Paper

Low Power Techniques for an Android Based Phone

Thimmarayaswamy Krishnappa*, Non-member, Mary D'Souza*, Non-member, Golla Varaprasad*a), Non-member

Android is the latest trend in mobile operating systems. Even though Android provides a complete set of application, middleware and Linux kernel for the phone applications developer, it does not fully utilize several standard kernel features. This work attempts to address the limitations of Android specific to power management at kernel level and proposes possible solutions for active and static power management in Linux to overcome these limitations. The developed solutions for active power management include selection of suitable governor algorithm and modification of its parameters and implementation of a daemon process, which performs voltage and frequency scaling. Application level low power techniques for Android are also proposed to help application developers to optimize their software. The objectives of the work are realized by designing and implementing low power techniques for OMAP3530 based Beagleboard. The functionality of the implemented low power techniques is verified through several test cases using benchmarking applications such as 2D/3D rendering, playing a movie file and decompressing. The test execution results are presented in terms of performance, execution time, total current consumed, standby time and battery life. From these tests, it is concluded that through low power techniques it is possible to achieve up to 29% increase in battery life and up to 117% increase in standby time at no or very little performance degradation.

Keywords : Android, Power Management, Dynamic Voltage and Frequency Scaling, Open Multimedia Application Processor, Beagleboard.

1. Introduction

In recent years, there has been a tremendous shift in user preferences in mobile market segment. With rapid advancements in chip technology and software, the mobile phone technology has grown 'out-of-box' and a mobile phone is no more used just for making calls or messaging. With inclusion of mobile internet connectivity, gaming, multimedia applications and not to mention the increased usage of social networking services, the Smartphone has become 'soul-mate' for millions of users. Although Original Equipment Manufacturers (OEMs) of Smartphone's have taken care to provide 'acceptable' battery-life fulfilling the need for portable computing, entertainment and communication, processor specific power optimization features provided by chip vendors have not been utilized properly. The focus of this work is to fill the gap by designing and implementing low power techniques for an 'Android' based Smartphone. Although Android has been chosen as the platform for implementation, the suggested low power techniques apply equally to other Linux based mobile software architectures as well. Android has been chosen among others for its rapidly increasing popularity in the Smartphone market and a promising software framework, which is open source. Following section introduces the reader to Android framework in detail.

1.1 What is Android Android is a software stack, which includes Linux kernel as underlying operating system, middleware software such as application frameworks and libraries and the some of the applications specific to mobile platform. With the inclusion of these three major components Android provides a complete set of software for a mobile platform. Although Android is built on the top of Linux kernel and relies on the kernel for its various OS subsystems such as scheduler, memory management, process execution, device drivers, networking support etc., it adds its own enhancements to the mainline kernel. So even though some literatures claim Android itself as operating system, it is not but a software framework or stack built on the top of Linux kernel.

2. System Development

2.1 System Requirements The requirements to fulfil the objective of the work are listed below,

- The hardware platform should run on Linux kernel with Android drivers.
- The hardware platform should run Android.
- The processor should include working power management and frequency/voltage scaling drivers in the kernel.
- The processor should support low power modes such as sleep and deep sleep.
- Low power techniques should cover active and passive power management at Linux and Android levels.
- The Android level techniques should give proper coding guidelines to make use of Android power management framework for efficient application development in Java.
- Designed low power techniques should not bring down the performance of the system and hence user experience, but should balance power/performance trade-off.
- The architecture independent software should be portable across different hardware platforms.

a) Correspondence to: Dr.G.Varaprasad. E-mail: varaprasad555555@yahoo.co.in
* B.M.S.College of Engineering, Bangalore 560 019, India

2.2 Hardware Requirements Table 1 shows the requirements required for Linux environment to download and build Android root file system and kernel[12],

Table 1. Hardware requirements.

Feature	Minimum Requirements	Remarks
Processor	ARM926 based	Android is designed targeting primarily at ARM based handset
Memory	128MB RAM/256 MB Flash	For runtime memory requirements and to boot Android root file system from NAND/NOR flash
Storage (optional)	SD/MMC CARD	For storing user applications and multimedia data
Primary Display	DVI-D monitor or QVGA TFT LCD with touch screen	Android supports touch screen interface and QVGA/HVGA displays of up to 2.8” screen size
Debug Interface	Minicom or USB-NET based ADB (Android Debug Bridge)	For pushing application files and to browse root file system and kernel files
Navigation	QWERTY keypad or USB keyboard / mouse	Android supports qwerty keypad with 5-way navigation by default

2.3 Software Requirements Following are the tools/utils requirements for Linux environment to download and build Android root file system and kernel [12],

