Android Kernel Logbook

Starting point:

Testing device: LG NEXUS 5 (hammerhead)
Android Version: 5.01 Lollipop
Base Android Kernel: 3.4 lollipop-release
Building toolchain: arm-eabi-4.6
Building on OS: Ubuntu 14.10

Repository

https://github.com/pplant/powerm_kernel/tree/master/module_powerm

Setting up the environment

1) Create a base folder that contains all sources, tools, etc

```
$ mkdir AndroidKernel
```

2) Clone/Checkout kernel sources. (should create a folder msm)

```
$ git clone https://android.googlesource.com/kernel/msm.git
```

```
$ cd m
```

3) Create supporting bash file. (within AndroidKernel folder)

```
$ nano run_this_android.sh
```

Add the following lines to the file:

```
export CC=$(pwd)/arm-eabi-4.7/bin/arm-eabi-
export CROSS_COMPILE=$(pwd)/arm-eabi-4.7/bin/arm-eabi-
export ARCH=arm
export SUBARCH=arm
export PATH=$PATH:$(pwd)/andorid_boot_tools_bin
```

Change modifiers and source file

```
$ chmod +x run_this_android.sh
$ source run_this_android.sh
```

luis> I managed to compile the goldfish kernel last week using the above steps

Note:The source command apparently works only for one terminal session therefore you need to re-run this command every time you open a new terminal window when building the kernel.

4) Clone bootimg-tools and build them. You should clone it from your base folder(AndroidKernel). **For the next commands open a new terminal window otherwise the build process of the bootimg-tools will fail. Because of the previously set environment variables (toolchain).**

```
$ git clone https://github.com/pbatard/bootimg-tools.git
$ cd bootimg-tools
$ make
$ cd cpio
$ gcc mkbootfs.c -o mkbootfs -I../include
```

Now return to the base folder (AndroidKernel) and run the following commands:

```
$ mkdir andorid_boot_tools_bin
$ cd andorid_boot_tools_bin/
$ cp ../bootimg-tools/mkbootimg/mkbootimg .
```

```
$ cp ../bootimg-tools/mkbootimg/unmkbootimg .
$ cp ../bootimg-tools/cpio/mkbootfs .
```

Preparing for the kernel build/deployment process

In order to flash our custom kernel to the device we need the stock boot.img file which we are going to open/extract and repack again after our custom kernel has been placed inside.

https://dl.google.com/dl/android/aosp/hammerhead-lrx22c-factory-0f9eda1b.tgz

Update!!!! Since the stock boot.img has no super user access we need to use a rooted boot.img

http://download.chainfire.eu/580/SuperSU/nexus5-hammerhead-lpx13d-kernel.zip?retrieve_file=1

Extract the package, unzip the file and copy the boot.img within a new folder called boot_img created within the base folder.

Building and deploying

0) Clean environment before every build

```
$ make clean && make mrproper
```

1) Run the following command from your base folder

```
$ source run_this_android.sh
```

2) Prepare and compile the android kernel by running the following commands

```
$ cd msm
```

Open the Makefile and change the extraversion

```
$ make hammerhead_defconfig
$ make menuconfig
$ make savedefconfig
$ mv defconfig arch/arm/configs/pm_defconfig
$ make -j4
```

3) Unpack the original boot.img

```
$cd .. (navigate back to the base folder)
$ unmkbootimg -i boot_img/boot.img
```

b4) Copy your zImage-dtb (kernel) to the unpacked boot.img and repack it again

```
$ cp msm/arch/arm/boot/zImage-dtb kernel
$ mkbootimg --base 0 --pagesize 2048 --kernel_offset 0x00008000
--ramdisk_offset 0x02900000 --second_offset 0x00f00000 --tags_offset
0x02700000 --cmdline 'console=ttyHSL0,115200,n8
androidboot.hardware=hammerhead  user_debug=31 maxcpus=2
msm_watchdog_v2.enable=1' --kernel kernel --ramdisk ramdisk.cpio.gz -o
boot.img
```

5) Deploy (flash) your new boot.img to the device

```
$ adb start-server
$ adb reboot bootloader
$ sudo fastboot boot boot.img
```

Source:http://marcin.jabrzyk.eu/posts/2014/05/building-and-booting-nexus-5-kernel

HelloWorld Module

A simple kernel module printing out a line when loaded and when unloaded. The module is loaded into the kernel and not build into the kernel.

Prerequisites:

- Uncomment/Add the following lines with the .config file which is contained in the msm folder

  ```
  CONFIG_MODULES=y
  CONFIG_MODULE_UNLOAD=y
  ```

- Rebuild the kernel
- Flash/Boot the kernel to your device

Update: Keep in mind every time you run the command make mrporper your changed .config file will be deleted and you have to reinsert the above lines!!!

