Battery Management Unit User Guide

For BCM96x DSL/PON Linux

Version 1.7

5300 California Avenue • Irvine, CA 92617-7013 USA • Phone: (949) 926-5000 • Fax: (949) 926-5203

Table of Contents

Battery Management Unit User Guide

REVISION HISTORY

Revision Number	*Date*	*Change Description*
V1.0	11/24/2013	Initial Release.
V1.1	1/21/2014	More compile instructions, very basic information to get started
V1.2	5/14/2014	Added Introduction, Architecture, Estimated Time Remaining, extended the Shell utility section, added a Logs section, CMS, Interrupt, Energy saving and Reference sections.
V1.3	9/10/2014	Updated the section on Trimmed parts to reflect the change in the Makefile to forbid/allow the code to run on non-trimmed parts
V1.4	9/25/2014	Updated section on CPU frequency control to explain how the ondemand governor is supported. Updated the Wi-Fi and PCIe section to refer to the newly added script and code changes that allow disabling the Wi-Fi interface and powering down PCIe when on Battery.
V1.5	12/23/2014	Clarification on how to compile the module to power down the PCIe interface
V1.6	02/24/2015	Added Starfighter switch power options and API + CMS details
V1.7	02/27/2015	Added a small section on how to compile without CMS

1.0 INTRODUCTION

This document is aimed at understanding how the battery software is integrated into the Broadcom DSL/PON Gateway reference software and how to use it. It does not provide a detailed description of how the battery software is implemented. To obtain more details on the hardware and software battery subsystem, please consult the “Theory and Operation of the Broadcom 40nm Battery Subsystem” (3383-AN101-R in DocSAFE).

2.0 ARCHITECTURE

The battery system is divided in 3 main hardware subsystems and 2 software subsystems. The board supports hardware (FETs and passive components) that are capable of discharging and charging a battery. The FETs are controlled by an independent processor called Pico which runs battery control firmware and which closely monitors the battery voltage using integrated ADCs. Pico firmware is completely responsible for the charge, discharge and monitoring of the battery status and runs independently from the rest of the Gateway chip. Finally the host processor runs battery application code that is responsible for initializing the Pico processor, for validating the battery, for programming Pico with battery characteristics, for controlling life tests (calibration of the battery) for collecting battery data that can be communicated to operators and users.

3.0 COMPILE INSTRUCTIONS AND OPTIONS

Only the software running on the host processor, the “bmud” battery daemon, requires to be compiled. The firmware running on Pico is precompiled and built-in bmud. Certain decisions need to be made prior to compiling bmud.

3.1 Menuconfig

To enable the Battery Management Unit Software, run the menuconfig utility and enable, under “Other Features”, the “Battery Management Unit” option. You must also select the “Battery Configuration” to be either “2-Cell” or “3-Cell”. Do not guess this configuration, it must match the type of battery pack you will be using. These numbers correspond to the number of series cell. “2-Cell” generally corresponds to 7.4 Volts batteries, and “3-Cell” corresponds to 11.1 Volts batteries.

3.2 Board Parameter

A board parameter is defined in boardparms.c:

```
{bp_usBatteryEnable,        .u.us = 1},
```

This board parameter is verified during initialization in bcmdrivers/broadcom/char/board/impl1/board.c and in userspace/private/apps/bmud/bmud.cpp. If the parameter is present and set to 1, board.c will power up the BMU block and bmud.cpp will proceed with battery initialization, otherwise it will quit.

To enable the Battery Management, this board parameter must be added for your board ID in the file shared/opensource/boardparms/bcm963xx/boardparms.c.. Using a board parameter associated to a board ID allows to use the same image on systems that support batteries and systems that don’t.

