A Project Report

On

**Energy Efficient Video Player for Android**

Submitted towards partial fulfilment of the requirement for the award of degree of

**BACHELOR OF ENGINEERING**

In

**COMPUTER SCIENCE & ENGINEERING**

2018-2019

By

**RIZWAN ULLA KHAN**

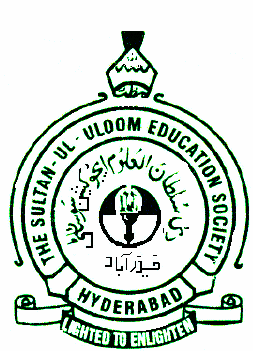
**(1604-15-733-056)**

**ABDUL GAFFAR**

**(1604-15-733-115)**

**SAJEED HUSSAIN**

**(1604-15-733-312)**

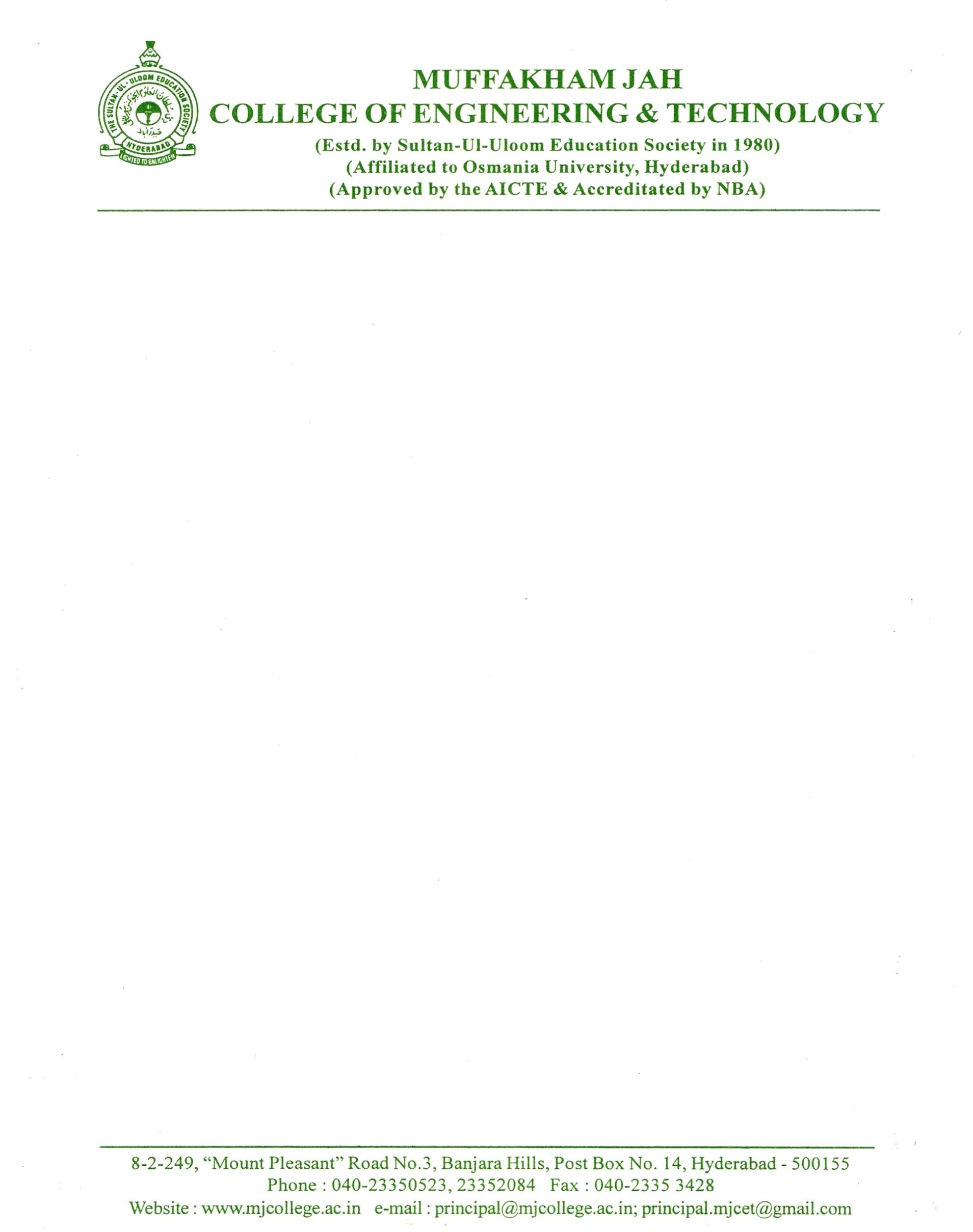


**Department of Computer Science and Engineering**

**Muffakham Jah College of Engineering. And Technology**

**(Affiliated to Osmania University)**

**Hyderabad-500 034**

****

**CERTIFICATE**

This is to certify that the project work entitled “**Energy Efficient Video Player For Android”** is a bonafide work carried out by **Mr. RIZWAN ULLA KHAN(1604-15-733-056), Mr. ABDUL GAFFAR (1604-15-733-115) and Mr. SAJEED HUSSAIN (1604-15-733-312)**in partial fulfilment of the requirements for the award of degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** by **OSMANIA UNIVERSITY**, Hyderabad, under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

Project Supervisor Head of the Department

**Mrs. GOURI PAIL Dr. A. A. MOIZ QYSER**

Assistant Professor Professor and Head

Department of CSE Department of CSE

MJCET, Hyderabad. MJCET, Hyderabad.

**DECLARATION**

This is to certify that the work reported in the present project entitled entitled **“Energy Efficient Video Player For Android”** is a record of work done by us in the Department of Computer Science and Engineering, Muffakham Jah College of Engineering &Technology, Osmania University. The reports are based on the project work done entirely by us and not copied from any other source.

**Mr. Rizwan Ulla Khan**

**Mr. Abdul Gaffar**

**Mr. Sajeed hussain**

**ACKNOWLEDGEMENTS**

I would like to express my sincere gratitude and indebtedness to my project supervisor Mrs. K. Sridevi for her valuable suggestions and interest throughout the course of this project.

I am also thankful to Head of the department **Dr. A. A. Moiz Qyser** for providing excellent infrastructure and a nice atmosphere for completing this project successfully.

I convey my heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed.