- git 1.5.4 or newer and the GNU Privacy Guard
- JDK 5.0, update 12 or higher
- flex, bison, gperf, libsdl-dev, libesd0-dev, libwxgtk2.6-dev (optional), build-essential, zip, curl
- For developing Java applications for Android following software packages are required,
- Android SDK (Linux / Windows version)
- Eclipse IDE (Java version)
- Android Development Tools (ADT)
- Android Emulator

2.4 Design Specifications The Linux power management design specifications for the proposed work are partially derived from CELF Specifications V1.0R2, which is commonly used across different types of consumer electronics products including mobile phones. The CELF specifications classify Linux power management into two categories,

- Platform suspend/resume or Static power management
- Active power management

The platform suspend/resume specification defines static or passive power management of the system, i.e. when the processor is not executing any process or in other words user is not interacting with the phone. The static power management refers to power saving during longer idle periods, where as dynamic power management refers to power saving during very short idle periods [11]. Current work focuses on platform suspend/resume and dynamic power management. With respect to these two features, following design specifications are derived,

1) Static Power Management Specifications: System should enter suspend or low power modes as supported by processor, such as standby and deep sleep modes.

- 'standby/wakeup' button to enter and exit suspends state.
- Memory should be retained during the sleep and deep sleep modes.
- Turn off unused clocks and voltage sources during the low power modes.
- System should be capable of waking up and return to run mode, either due to 'wakeup' button press or due to preconfigured time out.
- System should have acceptable wake up latency.

2) Active Power Management Specifications: An efficient kernel-space or user-space power policy manager or governor algorithm should be selected for active power management. The selected policy manager or governor should provide user programmable settings during runtime to adapt the policy manager behaviour to dynamic load requirements.

3) Android Power Management Specifications: The Android power management design specifications are defined by Android Open Source Project in its Platform Developer's Guide. The guide states that, 'Android supports its own Power Management (on top of the standard Linux Power Management) designed with the premise that the CPU shouldn't consume power if no applications or services require power'.

2.5 System Block Diagram Fig 1 illustrates different layers involved in the power optimization design specific to Android. The dotted blocks represent the layers involved in the design and implementation of low power techniques.

Here Android applications make use of APIs provided in the Android power manager class, which in-turn has rooted and hard-coded access to Linux kernel level drivers, functions and other interfaces (/proc and /sys).

Android-omap-kernel-2.6.29 does not include cpufreq and power management drivers. These drivers are still under development and testing stage in separate OMAP-PM branch, which doesn’t include Android drivers. Hence either Android drivers have to be ported to OMAP-PM branch or cpufreq or power management drivers have to be ported to android-omap-kernel-2.6.29. The present implementation follows the latter method. To avoid kernel version dependency issues, instead of OMAP-PM-2.6.33-rc8 (latest), OMAP-PM-2.6.29 (older) kernel is used in extracting cpufreq and power management related driver files and integrating them into android-omap-kernel-2.6.29.

Also OMAP-PM branch includes architecture specific drivers for clock and power management, which need to be integrated with the Android kernel. The architecture independent cpufreq and power management drivers make use of architecture dependent clock and power management drivers (low level) to implement and export frequency/voltage scaling and power management functionality to kernel or userspace (through sysfs interface). The governor algorithms use sec and third dotted blocks for handling frequency scaling policies.

3. Design of Low Power Techniques

This section deals in detail design of Linux specific low power techniques and Android specific low power techniques. Linux specific implementations run at kernel level and are fairly independent of Android layers.

3.1 Linux Low Power Techniques

As per the design specifications, Linux power management is divided into active and static power management. Following sections explain the design of low power techniques for both active and static power management.

1) Active Power Management: The Linux active power management part is explained in detail in this section. Active power management refers to when the processor is executing applications or in other words user is engaged with the phone. The OMAP-PM-2.6.29 branch supports frequency and voltage scaling and other power management features. But the limitation is that the Ondemand governor only allows minimum sampling rate to be 5 sec where as minimum sampling rate of Conservative governor is fixed at 50 sec. It is not feasible with a sampling rate of 50 sec, to achieve acceptable power saving by frequency scaling technique. Hence, the proposed work only relies on Ondemand governor for employing frequency/voltage scaling method as one of the low power techniques. The workaround solution for having lower minimum sampling rate value is to write a user space daemon process, which would function similar to Ondemand or Conservative and offer lower sampling rate values. It will calculate the system workload with a sampling rate as selected by user and perform frequency scaling as per the load requirements. Such a process is implemented using Userspace governor, which exports cpufreq interfaces to user land though sysfs. The proposed work attempts to develop a daemon process with important parameters borrowed from Ondemand/Conservative such as up_threshold and down_threshold.

