on the host system to access parameters made available by the gas gauge IC. Multiple levels of abstraction are between the gauge and a given application. This document describes an example implementation of each of these layers in an effort to provide a programming model for real-world implementations. Project collateral discussed in this application report can be downloaded from the following URL: www.ti.com/lit/zip/SLUA543.

Contents

List of Figures

1 Glossary

Board Support Package (BSP): this is the code that supports WinCE on a given hardware platform.

Platform Support Package (PSP): this is the code that supports Linux on a given hardware platform. Functionally equivalent to a BSP for Linux.

2 Example Hardware and Configuration

For this application report, an AM3517 eXperimenter Kit with the additional display is used. See http://focus.ti.com/docs/toolsw/folders/print/tmdxevm3517.html

This board uses a Texas Instruments AM3517 processor and also contains memory and some basic peripherals. This platform has both WinCE and Linux support packages and is used as the hardware base for the project described in this document.

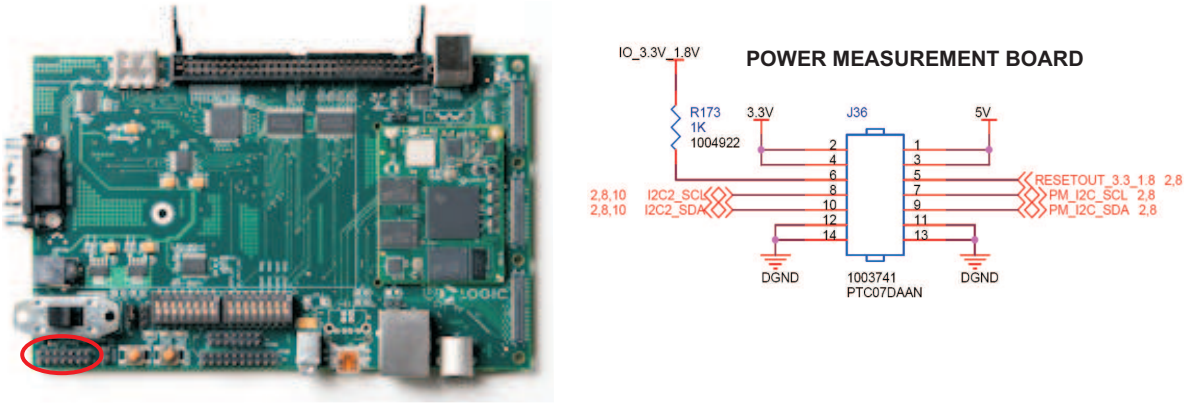

Figure 1. AM3517 eXperimenter Kit

The AM3517 eXperimenter Kit has I²C line brought out to jumper J36, as indicated in Figure 1. The I²C bus connections and ground from this header are connected to bq275xx to allow the processor to communicate with the gauge.

2.1 WinCE Software/Hardware Configuration

The AM3517 eXperimenter Kit is booted from the SD memory card on the port. In order to boot, the board needs three files that are built when you compile your system in Visual Studio: MLO, EBOOT0SD.nb0, and NK.bin. MLO is the start-up code that is called by the board's BIOS. MLO then provides the appropriate environment to run EBOOTSD.nb0, the bootloader. This program provides a menu interface over the RS-232 debug serial line that allows the user to run the image on the SD card or to download one via the network interface. NK.bin is the system image that is booted. It contains the operating system as well as all user applications. This can be loaded off of the SD card or may be downloaded over a LAN from Microsoft™ Visual Studio™. Details on both of these methods are available from Adeneo Embedded in its documentation of the AM3517 eXperimenter Kit Board Support Package (BSP) (http://www.adeneo-embedded.com). MLO, EBOOTSD.nb0, and NK.bin must all exist on the first partition of the SD card, and the partition must be formatted in a FAT filesystem.

2.1.1 Getting up and Running

This project requires the hardware described in Section 2.1 and the following software:

1. Microsoft Visual Studio 2005 SP1 with Platform Studio installed with all updates as of March 2010.
2. The Windows CE BSP for the AM3517 eXperimenter Kit. This is available from Adeneo's Web site, http://www.adeneo-embedded.com, along with instructions on how to install it into Microsoft's Platform Studio.
3. Source code and project files available with this application report are located in the product folder on the TI Web page.