3.3 Attenuation Resistors

You need to hardcode the values of the VBAT attenuation resistors that are you use on your board. If you do not select the proper resistor settings, you are risking of overcharging the battery and serious damage can result. An incorrect resistor setting can also cause the system to declare a battery as invalid when it should actually be valid. Some examples are defined in userspace/private/apps/bmud/Makefile. Uncomment the appropriate setting for your board, or add your own.

```
# These must match the board if the defaults in battery.c are not
appropriate. Uncomment out or edit as necessary.
# The defaults for 2-Cell packs are 14.7kOhms and 3.24kOhms and for 3-Cell
packs, 30.1kOhms and 4.12kOhms
# These are for a charge voltage of 8.6V with a 2-Cell pack
# CFLAGS += -DVBAT_ATTEN_UPPER_KOHMS=15.4 -DVBAT_ATTEN_LOWER_KOHMS=3.3
# These are for a charge voltage of 8.7V with a 2-Cell pack
# CFLAGS += -DVBAT_ATTEN_UPPER_KOHMS=11.5 -DVBAT_ATTEN_LOWER_KOHMS=2.43
```

3.4 Estimated Time Remaining

When the system is running on battery, it is possible to calculate the estimated time remaining based on battery charge, discharge current and calibration data saved in the battery discharge table. However, when the system is running on utility power, there is no discharge current and an estimate of the system power consumption is necessary to perform the calculation of the estimated time remaining. This power consumption estimate must be set in the variable kNOMINAL_DCHG_WATTS. You can edit the file Custome/MergedNonVolDefaults.h to set this value appropriately, or simply define it in the Makefile. The default value in release software is 5 Watt. If you leave it unchanged and the system consumes a different amount of power when running on batteries, you will notice a discrepancy for the Estimated Time remaining when running on battery compared to when running on utility power.

3.5 Trimmed part support

When chips that support batteries are manufactured, they need to go through a calibration process allowing them to accurately measure battery voltage. This trimming process can only be put in place a certain time after a new chip is introduced, which means that newly introduced chips are not trimmed. Since a non-trimmed part may read the battery voltage inaccurately, charging a battery to its maximum voltage with a non-trimmed part is risky and must not be done. Starting with release 4.16L.02A, the software is delivered in the configuration to be used in production, which is to forbid using non-trimmed parts. In this configuration, the battery controller simply does not run if loaded on a non-trimmed chip. For testing purposes only, you can change this behavior in the bmud Makefile by setting ALLOW_UNTRIMMED_PARTS to TRUE. This will allow charging batteries on chips that are not trimmed, but will restrict the charge level to 75% of the battery capacity. Please consider the following warnings also written in the Makefile:

WARNING: Production code MUST NOT have this setting set to TRUE
WARNING: Production code MUST have ALLOW_UNTRIMMED_PARTS set to FALSE
WARNING: Setting ALLOW_UNTRIMMED_PARTS to TRUE is done at your own risk

After changing this setting, you will need to make clean the bmud application then recompile. Also note that if the code is running on an non-trimmed part and the above flag is set to TRUE, the command “bmu show” will report the max charge level to be 75%, even if the bmud is configured to use a higher charge level.

3.6 Logs

The battery application software can generate logs while it runs. In the current software releases, the logs still require to be enabled at compile time. There are two types of logs that are available. One is a low-level logging mechanism that provides output about the battery status and measurements and is available by changing the makefile to compile with debug enabled:

```
# Uncomment to enable low-level debug prints
# CFLAGS += -DBCM_DEBUG_MSG=1
```

The other is a higher level logging mechanism that provides information about the status of the battery system state machine, in particular while running battery life tests or other similar functions. These logs can be enabled by uncommenting out the following line in BatteryController.cpp:

```
//    fMessageLogSettings-
>SetEnabledSeverities(BcmMessageLogSettings::kInformationalMessages |
BcmMessageLogSettings::kFatalErrorMessages |
BcmMessageLogSettings::kErrorMessages |
BcmMessageLogSettings::kWarningMessages);
```


3.7 Disabling CMS

The battery code can be compiled with CMS disabled. In menuconfig, under “Major Features”, unselect “Use Broadcom Configuration Mgmt System (CMS)”. This will disable some of the code found in the file bmud.cpp and will allow it to run without a CMS integration. The code will however continue to use a few low-level CMS functions to send and receive messages over a socket, available when CMS is compiled out.