Finally, I would like to take this opportunity to thank my family for their support through the work. I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

**Mr. Rizwan Ulla Khan**

**Mr. Abdul Gaffar**

**Mr. Sajeed hussain**

# Abstract

Media player support all modern electronic devices nowadays. However, power management is one of the most critical issues in the design of today’s Media player. The goal of power management is to maximize performance within a given power budget The main of this project is to reduce the electrical energy costs of operating data centers of media player. A common way of cost reduction is to perform dynamic voltage and frequency scaling (DVFS), thereby matching the CPU’s performance and power level to incoming workloads. Another power saving technique is CPU consolidation, which uses the minimum number of CPUs necessary to meet the service request demands and turns off the remaining unused CPUs. DVFS has been already extensively studied and verified its effectiveness. On the other hand, it is necessary to study more about effectiveness of CPU consolidation. Key questions that must be answered are how effectively the CPU consolidation improves the energy efficiency and how to maximize the improvement. These questions are addressed in this paper. After understanding modern power management techniques and developing an appropriate power model, this paper provides an extensive set of hardware-based experimental results and makes suggestions about how to maximize energy efficiency improvement through CPU consolidation. In addition, CPU consolidation algorithms, which reduce the energy delay product up to 13% compared to the Linux default DVFS algorithm

However, power management is one of the most critical issues in the design of today’s Media player. The goal of power management is to maximize performance within a given power budget. Power management techniques must balance between the demanding needs for higher performance/throughput and the impact of aggressive power consumption and negative thermal effects. Many techniques have been proposed in this area, and some of them have been implemented such as the well-known DVFS technique which is used in nearly all modern Media player. This paper explores the concepts of multi-core, trending research areas in the field of Media player and then concentrates on power management issues in multi-core architectures. Moreover, it proposes a new technique for power management in Media player based on that survey.

**CONTENTS**

1. **Introduction 8**
   1. Introduction
   2. Problem Statement
   3. Implementation
2. **Literature Survey 13**
   1. Predicting the frequency for minimum energy consumption
      1. The critical Speed
      2. Relationship Between MAR and Critical Speed

2.1.3 Construction of The MAR-CSE Equation

* 1. MX Player Vs VLC Player—Features & Tool

2.3 Detailed Comparison of VLC and MX-Player

1. **System Analysis** **22**
   1. Problems with Existing System
   2. Proposed System 3.2.1 Technical Specification
   3. Software Requirement Specification.

3.3.1 Hardware requirements

3.3.2 Software requirements

1. **System Design** **25**

* 1. System Architecture
  2. Android power management
  3. Linux CPU Frequency Technique
  4. Kernel Space Governor
  5. User space Governor

1. **Implementation** **35**

1. **Testing 42**
2. **Screenshots/Outputs** 47

1. **Conclusion** 50

1. **Future Enhancements** 52

References

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **S NO** | **FIGURE ID** | **FIGURE NAME** | **PAGE NO.** |
| 1 | 2.1 | Energy consumption for different frequencies | 15 |
| 2 | 2.2 | Video players battery life | 19 |
| 3 | 4.1 | Implementation of the energy conservation | 28 |
| 4 | 4.2 | The android power management | 30 |
| 5 | 4.3 | Linux cpufreq technique | 31 |
| 6 | 4.3 | Curve of the MAR-CSE equation | 32 |
| `  7 | 4.4 | The flow chart of kernel space governor | 33 |
| 8 | 4.5 | The flow chart of user space governor | 34 |
| 9 | 4.5 | DVFS service | 35 |
| 10 | 8 | The result of proposed OI-based DVFS algorithm |  |

**LIST OF TABLE**

|  |  |  |  |
| --- | --- | --- | --- |
| **S NO** | **TABLE ID** | **TABLE NAME** | **PAGE NO.** |
| 1 | 2.1 | The frequency/voltage used | 16 |
| 2 | 2.1 | Critical speed and MAR for the benchmarks | 17 |
| 3 | 4.2 | The Flags for powermanger | 29 |

# 1.0 INTRODUCTION

**1.1 Introduction**

The latest Media player, such as in smart phones, and tablet PCs, with performance comparable to that of desktop PCs can now serve consumers who want to operate highly complex applications. Many mobile companies have now released recent smart phones with state-of-the-art applications processors (APs), consisting of a multi-core, high-performance GPU, and many dedicated IPs. Usually, an embedded system has millions of design possibilities. However, the growth of technology regarding batteries has been relatively very low with regard to the growth of system performance. Thus, energy consumption by mobile systems is a very hot issue both in terms of hardware and software, and many related technologies exist to address this, including a DVFS.

We focused on a DVFS algorithm for reducing energy consumption by the system, because the power save governor that apply to the lowest frequency does not save the maximum amount of energy. The main reason for these problems is that many resources are operated for longer times due to the lower frequency. To address this problem, a race-to-halt strategy that runs as fast as possible is the best way to save energy. However, it is not applicable in all situations. Even if a system is set to the highest frequency for finishing work, the memory is still operated at a slow speed in memory-bound applications that includes more memory operations than CPU operations because of the slower memory speed versus CPU speed. Thus, it is important to find an optimal frequency for conserving energy in the system. In efforts to conserve energy, considering memory, recent studies haveintroduced various approaches. However, these studies still have limitations. Jejurikar et al. presented a task slowdown algorithm for tasks that affect the leakage current of the processor. It has the ability to calculate a slowdown factor of tasks in a real-time OS using a simulation program. Liant et al. Proposed an AD-DVFS based on Memory Access Rate (MAR) that calculates memory activity using a cache miss parameter on the Performance Monitoring Unit (PMU). However, neither method considers state-of-the-art embedded system architectures, including a read-write-allocate cache or a hardware prefetch unit.

In this we introduce a new DVFS algorithm based on Operational Intensity (OI) for Media player and our experiment explains the relationship between OI and energy consumption. Additionally, we verified the performance of our proposed method using the FPMark benchmark program of the EEMBC consortium

The evolution of Media player led to the evolution of many research areas. Before the appearance of Media player, the speed of microprocessors increased exponentially over time. More speed requires more transistors. Moore observed that the number of transistors doubles approximately every two years. With the rapid increase in speed, the number of transistors in processors increased in a way that it can’t scale to Moore’s law anymore as an extremely huge number of transistors switching at very high frequencies means extremely high power consumption. Also, the need for parallelism increased and the instruction level parallelism was not sufficient to provide the demanding parallel applications. So the concept of multi-core was introduced by Olukotun et al to design more simple cores on a single chip rather than designing a huge complex one. Now all modern microprocessor designs are implemented in a multi-core fashion. Multi-core advantages can be summarized as follows:

Introducing Media player aroused many related areas of research. Dividing code into threads, each can run independently is very important to make use of the power of the multi-core approach. However, not all code can be divided in such a manner. That issue was described by Amdhal in which concludes that maximum speedup is limited by the serial part and that is called the serial bottleneck. Serialized code reduces performance expected by the processor; it also wastes lots of energy. Also, the parallel portion of the code is not completely parallel because of many reasons such as synchronization overhead, load imbalance and resources contention among cores. The serial bottleneck research led to the evolution of asymmetric Media player.