Note: In order to load the module the device/kernel needs to be rooted!

1) Create a new folder within the base folder

```
$ mkdir module_helloworld
```

2) Create the module source file and its content.

```
$ nano helloworld.c
```

Add the following lines to the previously created file:

```
#include <linux/module.h>  /* Needed by all modules */
#include <linux/kernel.h>  /* Needed for KERN_ALERT */

MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("Peter Plant, 2015");
MODULE_DESCRIPTION("hello world module");

int init_module(void)
{
      printk(KERN_INFO "Hello android kernel...\n");
      return 0;
}

void cleanup_module(void)
{
      printk(KERN_INFO "Goodbye android kernel...\n");
}
```

3) Create a makefile for your module

```
$ nano Makefile
```

Add the following lines to the previously created file:

```
VERSION = 3
PATCHLEVEL = 4
SUBLEVEL = 0
EXTRAVERSION = .1-energy-aware

obj-m := helloworld.o
PWD := $(shell pwd)
KERNEL_DIR = $(PWD)/../msm
HOST_KDIR=/lib/modules/$(shell uname -r)/build
default:
        $(MAKE) ARCH=arm CROSS_COMPILE=$(PWD)/../arm-eabi-4.6/bin/arm-eabi-
-C $(KERNEL_DIR) M=$(PWD) modules
host:
        $(MAKE) -C $(HOST_KDIR) M=$(PWD) modules
```

```
clean:
	$(MAKE) -C $(KERNEL_DIR) M=$(PWD) clean
	$(MAKE) -C $(HOST_KDIR) M=$(PWD) clean
```

Note: When creating the file make sure that $(MAKE) is preceding a **tab** (not multiple space chars). Otherwise the build process will fail

5) Build the helloworld module

```
$ make (within the module_helloworld folder)
```

Run this command if you want to build your model for a android device. Our make file contains also a command to build the module such that it can loaded into the linux kernel on your ubuntu. This allows you to faster test your model.

```
et$ make host
```

Note: A module build with this command **can't be loaded into the android kernel** due to different architectures

Once the build process has finished you should see a set of new files, we are looking for the **helloworld.ko** file.

Note: It is recommended to clean before each build! (make clean)

6) Transfer helloworld.ko to your device and load it into the kernel

```
$ adb push helloworld.ko sdcard/hallworld.ko
$ adb shell
```

All following commands are executed on the command line interface of your device

```
$ su
# insmod sdcard/helloworld.ko
# dmesg (in case this command doesn't work exit the adb shell and run adb shell dmesg)
```

You should now see the line you printed out when your helloworld module is loaded.

Source: http://forum.xda-developers.com/showthread.php?t=1236576

Power Management Module - Kernel patches

In order to facilitate the sharing of changes done on the kernel source code we added a folder patches to the base folder. This folder contains patch files (files with *.patch)

Copy all of those files within your msm folder and run the following command for each patch file:

```
$ git am < *.patch
```

It is recommended to create and checkout a new branch before applying the patches. Copy/move the created files to the patch folder outside msm and commit them.

In order to create a new patch make sure your are in a new branch, all changes are committed and the code is compilable (NO ERRORS). Run the following command to create a new patch (a new patch file is created for each commit done on the branch).

```
$ git format-patch android-msm-hammerhead-3.4-lollipop-release
```

Power Management Module - Read battery status

The power management module should execute periodically a portion of code which reads the current battery status. The next step will be to take some actions according to the current battery status.

Prerequisites: See halloworld module prerequisites

The following function shows the code which reads the battery capacity by calling a method of the battery driver (power supply). It gets called periodically by the timer used within the module.

```
static void read_battery_state(unsigned long data)
```

```
{
	int result = 0;
	union power_supply_propval capacity_val;
	printk(KERN_INFO POWER_TAG": Capacity couldn't be read\n");
	result =
psy->get_property(psy,POWER_SUPPLY_PROP_CAPACITY,&capacity_val);
	if(!result){
		int capacity = capacity_val.intval;
		printk(KERN_INFO POWER_TAG": The charge level is
%d\n",capacity);
		check_power_class(capacity);
	}else{
		printk(KERN_INFO POWER_TAG": Capacity couldn't be read\n");
	}

	//Read expires such that timer calls method again
	read_timer.expires = jiffies + DELAY * HZ;
	add_timer(&read_timer);
}
```

Sources:

http://stackoverflow.com/questions/16920238/reliability-of-linux-kernel-add-timer-at-resolution-of-one-jiffy

Power Management Module - Power classes

Since our power manager needs to be progressive (dependent on current battery capacity) we introduced 9 power classes. Each power class is related to a certain battery capacity (current battery level percentage) range and when activated different actions are performed in order to reduce the overall power consumption.