3.8 Running the compile

After all the above settings are selected, you can proceed with the normal compile command.

4.0 SHELL UTILITY

An application called bmuctl (short name: bmu) is also available to interact with the battery controller. To see the available commands, type:

bmu

This command will show the battery controller status and the status of batteries connected to it:

bmu show

There are also sub-command that are available. To use an ATE sub-command, prepend the command with the words “ate/”, for example:

bmu ate/forceReadEPROM A

To start a battery life test:

bmu ate/forceLifeTest A

All commands can be abbreviated as long as they can still be distinguished from other commands. For example, the life test command can be invoked as:

bmu ate/forceL A

The prints that are occurring after invoking one of the bmuctl shell utility commands are generated by the bmud application, and not by the bmuctl itself. Because of this, we had to insert special code to allow seeing these prints when invoking the commands from a telnet session instead of the console. However, the output cannot be redirected to a file.

5.0 CMS (CONFIGURATION MANAGEMENT SYSTEM)

CMS is the subsystem used in the Broadcom reference software to maintain a database of configuration and status data and save the necessary information to non-volatile memory. A single file in the bmud application is used to integrate it with CMS. If your system is not using CMS, you will have to review the functionality provided by bmud.cpp and replace this file with an equivalent that integrates with your own CMS.

bmud.cpp is responsible for starting the battery daemon, receiving messages from other applications in the system (for example, bmuctl and httpd), retrieving configuration data from the non-volatile database, transferring battery information back into the non-volatile database, and reporting status information to a volatile section of the database. All this functionality remains necessary but is affected if you migrate the battery system to a different CMS.

6.0 API

There are two types of API available for the battery controller. A high-level CMS-based API and a low-level non-CMS based API. The high-level CMS-based API is documented in userspace/private/include/bmu_api.h and is available for obtaining status information generated by the battery controller, making it available to other applications in the system such as the HTTP daemon or the CLI interface. This high level API is just a wrapper to a lower-level non-CMS based API which can best understood by reading through bmud.cpp. A product that is not using CMS can modify bmu_api.h and bmud.cpp to be independent of CMS while still providing access to the same status information.

Additional low-level APIs are used in bmud.cpp to update control data and save battery pack data. The control data configures the battery controller at initialization and can also change the controller settings during runtime. In our reference design, the control data is saved in flash memory by the CMS or its equivalent to ensure that the configuration persists across reboots. The battery controller does not implement the data persistence. It is the responsibility of CMS or a similar system to implement data persistence and to program the battery controller at initialization and when the configuration is changed.

The battery pack data is an internal representation of the calibration data for the last 2 battery packs that were used with the product. This calibration data is computed by the battery controller during Life Test and is used in case a battery pack is removed then re-inserted or in case the system is rebooted. The CMS system or equivalent is responsible for storing this data in a persistent manner and restoring it on a reboot.

In the next sections, the information accessible through the API is listed.

6.1 Controller status information

BmuControllerStatusInternalGet:

```
    BatteryController->Version
    BatteryController->BuildDateTime
    BatteryController->OperatingOnBattery
    BatteryController->State
```

```
    BatteryController->NumberOfPresentBatteries
    BatteryController->InputVoltage
    BatteryController->Temperature
    BatteryController->EstimatedMinutesRemaining
    BatteryController->BatteryCurrent
    BatteryController->OperatingOnBattery
    BatteryThread->PowerSourceStartTime
```