The user-space daemon process shown in fig 2 makes use of signals to generate timing interrupt, which is configurable by the user. It decides the sampling rate value. For example, if the signal raises an alarm every 1 sec, which means the sampling rate, is equal to 1 sec. The process stays in sleep mode until signal alarm is raised. On raise of signal alarm, the process reads /proc/stat interface and extracts following parameters,

- User time
- Nice time
- System time
- Idle time
- IO wait time

Nice time and IO wait times are ignored as test cases used do not consider niced processes as well as IO bound processes. Hence nice time and IO wait times are added with idle time to obtain effective idle time. Active percent is calculated as follows,

Total time elapsed = idle time + system time + user time Idle time elapsed = idle time – last idle time Idle percent = (idle time elapsed * 100)/total time elapsed

Active percent = 100–idle percent

The daemon process switches to highest Operating Performance Points (OPP) (OPP4=550MHz), if the active percent is greater than or equal to up_threshold. If its less than or equal to down_threshold, then process switches to next lower OPP, otherwise it stays at the last selected OPP. Between the consecutive samplings, the process goes to sleep mode.

2) Static Power Management: For using omap-pm-2.6.29 branch for various power management features, debug file system has to be enabled first [13].

To mount debugfs,

```
# mount -t debugfs /debug
```

As mentioned previously in design specifications, following static power management features are implemented in OMAP3530 processor [13].

- Suspend

To suspend the system to low power mode with full chip retention during suspend mode, following instruction need to be used,

```
# echo mem > /sys/power/state
```

- Resume

To wake-up from suspend mode a time out setting is used,

```
# echo '15' >
/debug/pm_debug/wakeup_timer_seconds
```

To wake-up from suspend mode ‘user’ button on Beagleboard is pressed as an interrupt. To wake-up from suspend mode through serial console, user need to press and hold any of key on host keyboard for few sec.

- Data Retention

To retain data during deep sleep mode, following instructions need to be used,

Sleep while idle - System enters low power mode, when no activity is observed for longer time.

```
# echo 1 >
/debug/pm_debug/sleep_while_idle
```

Voltage off while idle – turn off VDD1 and VDD2 voltage sources upon entering low power mode.

```
# echo 1 > /debug/pm_debug/voltage_off_while_idle
```

Enable off mode- turn off power domain transitions upon entering low power mode

```
# echo 1 > /debug/pm_debug/enable_off_mode
```

3.2 Power Management Framework

wake_lock is the mechanism developed by Google to manage the android power management framework. A wake_lock prevents the system from entering suspend or other low-power states when the wake_lock is active. By gaining a wake_lock the android essentially indicates that it needs the CPU attention (and other mobile hardware resources) and the CPU should not slow down to the state saving power. Thus Android applications and services are required to gain wake_lock’s in order to request CPU resources [7].

A locked wake_lock, depending on its type, prevents the system from entering suspend or other low-power states. There are two settings for a wake_lock:

- WAKE_LOCK_SUSPEND: prevents the system from suspending
- WAKE_LOCK_IDLE: prevents going into a low-power idle state where response times are large

Android kernel provides the APIs such as wake_lock and wake_unlock to handle these locks.

All Android power management calls follow the same basic steps [7]:

1. Acquire handle to the PowerManager service
2. Create a wake_lock and specify the power management flags for screen, timeout, etc
3. Acquire wake_lock of suitable type mentioned earlier
4. Perform rest of the application’s operation
5. Finally release the previously acquired wake_lock

There are different types of wake_lock’s used under different circumstances and as per needs of the application:

- ACQUIRE_CAUSES_WAKEUP
- FULL_WAKE_LOCK

- ON_AFTER_RELEASE
- PARTIAL_WAKE_LOCK
- SCREEN_BRIGHT_WAKE_LOCK
- SCREEN_DIM_WAKE_LOCK

It is advantageous to use the lowest level wake_lock as much as possible. Also specifying user-defined timeouts rather than the default values gives better and results.

Following code gives the idea of using wake_lock's within an application [7].

Example:

```
/* Step 1 */
PowerManager pm = (PowerManager)
mContext.getSystemService(Context.POWER_SERVICE);
/* Step 2 */
PowerManager.WakeLock wl =
pm.newWakeLock(
PowerManager.SCREEN_DIM_WAKE_LOCK |
PowerManager.ON_AFTER_RELEASE, TAG
);
/* Step 3 */
wl.acquire();
/* Step 4 */
// Rest of the application's operations
/* Step 5 */
wl.release();
```

wake_lock's are implemented in different ways at the Android kernel level (these are controlled using Android standard APIs used by applications).

Driver API: A driver uses the wake_lock API by adding a wake_lock variable to its state and calling wake_lock_init, wake_lock_destroy, wake_lock_timeout.