The concept of asymmetric Media player implies that the design would include one large core and many small cores. The serial part of the code will be accelerated by moving it to the large core and the parallel part is executed on the small cores. This accelerates both the serial part by using the large core and the parallel part as it will be executed simultaneously on the small cores and the large core to achieve high throughput. Using asymmetric cores can be more energy efficient too. In Mark et al. described how asymmetry can be achieved. They divided it into static and dynamic methods. For static methods, cores may be designed at different frequencies or a more complex core with completely different micro-architecture may be designed. In dynamic methods, frequencies can be boosted dynamically on demand or small cores may be combined to form a dynamic large core and this is described in detail in. Other research topics related to Media player that emerged include the following: power management, memory hierarchies in Media player, the design of interconnection networks in Media player, heterogeneous computing in Media player, reliability issues in Media player and parallel programming techniques. In power management, the main objective is to reach the maximum performance of the processors without exceeding a given total power budget for the chip. There has been lots of research on power management in chip multiprocessors. Here we are going to discuss most of those techniques and some modern works that try to optimize the efficiency of these techniques. In this paper we examine all popular techniques in detail and how they work to minimize performance losses while saving power. We

investigate the suitable technique for each case (workloads, power budget available, critical systems) and how to make these techniques even more suitable for their cases.

# 1.2 Problem statement

Power management has become a major issue in the design of multi-core chips. There are many negative effects that result from increasing power consumption such as unstable thermal properties of the die and hence affecting the system performance which makes power consumption issue sometimes more important than speed. An important observation is that threads running on different cores do not need the same power all time to execute at high performance. There are some waiting times due to memory read/write operations for example which require saving unnecessary processing power. So, to achieve a good balance between scalar performance/throughput performance and power it is essentially required to dynamically vary the amount of power used for processing according to temporal analysis of the code needs.

Developed power management techniques can be classified into two main categories: reactive and predictive. In reactive techniques, the technique reacts to performance changes in the workload. In other words, a workload may initially have states that need high performance, others of I/O waits and low performance. When the state of the workload changes, the technique reacts to that change accordingly. However, there might be some lag between workload phase changes and power adaptation changes which may lead to states of either in-efficient energy consumption or performance degradation. On the other hand, predictive techniques, for example overcome this issue. Those techniques predict phase changes in the workload before they happen, and hence act immediately before a program phase changes. That leads to optimal energy-saving and performance results. However, there is no workload that can be fully predicted, so reactive techniques are used for portions that cannot be predicted (which is usually more than 60% of the entire workload). So, reactive techniques are inevitable to use and consequently we concentrate in this study on those techniques. Here, we are examining some of the dynamic techniques as shown in to achieve the best level of power management inMedia player. We also discuss some issues related to each of these techniques and how previous research attempted to handle these issues.

# 1.3 Implementation

# MODULES

**Module-1**

In this module a welcome screen is first shown with the name of the application and an option to continue then a user interface is provided to for the user to interact and select the required video from the lit of video files he wishes to play. The list of video files is displayed in a plain, simple and an uncomplicated manner. This makes the whole experience for the user very comfortable and it can even be easily understood by kids.

The list of files is shown from the local storage of the device. Once the user selects the required file he wishes to play, the work of this module is done.

# Module-2

In this module, application sends the video file selected towards the proposed DVFS where a number of steps are performed on the file where the least CPU frequency required by the device to play the file while maintaining the QoS(Quality of Service) is calculated. The QOS constraints are not compromised for energy efficiency.

# Module-3

In this module, the file which was chosen by the user after having gone through the DVFS steps where the minimum CPU Frequency is calculated is placed into the buffer. A buffer is a temporary storage area, usually in RAM. The purpose of most buffers is to act as a holding area, enabling the CPU to manipulate data before transferring it to a device.

The processes of reading and writing data to a disk are relatively slow; many programs keep track of data changes in a buffer and then copy the buffer to a disk. This helps in maintaining efficiency.

# Module-4

In this module, finally the file placed in the buffer is fetched by the application and played.

The video is played till the user wants it to be played. The user is provided with three options; play, pause and stop. User can use any of these three given options to perform actions on the selected file. These options provide user with flexibility.

# 2.0 LITERATURE SURVEY

2.1PREDICTING THE FREQUENCY FOR MINIMUM ENERGY CONSUMPTION

The existence of the critical speed is first introduced. The relationship of the memory access rate and the critical speed is then discussed. After that, the core method of the proposed energy conservation algorithm – prediction of the critical speed by a prediction equation based on the relationship – is then explained.

### The Critical Speed

In real applications, the CPU usually needs to stall and wait for memory accesses. Consequently, the memory operations also influence the total energy consumption. From previous studies, it has been found that reducing the frequency may not always induce lower energy consumption. We have observed that the lowest energy consumption usually appears at some operating speed other than the slowest clock rate. The frequency at which the lowest energy can be obtained is defined as the critical speed for the executed code on the running machine.

how the energy consumption changes for two benchmark programs when different frequencies were used. The data were collected on our target platform through real measurements. The measured values only counted the energy consumed by the CPU and the memory. In the figure, we can see that the minimum energy consumption appears at a frequency higher than the lowest frequencies for both programs.

This phenomenon is caused by the reason that as the CPU frequency is decreased, the total execution time is lengthened. Extra energy consumption will be introduced by the memory subsystem during the extended execution time, because the memory subsystem needs to be kept in active mode while the processor is working. The total energy consumption may thus be increased inversely.

From the measured data, an approximation curve equation describing the relationship between the energy consumption and the frequency can be created by regression analyses. From the equation, the frequency which can induce the minimum energy value, i.e. the critical speed, can be obtained. The approximation equations computed from the measured data for fft\_b and jpeg\_b and the corresponding curves. The local minimums of the curves indicate their critical speeds. As described in the first section, our goal is to find the critical speed during the execution time for the maximum energy conservation.

### Relationship between MAR and Critical Speed

Since energy consumption is affected by the memory access behavior of the running program, we use the memory access rate (MAR) index as an indicator for the memory access property. It is defined as the ratio of the total number of data and instruction cache misses (*Ncache\_miss*) to the number of instructions executed (*Ninstr\_exec*). The formula of MAR is as follows.