Each power class is contained in a separate c-file identified by power_class_Y.c where Y is the power class identifier. A power class has the following basic structure:

```
#include "include/power_class.h"
#include "include/powerm.h"
#include <linux/kernel.h>

static void activate(void)
{
    printk(KERN_INFO POWER_TAG": Activated power class X\n");
}

const struct power_class_ops *POWER_OPS_X = &((struct power_class_ops){
activate });
```

If a new power class is selected the activate function is called which will start the procedure to change kernel settings/power_classes in other modules. This function gets called by a “kind of” polymorphism pattern.

Sources:

http://stackoverflow.com/questions/1225844/what-techniques-strategies-do-people-use-for-building-objects-in-c-not-c

Power_class /Actions	**Screen Brightness**	**Vibration**	**Auto Screen off**	**Max volume lock**	**Auto switch off bluetooth**	**Auto switch off wifi**	**GPU underclocking**	**CPU governor**
P X	100%	Default	Default	Default	Not active	Not active	Default	X
P 8	80%	Default	80%	Default	Suspended state	Suspended state	Default	8
P 7	80%	Default	60%	Default	Suspende	Suspende	Default	7

					d state	d state		
P 6	70%	50%	60%	Default	All states	Suspended state	Default	6
P 5	70%	50%	50%	50%	All states	Suspended state		5
P 4	50%	50%	50%	50%	All states	All states		4
P 3	50%	50%	30%	50%	All states	All states		3
P 2	30%	50%	30%	30%	All states	All states		2
P 1	20%	Off	20%	30%	All states	All states		1
P 0	10%	Off	20%	30%	Forced	All states	Minimum	0

As already mentioned a power_class is identified by an id. The concept behind those id's is rather simple the current battery level (shown in percentage) is divided by 10 the result returns the current power class (e.g 50/10=5 -> power class 5). There exists a special case the power class X which falls in the range of 100 to 90 battery level. For this power class no actions are done, since the battery level is high and no power saving settings are needed.
Every setting done will have an impact on the phone's UX. Therefore power saving settings are done progressively. The lower the battery level the stricter the power settings are going to be.

Screen brightness:
Setting the screen brightness according to the power level seems to be trivial but could have a measurable impact on the battery consumption.

Range/Unit: 1-255
Phone state: Working state
Estimated impact: 2/4
Impact on user: 3/4

Auto screen off:
Setting the time until the screen is automatically switched off could potentially reduce power consumption.

Phone state: Working state

Estimated impact: 1/4
Impact on user: 1/4

Max volume lock:
Setting to lock the volume at a certain battery level such that headphones or speakers consume less energy. Impact is rather low because users are not using the speaker all the time.

Phone state: Working state
Estimated impact: 1/4
Impact on user: 3/4

Auto switch off bluetooth:
This setting should periodically check if the bluetooth is being used. In case it is not the bluetooth is switched off. In suspended state it will be completely switched off. As a radical step the bluetooth can not be used at all in power_class 0.

Phone state: All states
Estimated impact: 1/4
Impact on user: 1/4

Auto switch off wifi:
This setting should periodically check if the wifi is being used. In case it is not the it is switched off. In suspended state it will be completely switched off. Wifi is commonly used therefore an intelligent wifi power manager could gain some battery life.

Phone state: All states
Estimated impact: 2/4
Impact on user: 2/4

GPU underclocking:
This is more or less same as the cpu underclocking. It is maybe more visible to the user because of leggy graphics.

Phone state: All states (for suspended state underclocking set to minimum)
Estimated impact: 1/4
Impact on user: 3/4

CPU governor:
The idea here was to introduce power_classes within the governor and depending on the current power class certain actions should be taken. The overall goal is to lower the voltage led to the cpu which leads to less power consumption.