User-space API: There is also a Kernel user-space interface. Writing a wake_lock name to /sys/power/wake_lock establishes a lock with that name, to release the lock we need to write the name to /sys/power/wake_unlock

Power management is achieved at its best by handling the low level (hardware component level) power management by the applications, i.e. putting the device on to sleep mode when not being used instead of waiting for auto suspend. The best solution is to combine Linux specific low power techniques with Android specific techniques to achieve better results.

4. Testing and Validation

4.1 Test Plan The test plan includes testing the functionality of hardware platform and software such as Android porting to Beagleboard, NFS sharing between host and target, working of ADB, USB keyboard interface and working of Linux power management drivers through debugfs and sysfs interface. Then the implemented low power techniques are tested for their functionality by running test applications with governor or user-space daemon process executing in the background.

The cpufreq driver source code is modified to output kernel debug messages whenever a frequency change occurs. Test scripts are written to run test applications with the different parameter settings for up/down thresholds. An example test script includes following steps,

1. switch to Ondemand/Userspace governor
2. set up/down threshold values (Ondemand or daemon process)
3. start daemon process
4. log time_in_state
5. execute test application 1
6. sleep for 10 sec
7. execute test application 2
8. log time_in_state

#/sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state interface provides the amount of time spent in each of the frequencies supported by the CPU. Time units here are 10ms. By logging time_in_state before and after the execution of test applications we will come to know in what frequency how much time was spent during the execution of all test applications, which helps to calculate total current consumed, which in turn helps to calculate battery life.

1) Test Environment Setup: Fig 3 shows the test setup. Beagleboard is interfaced with necessary peripherals to operate Android such as USB mouse (optional) and USB keyboard. Since Beagle board is not capable of supplying current to USB devices connected to OTG port (limitation is 100mA), a self-powered USB hub is required to which mouse and keyboard or any other USB devices are connected. The Beagle board only supports HDMI interface. Hence, a HDMI-to-DVI-D cable is used to display running Android screen on DVI-D monitor. The Beagleboard is connected to host machine running minicom utility using RS232 debug port. From minicom, it is possible to browse the rootfs and kernel from the host keyboard in command mode only. A voltmeter (not shown in figure) is con connected across jumper J2 on the Beagleboard, which taps current supplied to the board (either from USB or 5V power socket).

2) Test Applications: Based on the literature reviews, following test applications were chosen for testing the implemented low power techniques.

1. 2D benchmark suite
   - Arcs
   - Fill rate
   - Circles
   - Rectangles
   - Alpha blending
2. 3D benchmark suite
   - Colored cube
   - Lighting
   - Textures
   - Blending
   - Fog
   - Reflection
   - Multi-texture
   - Tea pot
   - Gears
3. Decompression
   - 10MB .tar.gz file
4. Movie play (.avi)
   - 480x272 pixel resolution
   - 25 fps frame rate

2D tests are considered as low-processor intensive applications, where as 3D tests are considered as high processor intensive applications. Both are capable of pushing processor workload to near 100%. Apart from above tests, decompression and playing a movie file of .avi format are also carried out as test cases to see how governor and daemon process behave for very low processor intensive applications. These two applications are capable of pushing processor workload to more than 40-60% but less than

90%. Initially, all tests are carried out keeping processor running constantly at a fixed OPP, with no policy or daemon running in background. Although OMAP3530 supports OPP5 (600MHz), it is considered as 'over-clocking' and not recommended due to stability reasons. And also android-omap-kernel-2.6.29 doesn't support OPP5. Hence all tests are carried out for OPP1 to OPP4 only. The performance of 2D and 3D benchmarking applications is measured in terms of frames per sec (fps) and the total execution time in sec, where as performance of decompressing and movie play test applications is measured in terms of execution time in sec. Current consumed by the board at particular OPP is measured first. Then upon running the test applications, using /sys/cpufreq/stats/time_in_state interface, it's calculated in what OPP how long the processor stayed during test application execution. Accordingly, total current consumed in each test case is calculated to arrive at battery life value, assuming a battery of capacity of 1000 mAh.

3) Battery Life Calculation: A battery of 1000mAh capacity is assumed for calculations. To obtain the battery life for a particular case, following calculations are carried out,

1. x = execution time (sec) / 3600 = execution time in units of hour

2. y = x * OPP current = total current consumption at particular frequency

3. z = 1000 / y = battery life in units of hours

Example: Assuming For OPP1, Execution time (sec) = 98.92 sec then

x = 98.92/3600= 0.0274 hours

y = x * 156 mA = 4.2865 mA

z = 1000 / y = 233.288 hours

4.2 Test Results Fig 4 shows the test results in terms of battery life measured in hours. Compared to OPP1 (lowest frequency 125MHz), all other scenarios give lesser battery life, which is obvious. But by default cpufreq enabled Android kernel is configured with OPP4, hence it gives best possible performance but at the cost of higher power consumption. Hence power and performance comparisons carried out are with OPP4. Ondemand governor with up_threshold value of 40, 60 and 90 give battery life of more than 200 hours for 2D test cases as evident from fig 4.