MAR= N cache\_miss

Ninstr\_exec

The statistical numbers can be retrieved from the hardware counters such as the PMU counters provided by the XscalePXA270 processor that we used. From this definition, we can see that a program with a lower MAR value implies that it tends to be a CPU-bound program, and a higher value implies that it tends to be a memory-bound program.

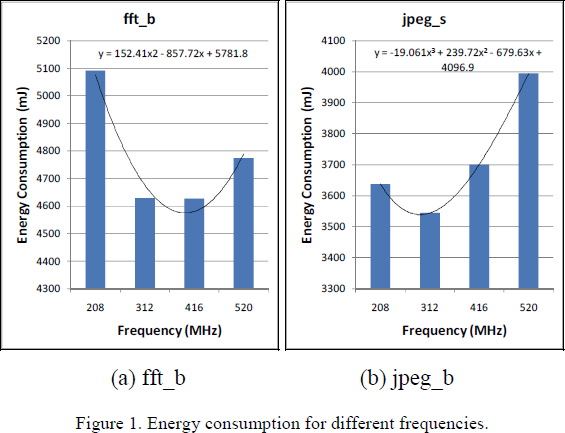
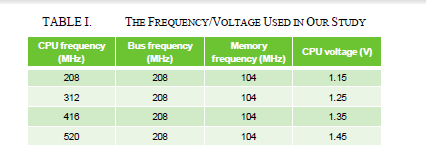
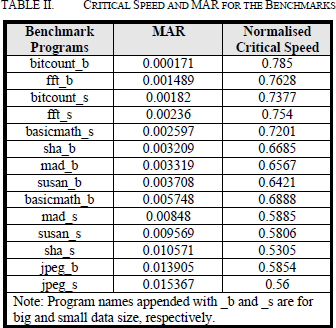


Figure 1. Energy consumption for different frequencies.

We have chosen the MiBench benchmark suite in our study. The selected benchmark programs include basicmath, bitcount, fft, sha, susan, jpeg, and mad. In considering that the size of the problem may affect how a program behaves, each benchmark program was tested

with two sizes of input data: large and small size. The critical speed and the MAR value were measured for each of these programs. The Xscale PXA270 development board was used for the measurements. The frequency and voltage combinations that we used on the platform are listed in Table I. The CPU frequency of 104MHz, which is actually supported by processor, was not considered in our study because the corresponding bus frequency is different from the one that is used for other CPU frequencies. Since our study does yet consider different bus frequencies, we have decided to exclude this configuration. Table II shows the measured results for all the benchmark programs. In the table, the critical speeds have been normalized with respect to the highest frequency, 520MHz. The table contents have been sorted according to the order of the MAR values, from the smallest to largest number. Note that the critical speeds are theoretical numbers calculated from the approximation curve equations of the measured data. The critical speed is typically not one of the discrete frequencies supported by the CPU. For example, the critical speed of fft\_b is 0.7628, which is about 396MHz and is not one of the supported frequencies listed in Table I. From the sorted result in Table II, it can be easily observed that there is a very interesting relationship between the MAR and the critical speed. That is, MAR is inversely proportional to the critical speed. As the value of MAR increases, the corresponding critical speed value goes down. That is, a program with a lower MAR (which tends to be CPU-bound) will have a higher critical speed, whereas a program with a higher MAR (which tends to be memory-bound) will have a lower critical speed.





This means that the energy consumption of a CPU-bound program may be raised as soon as the frequency is decreased.

For example, in Table II, fft\_b has a relatively low MAR value and is most likely a CPU- bound program. It has a higher normalised critical speed of 0.7628, which is about 396MHz. From Fig. 1(a), we can see that once the clock rate was set to a frequency lower than the critical speed, say 312MHz, its energy consumption started to increase. In contrast, a memory-bound program will be able to further reduce the energy consumption with an even lowerm frequency. For example, from Table II, we can see that the MAR of jpeg\_s has a relatively high value and is more like a memory-bound program. Its normalised critical speed is then a

lower value 0.56, which is about 291MHz. As a result, in Fig. 1(b), we can see that when the frequency was changed from 416MHz to 312MHz, which is still higher than its critical speed, the energy consumption was able to continue to be decreased.

### Construction of the MAR-CSE Equation

Since the critical speed, which by definition consumes the least energy, is usually higher than the lowest frequency provided by a processor, it can be used as the target frequency when the user wants to minimize the energy consumption. The major advantage of the critical speed is that not only can the energy saving be maximized, but also a better performance can be achieved. As a result, it can be used to replace the traditional method, which typically chooses to use the lowest frequency, for the purpose of maximum energy saving. Our goal is to predict and use the critical speed during the task execution time. Based on the inversely proportional relationship between the MAR and the critical speed, a regression equation can be created from the measured data in Table II by the least square curve fitting method. This equatiois called the MAR-based Critical Speed Equation (MAR-CSE). When a program's run-time MAR information can be obtained, the MAR-CSE equation can be used to predict the critical speed. The MAR-CSE equation and the approximation curve for the measured data are both illustrated in Fig. 2. To get the runtime MAR information, some counter values

must be retrieved from the PMU at task execution time. The MAR value can then be mapped to the corresponding critical speed through the MAR-CSE equation. At the critical speed, we may then have the programs running with minimum energy consumption but at a better performance level. In the following section, details of the implementation are described.

**2.2 MX Player Vs VLC Player—Features & Tool**

* + - The first look of VLC is better than MX Player, because it shows the whole screen with other options, but MX Player shows the library directly.
    - In case of theme, MX Player has many options to choose from. But VLC has only Black/Orange theme.
    - For alignment, VLC allows changing the alignment for both title and subtitles, but MX player allows changing the alignment of subtitles only.
    - VLC provided the option Android TV by which users can connect their device with TV via Bluetooth and enjoy watching movies, but MX Player doesn’t have the option.

## Media View:

* + - MX Player is better in adjusting screens with various options like fit-to-screen, stretch, crop, 100% or 150%, but VLC allows adjusting to center, vertical, horizontal, 16:9 & 4:3 ratios.
    - Start over or resume option are available on both the players.
    - Libraries can be listed by sort option based on titles, size, date, length, etc on both the players do not make any difference in this case.
    - Gesture controls for volume, brightness & forward-backward are also same.
    - Users can use both the player for playing videos as well as use other apps too on the same time.
    - VLC Player has the better audio player with equalizer to adjust frequencies, but MX Player a few in-built stereo modes only.