Once Platform Studio SP1 and the Adeneo BSP have been installed, open the project contained in this application report to your solution using the wizard. No changes are necessary, and you can build the solution immediately. This build process takes between 20–45 minutes to complete.

When the compile process completes, three files need to be transferred to the eXperimenter Board: MLO, EBOOTSD.nb0, and NK.bin. Transfer each of these files to the first partition of an SD card and boot the board. The bootloader provides a boot configuration menu over the RS-232 debug serial before booting the image. A successful boot runs Windows CE and provide a complete interface over the touch screen.

You may also configure the bootloader to listen for a boot signal from Visual Studio 2005. In this case, a USB cable needs to be connected to the debug serial port (the kit has an onboard USB to RS-232 converter), and an Ethernet cable to download the image. A crossover cable is needed if the development personal computer's Ethernet PHY does not automatically detect a point-to-point connection. This method is well-covered in Adeneo's BSP documentation. Effectively, you modify the boot configuration through the menu provided by the bootloader, and then use Visual Studio to connect to the device and download the image .

2.2 Linux Software/Hardware Configuration

The Linux system is also booted off of the SD memory card, but the method is different to set up. Similar to the Windows CE setup, the board requires three boot files on the first partition of the SD card, and this partition must have a FAT filesystem. The first file, MLO, is the initial code called by the BIOS. Note that although this file is identical in name to that built in the Windows CE environment, the actual code is different and not compatible. The Linux MLO boot code then loads the secondary bootloader, u-boot.bin. This provides a similar environment to the EBOOTSD.nb0 file built in the Windows CE environment. It provides boot configuration menu over the RS-232 debug serial line and loads the file uImage by default. uImage is an image of the Linux kernel. In general, you want to provide a filesystem on which to run Linux. This is done by adding a second partition to the SD card formatted with the ext3 filesystem. Note any kernel image on this second partition will never be booted. This second partition is only accessed after the kernel on the first partition has been loaded and is running in main memory.

2.2.1 Getting Up and Running

The Platform Support Package (PSP) for the AM3517 eXperimenter Kit is available free from and supported by Texas Instruments. See http://processors.wiki.ti.com for details. Instructions for how to set up the build environment are available at the processor's wiki. Once your build tools and environment are properly set up, build the default configuration as documented in TI's PSP documentation. To build the code as documented, you must have the proper toolchain.

2.2.2 References for Getting Started

- http://processors.wiki.ti.com
- http://processors.wiki.ti.com/index.php/AM3517_On-line_Workshop
- http://processors.wiki.ti.com/index.php/GSG:_AM35x_EVM_Software_Setup
- http://focus.ti.com/docs/toolsw/folders/print/linuxsdk-am35x.html

First, you must build the u-boot image. The source is available from the TI Web site (http://software-dl.ti.com/dsps/dsps_public_sw/psp/LinuxPSP/index.html). Once you install this image, you must also install the appropriate toolchain. For this application report, use the CodeSourcery toolchain available at http://www.codesourcery.com/sgpp/lite/arm/portal/release858.

After extraction and bringing your toolchain into the PATH, clean, configure, and build the u-boot image with the following commands: (run in the root directory of the u-boot source)

```
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm am3517_evm_config
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
```

Make sure that the u-boot tools are in the path:

```
export PATH= (Path to u-boot) /tools/:#PATH
```

Now, build the kernel and the filesystem that you will copy to the SD card to run on the eXperimenter board. Change to the root directory of the kernel source, and run the following commands.

```
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm am3517_evm_defconfig
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm uImage modules
make CROSS_COMPILE=arm-none-linux-gnueabi-
ARCH=arm INSTALL_MOD_PATH=(Path to device filesystem root) modules_install
```

This builds MLO, u-boot.bin, the kernel image, and the filesystem. You must repartition the SD card, and copy the appropriate boot and filesystem files to each partition. Documentation and scripts for performing these actions are available on the wiki for the on-line workshop linked to the preceding URL.