Phone state: All states
Estimated impact: To be determined
Impact on user: To be determined

Further ideas:

- Switch of leds (will only have a very small impact) (auxdisplay folder)
- 3G and phone signal (very important)
- Switch vibrations of

Power Management Module - Screen brightness

In order to set the screen brightness within the power management module we need to do some changes within the Android Kernel source code. Since the backlight driver provides no interface to read the lcd backlight current level directly we added the following function to the the ../msm/driver/video/lm3630_bl.c

```
int lm3630_lcd_backlight_get_level(void)
{
        int level = 0;
        if (!lm3630_dev) {
                pr_warn("%s: No device\n", __func__);
                return 0;
        }

        pr_debug("%s: level=%d\n", __func__, level);
        level = bl_get_intensity(lm3630_dev->bl_dev);
        return level;
```

```
}
EXPORT_SYMBOL(lm3630_lcd_backlight_get_level);
```

In order to use this function from the power management controller we added the following line to the header file of the backlight driver (../msm/include/linux/platform_data/lm3630_bl.h)

```
int lm3630_lcd_backlight_get_level(void)
```

The last change has been done in a configuration file which sets the default values for the backlight drive (../msm/arch/arm/boot/dts/msm8974-hammerhead/msm8974-hammerhead-misc.dtsi). Replace the following line:

```
lm3630,min_brightness = <0x05>;
```

with

```
lm3630,min_brightness = <0x01>;
```

This setting decreases the minimum backlight strength which allows us to save some extra energy in the lowest power class.

Patches:

```
0001-Added-get-level-function-in-backlight-driver.patch
0002-Added-get-maximum-level-function-to-backlight-driver.patch
0003-Bug-fixed-that-failed-the-compilation.patch
```

Power Management Module - Vibration

The vibration driver within the Android Kernel was missing an interface in order to control it from another module. Therefore we implemented an interface that allows us to control the vibration strength from our power management module.

We added the following lines of code to the vibrator driver (msm8974_pwm_vibrator.c):

```
void msm8974_pwm_vibrator_gain(int gain)
```

```
{
      forced_gain = gain;
h}
EXPORT_SYMBOL(msm8974_pwm_vibrator_gain);
```

and the line which uses the forced_gain is the following:

```
msm8974_pwm_vibrator_force_set(vib, forced_gain, pwm);
```

Patches:

```
0004-Added-interface-to-vibrator-driver-in-order-to-set-t.patch
```

Power Management Module - Progressive CPU Governor

We decided to implement the CPU underclocking (maybe little undervolting) over the cpu governor since statically setting the cpu clock frequencies could lead to situations where the actual cpu frequency is higher than needed.
Therefore we used the OnDemand Governor (default governor) as starting point. We added some little changes to increase the powersave_bias property according to the current battery capacity. In addition to that we changed the properties up_threshold and down_differential in order to reduce up samplings and increase down_samplings.

We also set dynamically the maximum frequencies used by the governor. With lowering battery capacity also the maximum frequencies will be reduced. Especially when running on the low power classes a limited maximum frequency will has negative effect on the device performance therefore those frequency are still subject to change.

up_threshold: defines what the average CPU usage between the samplings of 'sampling_rate' needs to be for the kernel to make a decision on whether it should increase the frequency. For example when it is set to its default value of '95' it means that between the checking intervals the CPU needs to be on average more than 95% in use to then decide that the CPU frequency needs to be increased.

powersave_bias: this parameter takes a value between 0 to 1000. It defines the percentage (times 10) value of the target frequency that will be shaved off of the target. For example, when set to 100 -- 10%, when ondemand governor would have targeted 1000 MHz, it will target 1000 MHz - (10% of 1000 MHz) = 900 MHz instead. This is set to 0 (disabled) by default.

Sources:
https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt
https://code.google.com/p/milestone-overclock/wiki/SmashingTheAndroidKernel

Power Management Module - Wifi switch off

(Not possible)

```
networkType enum values are:
0: WCDMA Preferred
1: GSM only <-- This would be "2G" on GSM networks
2: WCDMA only <--WCDMA is "3G" on GSM networks. You may know it as HSPA
3: GSM auto (PRL)
4: CDMA auto (PRL)
5: CDMA only <-- This would be "2G" on CDMA networks
6: EvDo only <-- EvDo is "3G" on CDMA networks
7: GSM/CDMA auto (PRL)
8: LTE/CDMA auto (PRL)
9: LTE/GSM auto (PRL)
10: LTE/GSM/CDMA auto (PRL)
11: LTE only 12: "unknown"
```

Power Management Module - GPU underclocking

Also here we decided to not statically underclock the gpu instead we are changing the allowed power level inside the gpu this is a number which goes from 0 to 6 (attribute name pwrlevel) whereas 0 is the max performance and 6 the lowest performance. Every power level is connected to a clock frequency defined:

arch/arm/boot/dts/msm8974-gpu.dtsi

Sources:

http://www.dailyphonetricks.com/lock-adreno-gpu-clock/

Power Management Module - I/O Scheduler

http://forums.androidcentral.com/general-help-how/293130-my-battery-38-hour-charge-tutorial.html
http://forum.xda-developers.com/nexus-4/general/info-mbqs-cpu-guide-thread-tips-io-t2429096