## Hardware & Software Decoder:

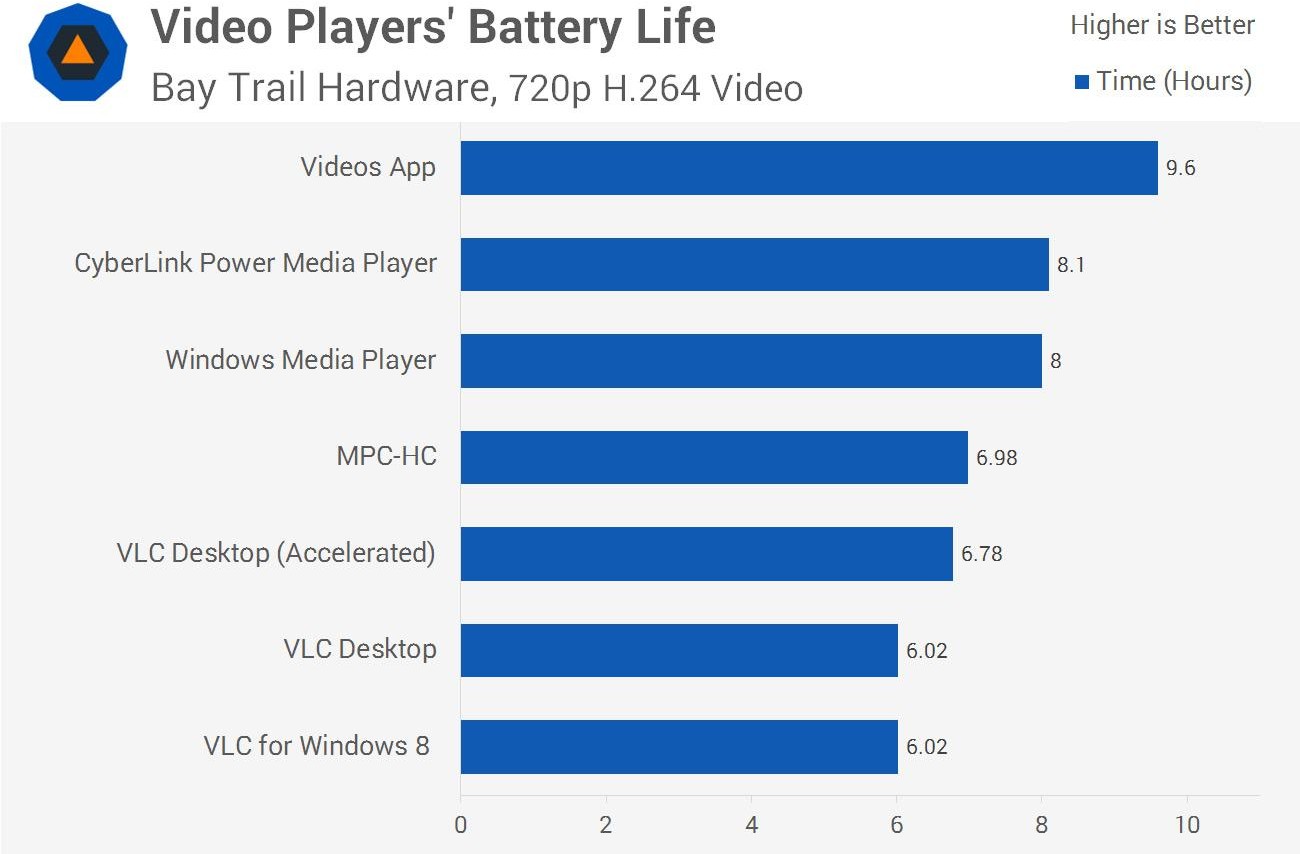
Sometimes it is better to have hardware decoder in your video player to watch videos like online videos of 1080p etc, because software decoder uses the cpu and often gets slow. In case of SW & HW decoder MX Player is better, because it has both. But VLC doesn’t have both the decoder for all android versions.

## Extra Features:

* + - There is lock feature on both VLC & MX player which helps the users to watch movies uninterruptedly.
    - VLC provides an option called Sleep Timer on the screen, by which users can set a time for a period of time and after the period, the media file will automatically stopped. But for the same purpose in MX player, users have to go to the menu.
    - VLC has provided an interesting feature JUMP TO TIME using which users can jump to the particular timing of a movie/video, but MX Player doesn’t have such feature.
    - VLC allows adjustments of subtitle and delayed audio, but MX player doesn’t have such feature.
    - MX Player has a feature for playing audio or video contents in background, but in VLC you have to go to audio player option.
    - In case of Subtitles, MX Player & VLC both have the same features of adjusting font color, size, alignment, language change, etc

**MX Player** offers just the right mix of ease of use and powerful features. An uncluttered viewer supports various swipe gestures and pinch to zoom, while the app features both software and hardware decoding for a broad variety of file formats. Subtitle controls, variable aspect ratio and a screen lock are all within reach of the main view screen, while other tweaks are hidden in the settings menu. Unfortunately, due to licensing issues, support for DTS and AC3 audio isn't available in the core app, which can be a deal breaker, though there are custom codecs available online.

**VLC for Android,** all-in-one media player that'll play just about any file format that you can throw at it, then give VLC for Android a spin. Out of the box it can play almost anything from the ubiquitous MP3 and MP4 files to more unusual formats such as MKV and FLAC. In addition to playing local files, VLC for Android also supports network streaming and media library organization, as well as advanced playback features such as multiple audio tracks and subtitles. But VLC for android somes get stutering when playing H.265 codec video.



**2.3 Detailed Comparison of VLC and MX-Player**

**The User Interface:**

Users reviews say that the first appearance of MX player is not so impressive and do not even contain a well designed menu area that is popular in case of VLC. But one must say that MX is perfect choice if you want to stay focused with video content delivery. Other than annoying ads, you will not find any other distraction on its interface.

In case of VLC, you can definitely enjoy finest screen controls with well managed search menu and it also offers refresh options to stay tuned with updated library. Users can easily switch between audio and video files over VLC.

**Viewing Media:**

It is really difficult to prove that any one of these players perform better in terms of viewing media category. Both of these offer great content delivery but you can put VLC on higher side just because of its ability to play videos without cluttering screen with ads. MX keeps on distracting users with so many ads that can even cover your desired video content on screen. You need to make payment to get pro version of MX that is free from ads but VLC is always free with loads of quality features.

**Extra Features:**

Developers often keep on improving their media player software with advanced features so that viewers can enjoy incredible experience. But in case of VLC you will find least modifications and it appears same from so many years. This tool is just fine for basic users if you don’t expect additional stuff.

On the other side, MX player can serve your video watching needs in better manner with added features. Its lock screen feature can keep kids out of control and gesture control feature enables users to enjoy easy controls over brightness and volume.