Effectively, you need the first partition to be a FAT partition with the MLO, u-boot.bin, and uImage. The second partition can be EXT3; hold the filesystem that Linux will load.

Once the SD card has been loaded, you can boot the board. The PSP does not contain touchscreen drivers, and therefore does not by default boot with the screen interface. Instead, a shell is exposed via the RS-232 debug serial line. Attach this to the computer, boot the board, and log in as root (no password).

The filesystem image downloaded to the board has Ethernet and SSH support. You can either attach the board directly to your development PC with a crossover cable (a normal cable is acceptable if your computer's Ethernet PHY autodetects point-to-point connections), or to a router to allow LAN access to the board. For simplicity, use this ability to log into the board remotely and download our application-level software.

3 Windows CE

3.1 Running the Example Code

Section 2.1 discussed how to build and download the image to the target board using the code associated with this application report. This code already contains the drivers and application necessary to read gauge information over the I^2C bus. In order to demonstrate this capability, simply make the physical connection and run the application.

Make the connections as shown in Figure 1. Ensure that power is connected to the gauge. Pullup resistors are already attached to the I^2C bus within the eXperimenter Kit. Once this is set up correctly, run the Command Prompt from the start menu. Run the command drvtest.exe, and observe the results. This program prints the values from the gauge's RAM in hexadecimal. If the driver fails to communicate with the gauge, it returns all zero values.

3.2 Understanding the Layers

Now that the gauge from a Windows CE application can be read, consider exactly what is involved in making this work. First, consider the structure of the Windows CE operating system. The code structure is depicted in Figure 2 in a hierarchical format.

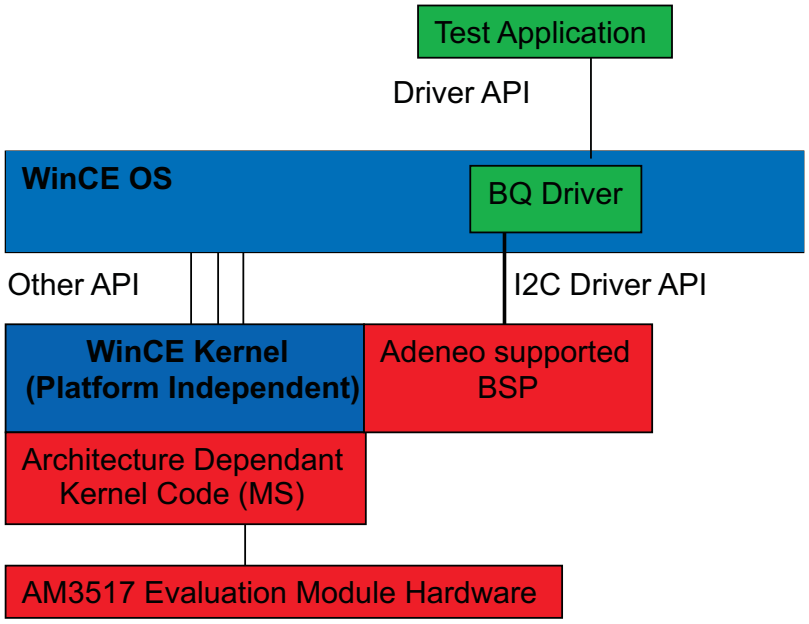

Figure 2. bq275xx Driver in WinCE OS Hierarchy

In the Windows implementation (similar to the Linux implementation), the BSP provides the middleware for the CE operating system and kernel to communicate with the hardware. Most of these modules are built as kernel-mode drivers that provide a standardized API to the kernel and OS.

The bq gauges use the I^2C protocol for communication with the host. Therefore, the driver must have access to raw I^2C communication. because raw I^2C bus access is not required for normal operation of the CE operating system, this is not a part of the standardized API with the CE kernel/OS. In this case, the AM3517 eXperimenter Kit BSP contains a kernel-mode driver for raw I^2C access that exposes an API to user-mode drivers. This kernel driver included in the BSP is platform-specific, and therefore the exposed API may vary between OEMs and their BSP providers. In spite of this incompatibility, the structure of the exposed API must be similar regardless of the hardware chosen. It must provide at initialization, read and write capabilities at the very least. Fortunately, this is all that is needed for this driver.