Similarly, for 3D test cases, the battery life obtained is more than 100 hours. Fig 5 shows the test results of decompression and video play. At OPP1, video (.avi format) was not able to play, hence it is neglected. In comparison to OPP4, Ondemand with up_threshold value of 90 and user-space daemon with up_threshold value of 90 and down_threshold value of 80 give better battery life of more than 90 hours, which is comparable with that of OPP2.

Fig 6 shows the standby time (hours) achieved by applying different combinations of static power management techniques. From the testing carried out, results obtained are summarized as below,

- For high processor intensive applications (2D/3D), Ondemand governor gives 10% to 16% increase in battery life compared to OPP4.
- For lower processor intensive applications (uncompressing/movie play), Ondemand gives 22% to 29% increase in battery life compared to OPP4.
- For lower processor intensive applications (uncompressing/movie play), users-pace daemon process gives 17% to 25% increase in battery life compared to OPP4.
- On applying all possible static power management features it is possible to achieve up to 117% increase in the stand by time.
- Although Ondemand gives better results, it only switches between highest and lowest OPP for 2D/3D benchmark testing.
- Although user-space daemon process gives slightly lesser battery life, it switches between all the five OPPs, effectively utilizing all the frequencies provided by the processor.

4.2 Recommendations

- Prefer Ondemand for running high processor intensive applications efficiently with no compromise in performance such as gaming, animation, dynamic flash page rendering etc.
- Prefer Ondemand and user-space daemon process for running regular low processor intensive applications such as music, low resolution streaming video, downloading, decompression etc.

5. Conclusion

Android is considered to be the future mobile operating system. It offers end-to-end software framework with middleware and libraries along with some of the common telephony applications for the smartphone developer. The proposed work exploits the features of Linux and Android focusing on its power management capabilities. Limitations of Android related to the power management were identified and arrived at design specifications to overcome the limitations, as per the recommendations given by Consumer Electronics Linux Forum. To achieve the objectives, a thorough literature study of Android stack, ARM architecture and various low power techniques possible for Linux and Android was carried out. To prepare development setup for implementation and testing of low power techniques Linux and Android were ported to OMAP3530 based target board. Necessary device drivers for clock and power modules of OMAP3530 were ported from OMAP-PM branch to android-omap kernel. Linux active and static power management features were implemented and tested for their functionality. A daemon process was designed and implemented as part of Linux active power management. Various test cases were carried out for active power management and arrived at results in terms of current consumption, performance, total execution time and battery life. From test results it was concluded that active power management gives up to 29% increase in battery life while static power management gives up to 117% increase in device standby time.

The proposed work gives Linux power management solution, which is applicable across any mobile operating system based on Linux and not just Android. The work offers further opportunities for the delegate and the research community to continue the work and come up with inventive and better solutions for saving power of mobile devices.

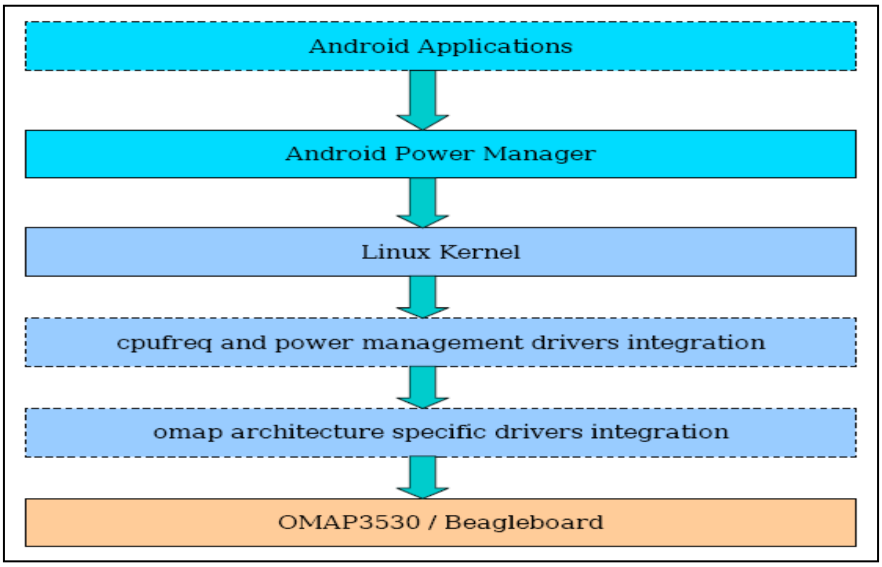

Fig. 1. System block diagram.