One more interesting thing to note about MX player is its advanced hardware integration. However, both these systems offer satisfactory services but still the constant updates on platform features make things more useful.

**Support:**

You will be able to avail great support service from developers online; it is possible to connect via chat with real support assistant and you can avail fast solutions for all troubles. But you may have to wait too long to get connected with support team as MX player forum stays too busy.

There is no doubt to say that VLC is much better option in terms of support service. You can stay connected to information via forum, Wiki, well designed troubleshooting guide and FAQs. The dedicated service team of VLC can provide you instant solution for all troubles.

# 3.0 SYSTEM ANALYSIS

**3.1 Problems with Existing System**

Power management has become a major issue in the design of multi-core chips. There are many negative effects that result from increasing power consumption such as unstable thermal properties of the die and hence affecting the system performance which makes power consumption issue sometimes more important than speed. An important observation is that threads running on different cores do not need the same power all time to execute at high performance. There are some waiting times due to memory read/write operations for example which require saving unnecessary processing power. So, to achieve a good balance between scalar performance/throughput performance and power it is essentially required to dynamically vary the amount of power used for processing according to temporal analysis of the code needs.

Developed power management techniques can be classified into two main categories: reactive and predictive. In reactive techniques, the technique reacts to performance changes in the workload. In other words, a workload may initially have states that need high performance, others of I/O waits and low performance. When the state of the workload changes, the technique reacts to that change accordingly. However, there might be some lag between workload phase changes and power adaptation changes which may lead to states of either in-efficient energy consumption or performance degradation. On the other hand, predictive techniques, for example overcome this issue. Those techniques predict phase changes in the workload before they happen, and hence act immediately before a program phase changes. That leads to optimal energy-saving and performance results. However, there is no workload that can be fully predicted, so reactive techniques are used for portions that cannot be predicted (which is usually more than 60% of the entire workload). So, reactive techniques are inevitable to use and consequently we concentrate in this study on those techniques. Here, we are examining some of the dynamic techniques as shown in to achieve the best level of power management inMedia player. We also discuss some issues related to each of these techniques and how previous research attempted to handle these issues.

**3.2 Proposed System**

The dynamic power consumed due to the switching of gates contributes greatly to the total power dissipated in CMOS circuits. The dynamic power consumption can be stated by the following equation.

P dyn =Nsw Cl V2dd f

In the equation, *Nsw* is the switching activity, *CL* is the load capacitance, *Vdd* is the supply voltage, and *f* is the operating frequency. From (1), we can see that the power consumption is proportional to the product of the frequency and the square of the supply voltage. As a result, decreasing the voltage has a quadratic effect on the reduction of the power consumption. Many previous works on DVFS have considered real-time systems, in which the job execution time can be extended to approach the deadline so that the frequency can be scaled down.

This in turn saves the energy consumption. However, most of the handheld devices in use today are not real-time systems. In addition, the source of energy consumption not only includes the processor, but also some other parts such as the main memory, the storage and the peripheral devices.

**3.2.1 Technical Specification**

DVFS technique called workload decomposition, in which the CPU workload is decomposed in two parts: on-chip and off-chip. This is based on some run-time statistics reported by the hardware counters. The on-chip workload means the execution cycles of the instructions in CPU operations, and the off-chip part represents the cycles for external memory accesses.

defined the memory access rate (MAR) according to the information provided by the hardware counters to evaluate the effect of external memory accesses. They proposed an online algorithm to achieve maximum energy saving by selecting the optimum frequency- voltage combination based on the system workload. indicated that the minimum energy consumption of a system may not appear at the slowest operating speed, and defined the critical speed of a task as the one which can assure the minimum energy consumption. We have also observed a similar phenomenon. we have further observed that there is an interesting relationship between the memory access rate and the critical speed. In this paper, a DVFS energy-conservation algorithm for maximum energy saving is proposed. It predicts the critical speed based on this relationship. A correlation equation, called the MAR – Critical Speed Equation (MAR-CSE), is first conducted to describe the relationship for the target platform, and is then used in the algorithm for the critical speed prediction.

Our algorithm has been implemented in the Android operating system as a power manager. Android is an open source operating system. It has been used extensively in smart phones and some other systems since it was released in 2008. It is built on top of the Linux kernel and contains a novel user level software stack for mobile devices. The software stack includes three layers: the Application layer in which the Android applications exist, the Application Framework layer

which comprises components supported by Android for application developments, and the Libraries layer which contains native processes and shared libraries. Android provides a simple power management framework.

It provides application developers with a set of power management interfaces through the Power Manager class. The Android power manager basically utilizes the DPM techniques. when an application needs to use the CPU or the back light for a period of time, it has to allocate a Wake Lock and acquire the lock so as to keep them powered on. If no application needs to hold the resource anymore, the CPU will enter a sleep mode and the back light will be turned off once the Android operating system thinks it can do so. While Android has provided an aggressive dynamic power management scheme, it is mainly used to reduce the energy consumption when some of the system components are idle or not being used. The Android operating system basically relies on the underlying Linux kernel support for dynamic voltage and frequency management. However, the default method, called the On demand governor implementation, was not designed for the case that we address, in which the user wants to minimize energy consumption when tasks are running.

The implementation of our energy conservation algorithm involves three parts, corresponding to the Android Application layer, the Android Libraries layer, and the Linux kernel-level driver. The MAR-CSE prediction equation is implemented in the power manager, which runs as an Android service to predict the critical speed which tends to minimize the energy consumption. During task execution time, the MAR information is retrieved and calculated from the hardware counters in the Performance Monitor Unit (PMU) of the Intel XScale PXA270 processor that we used in our experimental environment.

The power manager then uses the information as an input to the MAR-CSE equation to get the corresponding critical speed. Once the critical speed is obtained, it needs to be applied to the processor as the target frequency.

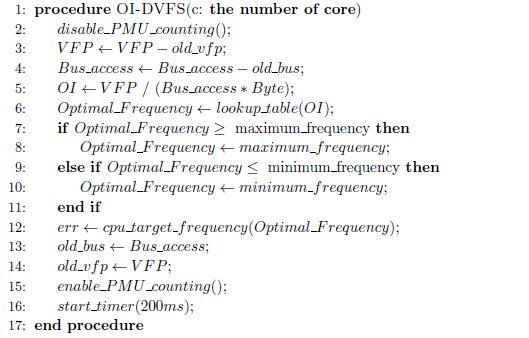
However, the predicted critical speed is usually not one of the discrete frequencies supported by the processor. We have used a method based on the dual-speed proposed in to find a pair of neighboring frequencies so as to approximate the critical speed.