6.2 Battery status information

```
BmuBatteryStatusInternalGet:
    BatteryController->BatteryPresent
    BatteryController->BatteryValid
    BatteryController->BatteryBad
    BatteryController->BatterySelected
    BatteryController->BatteryFullyCharged
    BatteryController->BatteryChargeLow
    BatteryController->BatteryChargeLowPercent
    BatteryController->BatteryChargeDepleted
    BatteryController->BatteryChargeStateUnknown
    BatteryController->BatteryChargeCapacity
    BatteryController->BatteryActualCapacity
    BatteryController->BatteryFullChargeVoltage
    BatteryController->BatteryDepletedVoltage
    BatteryController->BatteryMeasuredVoltage
    BatteryController->BatteryPercentCharge
    BatteryController->BatteryEstimatedMinutesRemaining
    BatteryController->PackTemperature
    BatteryController->BatteryLifeTestCount
    BatteryController->BatteryLastLifeTest
    BatteryController->BatteryNextLifeTest
    BatteryController->BatteryLifeTestState
```

Note that the battery state of health can be calculated using the above data:

```
    batteryStateofHealth = 100*batteryActualCapacity/batteryChargeCapacity
```

Also available (but not in shown in sample application)

```
    BatteryController->GetBatteryEpromParameter(…)
```

6.3 Controller configuration

The Battery Controller configuration can also be set:

```
BmuControllerConfigInternalSet:
    BcmBatteryNonVolSettings->PollIntervalSeconds(…)
    BcmBatteryNonVolSettings->MaxChargePercent(…)
    BcmBatteryNonVolSettings->UpsConfigLowBattTime(…)
    BcmBatteryNonVolSettings->ReplacementThresholdPercent(…)
    BcmBatteryNonVolSettings->LowChargePercent(…)
    BcmBatteryNonVolSettings->LifeTestingEnabled(…)
    BcmBatteryNonVolSettings->LifeTestPeriod(…)
    BcmBatteryNonVolSettings->LifeTestTOD(…)
```

```
BcmBatteryNonVolSettings->PackVoltageSmoothingSteps(…)
BcmBatteryNonVolSettings->BoardVoltageSmoothingSteps(…)
BcmBatteryNonVolSettings->PackTemperatureSmoothingSteps(…)
BcmBatteryNonVolSettings->BoardTemperatureSmoothingSteps(…)
BcmBatteryNonVolSettings->AllowedFaults(…)
```

6.4 Battery pack persistent information

The following battery pack data must be read periodically (by CMS or an equivalent) and updated in flash if changed. The API provides a method for saving battery pack information for two batteries.

```
nvi->flags
nvi->EstLifeRemaining
nvi->DesignCapacity
nvi->maxWh
nvi->guid
nvi->LifeTestCount
nvi->LastLifeTestStarted
nvi->NextUpdate
nvi->totalSeconds
nvi->seriesResistance
nvi->dchgVoltages
```

7.0 INTERRUPT

Pico generates an interrupt to the host processor when switching from utility power to battery mode. Our own code does not currently make use of this interrupt, however, one may want to use this to immediately reduce power consumption of the system when transitioning to battery, for example by controlling GPIOs that immediately shuts off external, power hungry subsystems.

The interrupt handler is available in the power management module in bcmdrivers/broadcom/char/pwrmngt/impl1/bmu.c. The interrupt handler invokes a kernel task in high priority when the interrupt triggers. You can register a function with this task and it will be invoked soon after the interrupt triggers. To register a function, you can use **`bcmRegisterBmuHandler`** and to de-register, **`bcmDeregisterBmuHandler`**. These functions are defined in bcmdrivers/broadcom/include/bmu.h.

8.0 DYING GASP SUPPORT

In DSL Gateways, a Dying Gasp mechanism must be supported to inform the Operator when a system is powering down. This Dying Gasp is implemented by installing large capacitors on the Gateway that are able to inject power after utility power has failed. When utility power fails, the voltage starts dropping and a voltage monitoring circuit generates a Dying Gasp interrupt to the Host CPU informing it of an imminent power failure. The voltage monitoring circuit needs to trigger at a fairly high voltage (normally around 10.5 Volts) to allow enough time to the software to react to the interrupt and transmit a Dying Gasp message to the Operator.