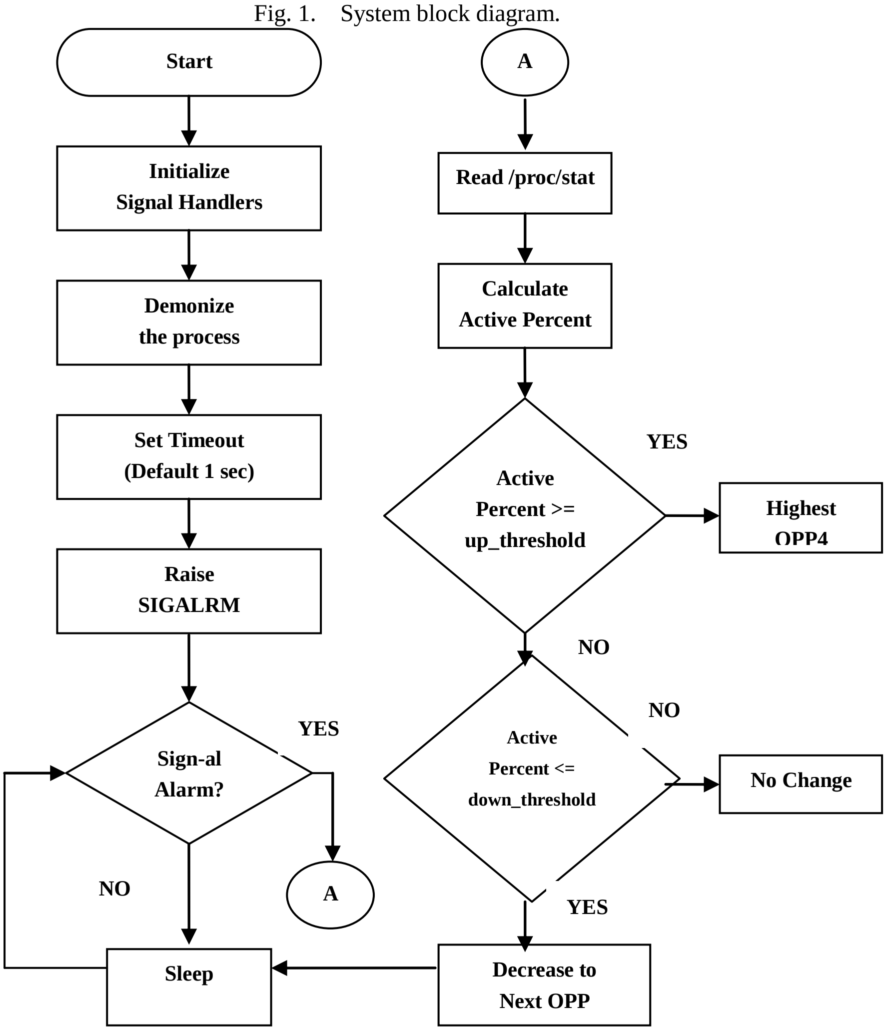

Fig. 2. Flowchart for daemon process.

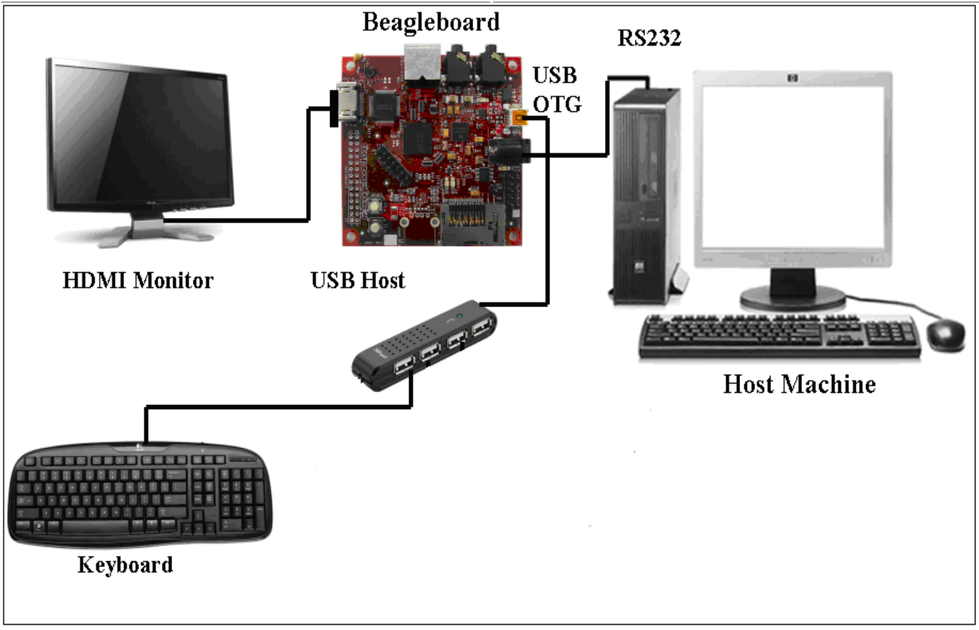

Fig. 3. Test setup.