In addition to retrieving task-related information, the hardware counters can also be used to estimate power consumption.

Contreras and Martonosi demonstrated a linear power estimation model for the processor by using the counters. The model provides an easy way to get. power consumption information without using external hardware equipments for the measurement. built a time model and an energy model, in which the hardware counters were also used to estimate performance degradation and energy consumption. Although these methods are convenient for power estimation without the need of extra equipments and complicated setup for the

measurement, to get more accurate data, however, we still chose to use standalone measurement hardware instead of the estimation methods

**Algorithm 1** The OI-based DVFS algorithm



**3.4SYSTEM REQUIREMENTS**

**3.4.1 HARDWARE REQUIREMENTS:**

* System : Pentium Dual Core.
* Hard Disk : 120 GB.
* Monitor : 15’’ LED
* Input Devices : Keyboard, Mouse
* Ram : 2GB.

**3.4.2 SOFTWARE REQUIREMENTS:**

* Operating system : Windows XP/7.
* Coding Language : Java
* Tool

**4.0 SYSTEM DESIGN**

**4. System Architecture**

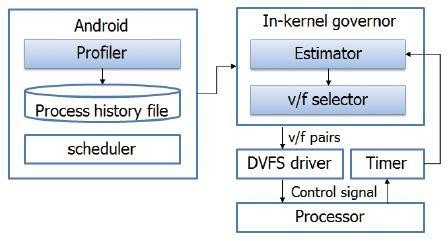
First, a *Profiler* traces a CPU usage of each process and makes a process history table. Then, *Estimator* brings schedule information from Android scheduler and estimate which process will be running in next time slice.

Finally, *v/f selector* selects appropriate voltage/frequency set for next time slice.

To estimate CPU usage of next time slice, the process information is profiled on the fly before v/f scaling. Android system basically provides the environment that is capable of extracting the kernel status information and canbe controlled through sysfs (file system) interface. We make the Process history table (PHT) by saving the MA (moving average) value of CPU usage of each process of every time slices. The reason of saving MA is to calculate average CPU utilization because CPU usage of each processes is varying with time varies. Profiler saves CPU usage value which is translated when processor runs at maximum frequency according to following equations.

*CUsaved*= *CUcurrent\_ freqmax freqcurrent*

In this equation, *CU* is CPU usage of each process



IMPLEMENTATION OF THE ENERGY CONSERVATION DVFSALGORITHM

**4.2 Android power management**

Android supports a simple and aggressive power management framework, called Power Manager. The Power Manager is just like DPM, it is designed with the rule that turns off the device resources such as keyboard, screen, and CPU if applications or services no longer need the resources any more.

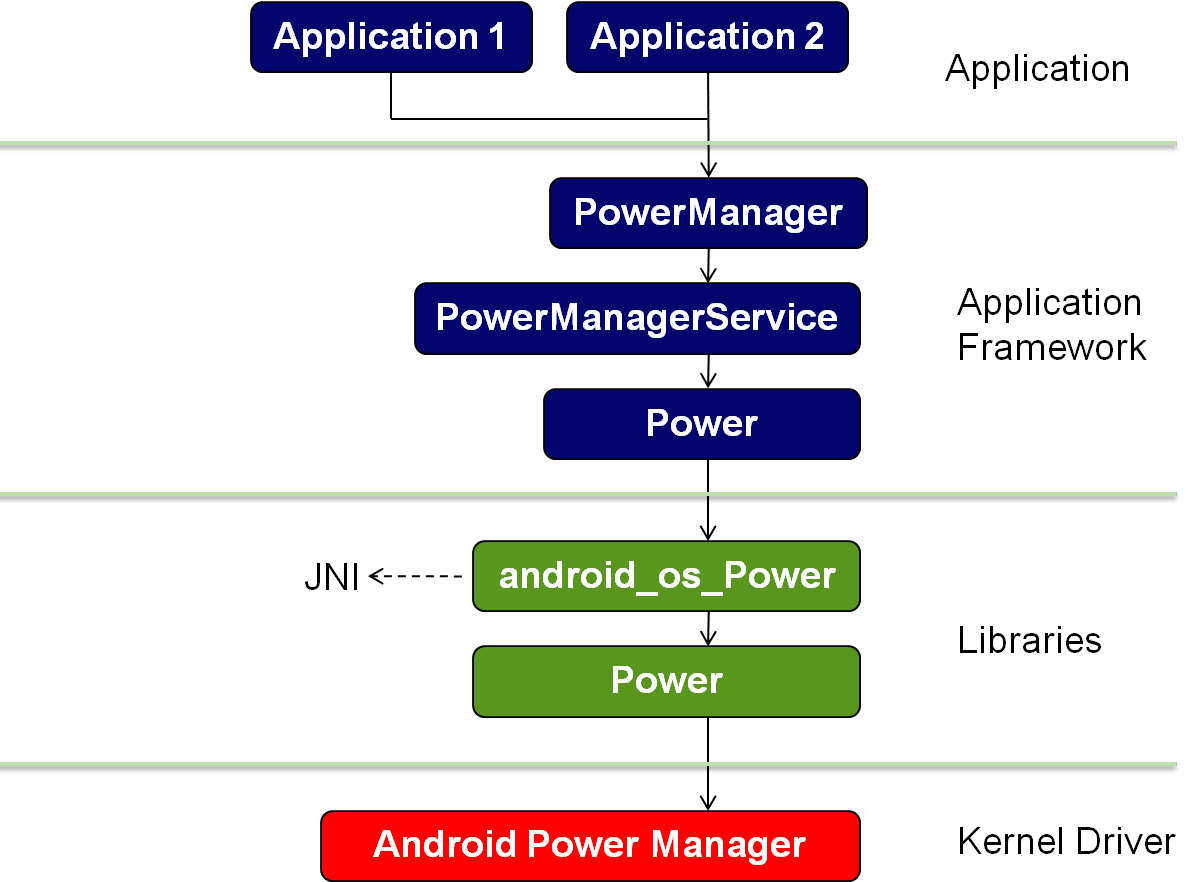
When an application needs to hold the resources, it has to declare a WakeLock and uses the WakeLock to acquire resources from Power Manager. On the other hand, the application has to release the WakeLock to ensure the resources can be turned off correctly when the resources are not needed anymore. the available WakeLock flags in Power Manager. Each flag relates to some of the power state of the device recourses.

a simple example for acquiring and releasing the WakeLock from PowerManager Freqlow and uses this until the end of the execution interval.