In a system supporting batteries, when utility power fails, the batteries take over and the Gateway can continue working normally. However, the batteries operate at a lower voltage (from ~6 to ~8.7 Volts) than

the dying gasp trigger voltage. Therefore, when batteries are present, the Dying Gasp interrupt must be disabled. When batteries are absent, the Dying Gasp interrupt can be re-enabled.

The Gateway may also fail when battery charge level becomes critically low. The software is able to monitor this situation and generate the Dying Gasp message without using an interrupt.

9.0 SPECIAL CONSIDERATION FOR NAND FLASH

In systems supporting NAND flash, the time available after the Dying Gasp interrupt has triggered is also necessary to safely stop write operations to the NAND flash. When utility power is off and running on batteries, there can be two conditions where power will fail. First, if the battery becomes critically low, the software can generate Dying Gasp and safely stop FLASH writes. However, if instead the battery is yanked out, a different solution is needed to prevent having any pending write to NAND flash. Our solution is to make the filesystem partitions running from FLASH to be read-only when utility power is unavailable and running from batteries. This mechanism is not yet available in the reference software but a product using NAND Flash should consider solving this problem.

10.0 ENERGY SAVING OPTIONS

When switching to battery mode, the reference software does not currently automatically turn off unnecessary subsystems to save energy. The main reason is that each operator has different requirements regarding what can be turned off at which moment. However, the reference software provides ways to shut down different subsystems such as USB, SATA, Ethernet PHY, or reduce the energy consumption of other subsystems such as the Host CPU. This is not a final list. We are in the process of extending the number of subsystems that can be shut down.

10.1 USB and SATA

The script /etc/init.d/disk.sh can be invoked to suspend and resume the USB and SATA interfaces. This command actually powers down the externally attached device and also shuts down the internal USB and SATA components in the chip, saving as much power as possible.

To manually suspend USB and SATA:

```
/etc/init.d/disk.sh suspend
```

To manually resume USB and SATA:

```
/etc/init.d/disk.sh resume
```

10.2 Switch and Ethernet

Starting with release 4.16L.04, a script is available to shutdown or suspend the Starfighter switch on the 63138 or 63148. The command has the format:

```
switch.sh <suspend|resume>  [-t <wan|lan|all>] [-l level]
```

Use suspend to bring the switch into low power state, and resume to bring it back to regular mode. The command supports optional level or type arguments.

The type argument (-t) has the following possible values:

wan) only suspend/resume the WAN interfaces (eth0)

lan) only suspend/resume the LAN interfaces (eth1, eth2, eth3, and eth4).

all) suspend/resume both WAN and LAN interfaces (default)

The level argument (-l) describes what level of powerdown is being requested. If a level is specified for a suspend operation, the same level must be specified in the subsequent restore operation. Currently only levels 1 and 9 are available. Levels 2 through 8 are reserved for future use. The available levels are as follow:

1) This results in a 'light' powerdown. Once powered down, the system is fully recoverable using the restore option within a few seconds.

9) This results in a complete powerdown. It causes all userspace applications and modules associated with the Starfighter to be unloaded from memory, and will completely shut down the Starfighter switch. Once in this state, a cold reboot of the whole system is needed to restore the switch to a usable state. Note that resuming with level 9 causes the modem to reboot.

Some example usages:

To bring down all lan interfaces

```
switch.sh suspend –t lan
```

To recover lan interfaces

```
switch.sh resume –t lan
```

To bring modem into deep powersavings

```
switch.sh suspend –l 9
```

To recover from deep powersavings (note: causes modem to reboot):

```
switch.sh resume –l 9
```

On older chips or releases, the switch.sh script is not available and instead, the commands below provide an intermediate solution. These commands perform a 'light' (recoverable) powerdown:

To bring a lan interface (eth1-eth4) down

```
ethctl eth1 phy-power down
```

To bring a lan interface (eth1-eth4) back up

```
ethctl eth1 phy-power up
```

To bring a wan interface(eth0) down

```
ethctl eth0 phy-power down
ethctl phy serdespower eth0 2;  #if port is serdes
```

To bring a wan interface(eth0) up

```
ethctl eth0 phy-power up
ethctl phy serdespower eth0 1;  #if port is serdes
```

Notice that powering down the wan interface does not completely disable the port. A serdes device will still autodetect a plugged in device, and restart itself if one is found.