For this BSP, the user-mode API is included in <sdk_i2c.h>. This user-level API uses an IOControl exposed by the I^2C kernel driver to obtain access to the low-level functions of I^2C read, write, open, close, etc. this project has built a user-mode driver that uses this API and provides a stream driver interface to any CE application. A full discussion of the stream driver interface is available on the MSDN (Microsoft Developer Network), and is not included here. Briefly, however, the stream driver interface exposes functions such as Open, Close, Read, Write, and IOControl. The driver DLL is registered in the CE registry and is accessed in a manner identical to the COM RS232 port (which is also a stream driver interface).

The test application demonstrates how to use the stream driver and read data from the bq gauge.

3.3 Project-Specific Application

Windows CE does not by default provide access to the I^2C bus at the application level. Therefore, in any application, you must build a kernel/user-mode driver that provides this functionality. In the preceding example, a previously written I^2C driver that existed within the board support package was leveraged. Then, some level of abstraction away from the bq gauge interface was provided and brought out the high-level API to the application layer.

Although the steps for a separate platform will be similar from the high level, the details may change significantly. Although it is likely that any board support package for a given platform contains some kind of I^2C driver, it is less likely that it will provide the same API as previously demonstrated. Therefore, you must adapt the previously mentioned source to handle whatever interface the BSP exposes.

Once this has been done, the described methodology of exposing a stream driver interface to the application layer can be used, and you may then design any Windows CE application to use these data.

4 Linux/Android

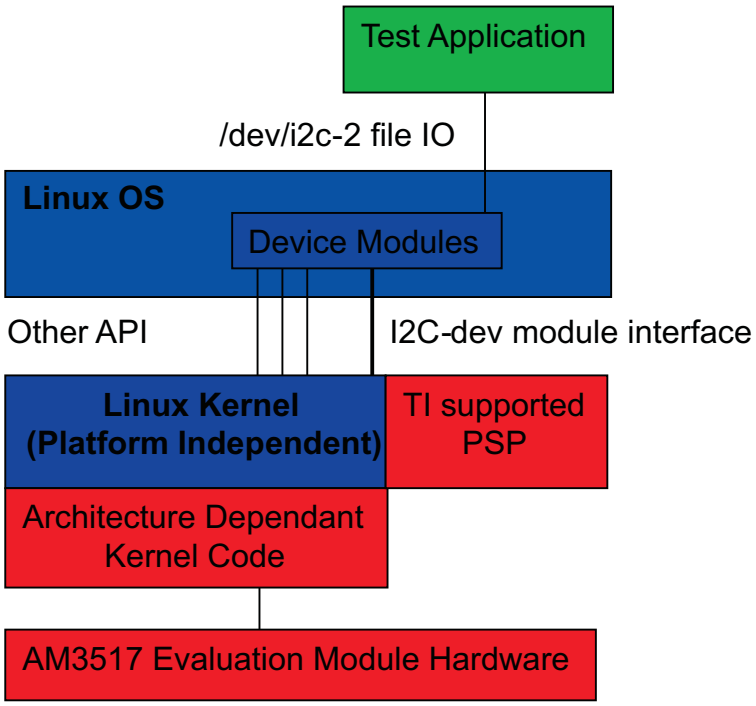

Figure 3. bq275xx Driver in Linux OS Hierarchy

The Linux and Android operating systems are distributed for embedded hardware much in the same way as Windows CE. Because Android runs on a Linux kernel, the Linux kernel is discussed here, but the steps are identical for Android platforms. For a given platform, the platform-independent Linux kernel is available, but must be used with a device-specific Platform Support Package (PSP). This package is essentially identical to the Windows CE BSP. It consists of drivers that provide the kernel and OS access to hardware services as well as start-up code that boots the kernel. Although the architecture is very similar, the differences between Linux and Windows arise when one investigates the means of implementation of each API. For this application, it requires raw access to the I^2C bus. In the Windows CE environment, this requires the development of a driver to interface directly with services provided by the BSP. The Linux kernel can actually be reconfigured to use the hardware-specific I^2C driver contained within the PSP and provide application-level access through a file interface in the /dev directory.