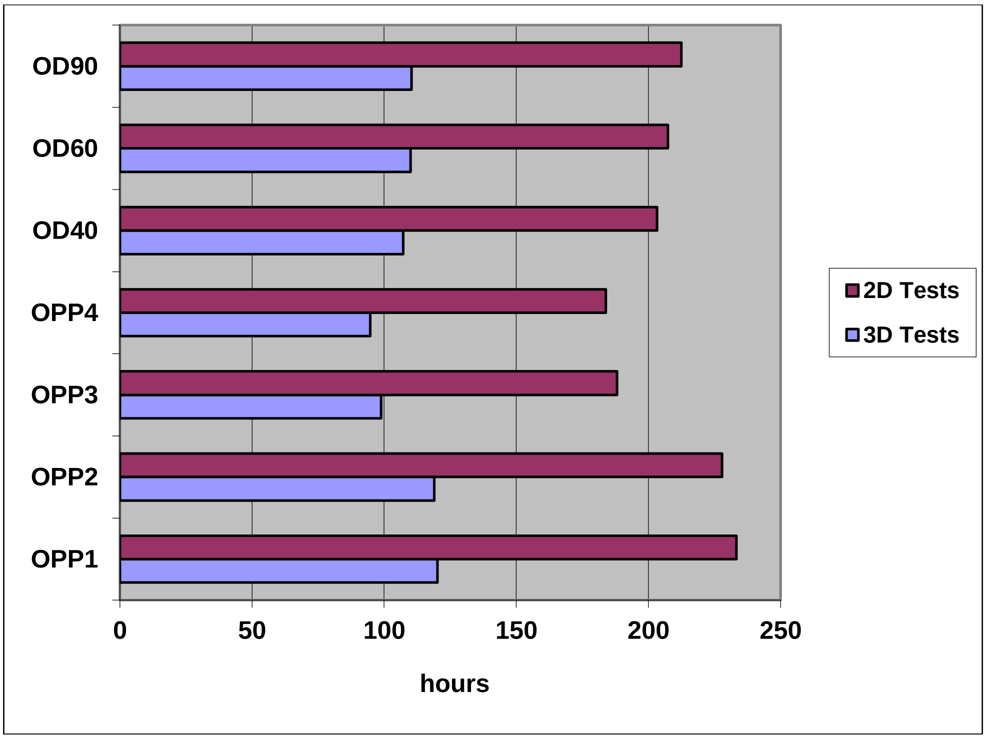

Fig. 4. Battery life of 2D/3D test results.