10.3 CPU Frequency

Under normal operations, the CPU already implements features to save Energy. One feature consists of disabling internal clocks when waiting for interrupts to occur (WAIT instruction on MIPS, WFI instruction on ARM). Additionally, it is possible to adjust the CPU frequency based on the amount of work it needs to do, using the CPUFREQ feature from Linux. This feature allows setting the CPU frequency manually or automatically. For battery mode operations, the CPU frequency can be set manually. For normal mode operations, to save power, the CPU frequency should be set automatically using the on-demand governor of the CPUFREQ feature.

These commands are available to control the CPU frequency.

Display the current governor used by the system:
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

List available governors in the system:
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors

Select the userspace governor (manual frequency setting):
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

List available cpu frequencies (in kHz):
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies

Show current frequency (in kHZ):
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq

Set cpu frequency to 750000kHZ
echo 750000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed

Select the ondemand governor:
echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

Display time spent in each of the available frequencies:
cat /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state

To use the ondemand governor, we had to disable the kernel Linux feature called Scheduler Runtime Sharing (BRCM_SCHED_RT_SHARE). We provided control to this setting through menuconfig (make menuconfig; Kernel Configuration Selection; Scheduler RT RQ sharing). By disabling this setting, realtime processes that are tied to a cpu will not be able to block non-realtime processes such as the one supporting the ondemand governor for extended periods of time. Starting with release 4.16L.03, we also made the ondemand governor be the default governor, through menuconfig (make menuconfig; other features; Automated CPUFREQ clk divider). By enabling this setting, ondemand becomes the default governor and cpu frequency becomes adjusted on demand. For more information about cpufreq and governors, please consult the linux documentation found in the folder kernel/linux-3.4rt/Documentation/cpu-freq in the BCA Software release.

10.4 Wi-Fi and PCIe

Starting with release 4.16L.02A, a script is available to uninstall the wireless modules and shut down the PCIe interfaces on the 63138 and 63148. To invoke this script, use these commands:

To suspend the service
/etc/init.d/wifi.sh suspend

To resume the service
/etc/init.d/wifi.sh resume

If you see this error message when invoking the script:
rmmod: can't unload 'bcm63xx_pcie': unknown symbol in module, or unknown parameter
it is because you have not compiled in the small module that handles the PCIe power control. The module can be included in the build by using “make menuconfig”, under “Kernel Confuguration Selection”, under “Misc Drivers”, select "PCI Express repower module".

On older chips or releases, the suspend feature is not available and instead, the commands below provide an intermediate solution. The -i wl0 option is only needed if your system has multiple wireless interfaces. In such case, you can also control the other interface by replacing wl0 with wl1 in the commands.

To bring wifi down
wlctl -i wl0 down

To bring wifi up
wlctl -i wl0 up

To increase the energy saved when Wi-Fi is brought down using the “wlctl down” command, you should make sure that PCIe ASPM is compiled into your system and enabled. To enable PCIe ASPM at compile time, in menuconfig, make sure that the feature “PCIe L1 Active State Power Management” is enabled. At run time, you can also verify that aspm mode is properly enabled by running the following command and verifying that it reyuirns a value other than 0.

To verify that ASPM is enabled on the Wireless interface, look for non-zero return value
wlctl -i wl0 aspm

11.0 REFERENCES

“Theory and Operation of the Broadcom 40nm Battery Subsystem” (3383-AN101-R in DocSAFE)