As a result, the development for Linux is substantially easier. You only need to recompile the kernel with the appropriate modules and then build an application that uses this API.

4.1 Reconfiguring and Reloading the Kernel

The PSP for the AM3517 eXperimenter Kit is available free from and supported by Texas Instruments. This PSP and instructions for how to set up the build environment are available in Section 2.2 of this application report and at the processors wiki (http://processors.wiki.ti.com/index.php/GSG:_AM35x_EVM_Software_Setup). Once the build tools and environment are properly set up, modify the build configuration to include the I2C-dev module as preloaded in the kernel build. Change to the Linux source root directory, and run the following commands.

```
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm distclean
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm am3517_evm_defconfig
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm menuconfig
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm uImage modules
make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm INSTALL_MOD_PATH=(Path to device
     filesystem root) modules_install
```

Once menuconfig has run, navigate to Device Drivers → I2C Support → I2C device Interface. Check this feature to be 'Built in'. Also, make sure that the I^2C bus support includes drivers for the OMAP platform.

Once you have rebuilt your kernel, transfer it to the AM3517 board for booting. Copy the uImage file to the first partition (replacing the previous version), and reboot the AM3517 eXperimenter Kit with this image. Note that if you browse to the /dev directory, there are three new files: /dev/i2c-1, /dev/i2c-2, and /dev/i2c-3. These are the three available I^2C buses on the board. Use bus 2 because the pins are easily brought out. Make the same connections to the EVM as described in Section 3.1.

4.2 Loading and Running the Application

Now, build the application to run on the eXperimenter Kit. Build instructions are contained with the code, which is available on the product Web page. Use the CodeSourcery tools to build the application binary.

Once the binary has been produced, you may copy it to the board in two ways. First, if Linux is already booted on the target board, connect the board to a LAN or directly to your development machine via a crossover cable. Configure the IPs properly and use ssh to copy it to the board's filesystem over the network. You may also manually place the application onto the filesystem partition of the SD flash memory before you boot the eXperimenter board from this image. Once this is successful, log into the board over SSH and run the program. Observe hexadecimal values for each of the parameters in the data RAM.

4.3 Understanding the Layers

Unlike with the Windows CE BSP, no additional driver code is needed to provide raw I^2C access, as it can be built into the kernel. The PSP accesses the I^2C hardware engine registers to provide a standard API to the Linux kernel. Next, a standard Linux kernel module brings this API to the /dev directory. The operating system provides basic file I/O routines to allow access to this file. Any application can subsequently access the I^2C bus by reading or writing this file through the standard OS routines at the application level.

The effective difference between the CE and Linux implementations is that the Linux kernel source already contains modules capable of bringing the PSP I^2C driver functionality to the application level. Therefore, your only task is to provide application-level abstraction to reading/writing the bq gauge.

This application-level abstraction is built into "bq.c" and "bq.h". It provides read/write capabilities and raw I^2C access to the gauge. Our test application includes this source to demonstrate the API

4.4 Project Application

Because the Linux kernel already contains the code necessary to bring access to the I^2C bus to the application level, the approach and source used in this application report is applicable to any embedded Linux application with a properly configured board support package.

For a separate platform, you can simply compile the Linux kernel and support package as described in this document. Building the I2C-dev module into the kernel brings the API to the application level in a standardized way. Therefore, the source code attached to this application report is drop-in compatible to any Linux OS.

Although this relative platform independence is convenient, cost is involved. Because the I^2C interface is standardized, access to the I^2C bus is not proprietary, and any application on the embedded Linux device can therefore access the I^2C bus. In certain applications, it may be necessary to secure access to the I^2C bus to a greater extent.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265