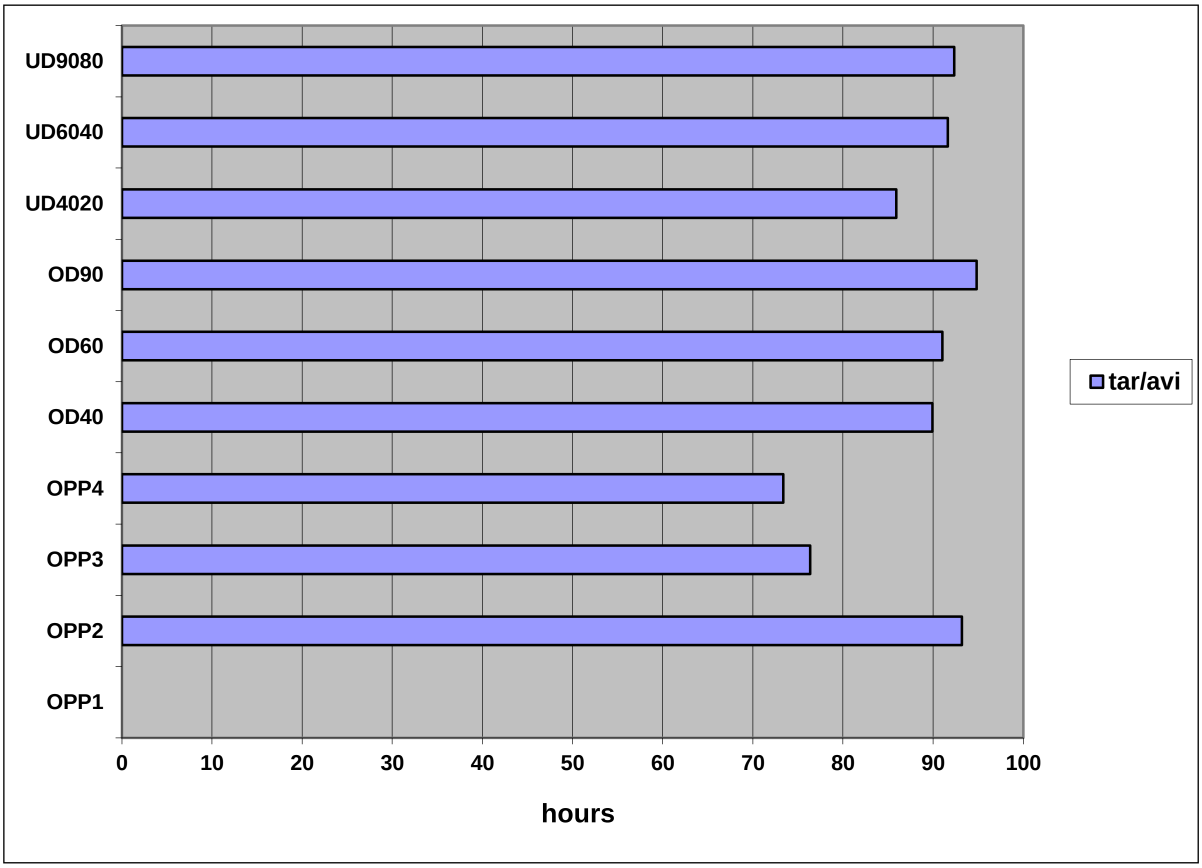

Fig. 5. Battery life of decompression and video test results.

Current (mA)	PM	Dynamic Tick	OPP4	CPUidle	Low Power Mode			Standby (hours)
					sleep_while_idle	voltages_off_while_idle	enable_off_mode	
300								3.33
300								3.33
279								3.58
231								4.33
231								4.33
161								6.21
161								6.21
138								7.24

Fig. 6. Static power management test results.

References

(1) Kihwan Choi, Ramakrishna Soma, Massoud Pedram (2004), "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times" , In Proceedings of The Conference on Design, Automation and Test in Europe, Vol. 1, pp. 10016.

(2) Jeff Sharkey (2009), "Coding for Life—Battery Life, That Is",

http://code.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html

(3) Venkatesh Pallipadi and Alexey Starikovskiy (2006), "The ondemand Governor: Past, Present and Future", In Proceedings of Linux Symposium, Vol.2, pp.223-238.

(4) OMAP35x Applications Processor Technical Reference Manual, Literature Number SPRUF98D, Texas Instruments, http://www.ti.com/lit/gpn/omap3525.

(5) www.beagleboard.org/uploads/BBSRM_B5.pdf.

(6) http://www.netmite.com/android/mydroid/development/pdk/docs/system_requirements.html.

(7) http://www.netmite.com/android/mydroid/development/pdk/docs/power_management.html.

(8) Mauro Marinoni, Giorgio Buttazzo (2006), "Balancing Energy vs. Performance in Processors with Discrete Voltage/Frequency Modes, In Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 294-304.

(9) Joo-Young Hwang, Sang-Bum Suh (2008), Woo-Bok Yi, Jun-Hee Kim, Ji-Hong Kim, Low Power MPEG4 Player, In the Proceedings of Linux Symposium Linux Symposium, Vol.1, pp. 219-228.

(10) Linux_2.6.32_source/ Documentation/cpu-freq/governors.txt.

(11) http://tree.celinuxforum.org/CelfPubWiki/CELF_Specification_V_1_0_R2?action=print.

(12) Welcome to Android, http://source.android.com/download.

(13) OMAP Power Management, http://elinux.org/OMAP_Power_Management.

Thimmarayaswamy Krishnappa (Non-member) received B.E in Computer Science and Engineering from Sri Siddhartha Institute of Technology, Tumkur in 2006. Currently, he is pursuing his M.Tech in Computer Science and Engineering from B.M.S. College of Engineering, Bangalore.

Mary D'Souza (Non-member) received B.E in Computer Science and Engineering from Sri Siddhartha Institute of Technology, Tumkur in 2006. Currently, she is pursuing her M.Tech in Computer Science and Engineering from B.M.S. College of Engineering, Bangalore.

Golla Varaprasad (Non-member) received B.Tech in Computer Science and Engineering from Sri Venkateswara University, Tirupati in 1999 and M.Tech in Computer Science and Engineering from B.M.S. College of Engineering, Bangalore in 2001 and PhD in Computer Networks from the Anna University, Chennai in 2004 and worked as a Postdoctoral Fellow at Indian Institute of Science, Bangalore in 2005. Currently, he is working as an Assistant Professor in B.M.S. College of Engineering, Bangalore. His areas of interests are MANET, mobile communications and SNMP.