***Table 4-2. The flags for PowerManager.[23]***

|  |  |  |  |
| --- | --- | --- | --- |
| **Flag Value** | **CPU** | **Screen** | **Keyboard** |
| **PARTIAL\_WAKE\_LOCK** | On | Off | Off |
| **SCREEN\_DIM\_WAKE\_LOCK** | On | Dim | Off |
| **SCREEN\_BRIGHT\_WAKE\_LOCK** | On | Bright | Off |
| **FULL\_WAKE\_LOCK** | On | Bright | Bright |

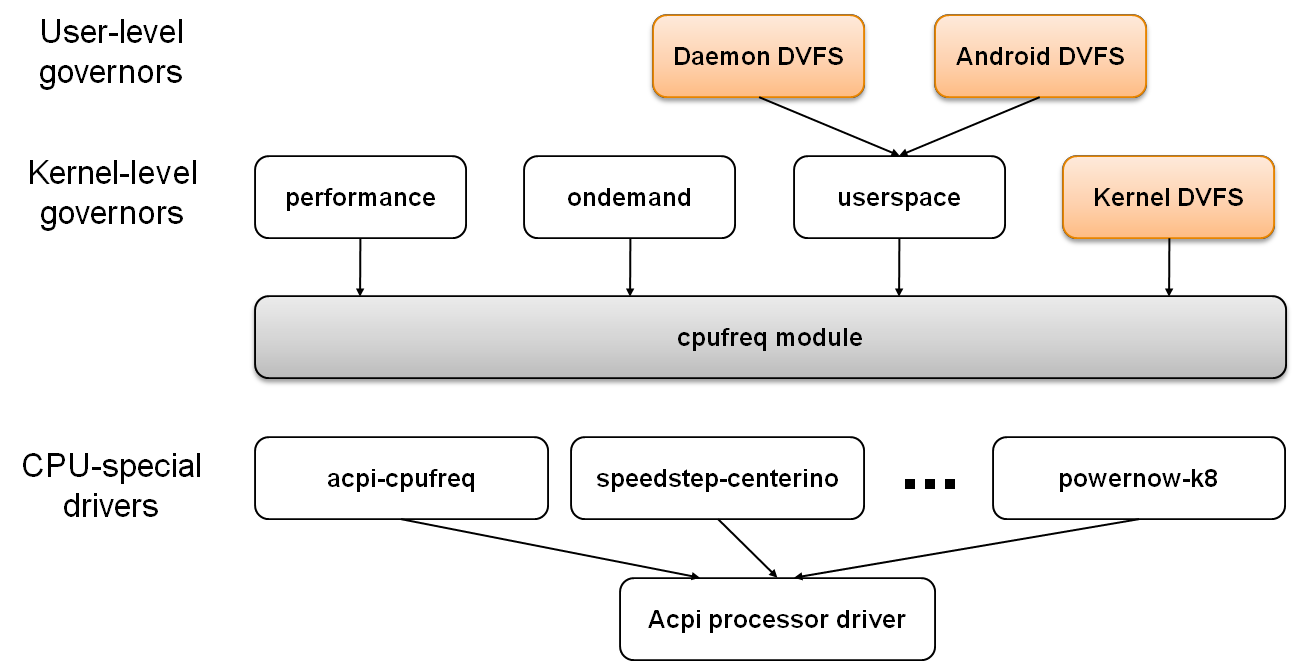
the architecture for Android power management. Applications acquire the resources from the PowerManager, and then PowerManager uses the methods in PowerManagerService. Accroding to the requests of PowerManager, PowerManagerService then call the native methods in the shared library. The shared library is written with the rule of Java native interface (JNI). It accesses *sysfs* provide by kernel driver to control the power management hardware.



**Figure 4-2. The Android power management**

**4.3 Linux cpufreq technique**

In the Linux kernel, cpufreq is a subsystem that provides a modularized interface to manage the CPU frequency. Through cpufreq, the governors can control frequency in a more convenient way. The following [Figure 4-2](#_bookmark0) depicts the cpufreq infrastructure

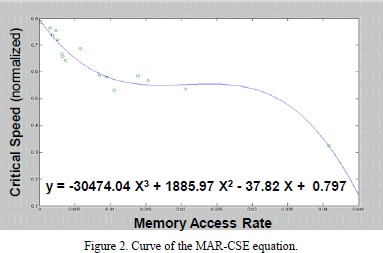


control the low-level frequency scaling hardware. It decouples the drive of the CPU-specific hardware from the policies. Examples of the CPU frequency adjustment technologies Intel○R SpeedStep○R Technology, AMD PowerNow!TM, and Intel Pentium○R 4 processor clock modulation, etc.

In kernel space, the kernel-level governors implement different policies for CPU frequency managements. For example, the performance governor always maintains the CPU frequency at the highest frequency; the ondemand governor manages the frequency according to the CPU utilization; and the userspace governor exports the available frequency information to the user space and allows the user-level governors to control the CPU frequency through the Linux sysfs interface.

We have implemented three versions of the proposed DVFS algorithm. One is

running in the Linux kernel space, which collects the MAR information in the Linux kernel directly. The other two are running in user space, which retrieve the information indirectly from the Linux kernel through the sysfs system interface.



In the Linux kernel under the Android operating system, the CPU freq subsystem provides a modularized interface to manage the CPU frequencies. The policy manager for power management is called a Governor in Linux, which controls the CPU frequency through the interface of CPU freq.

The CPU freq infrastructure, in which the CPU freq subsystem decouples the driver of the CPU-specific hardware from the management policies.

Several kernel-level governors have been supported by Linux for CPU frequency management. For example, the Performance governor maintains the CPU frequency at the highest frequency, the On demand governor manages the frequency according to the CPU utilization, and the Power resave governor sets the CPU frequency to the lowest clock rate.

Linux also provides the User space governor to export the available frequency information to the user space and allows the user-level governors to control the CPU frequency through the Linux system interface. Among the governors, the Power save governor is typically chosen when the user wants to minimize the energy consumption. We have implemented the proposed energy conservation DVFS mechanism as an Android service, which is a type of Android program, in the user space. It is called the AD-DVFS governor. The AD-DVFS governor is responsible for obtaining the current MAR value and predicting the critical speed according to the MAR-CSE equation. The execution flow of the AD-DVFS governor is shown in Fig. 4. Once the ADDVFS governor has been started, a Java thread is created to periodically perform the algorithm. According to our energy conservation algorithm, at the beginning of the execution flow, the AD-DVFS governor gets the statistical information from the PMU counters to compute the MAR value. Then, the critical speed is calculated from

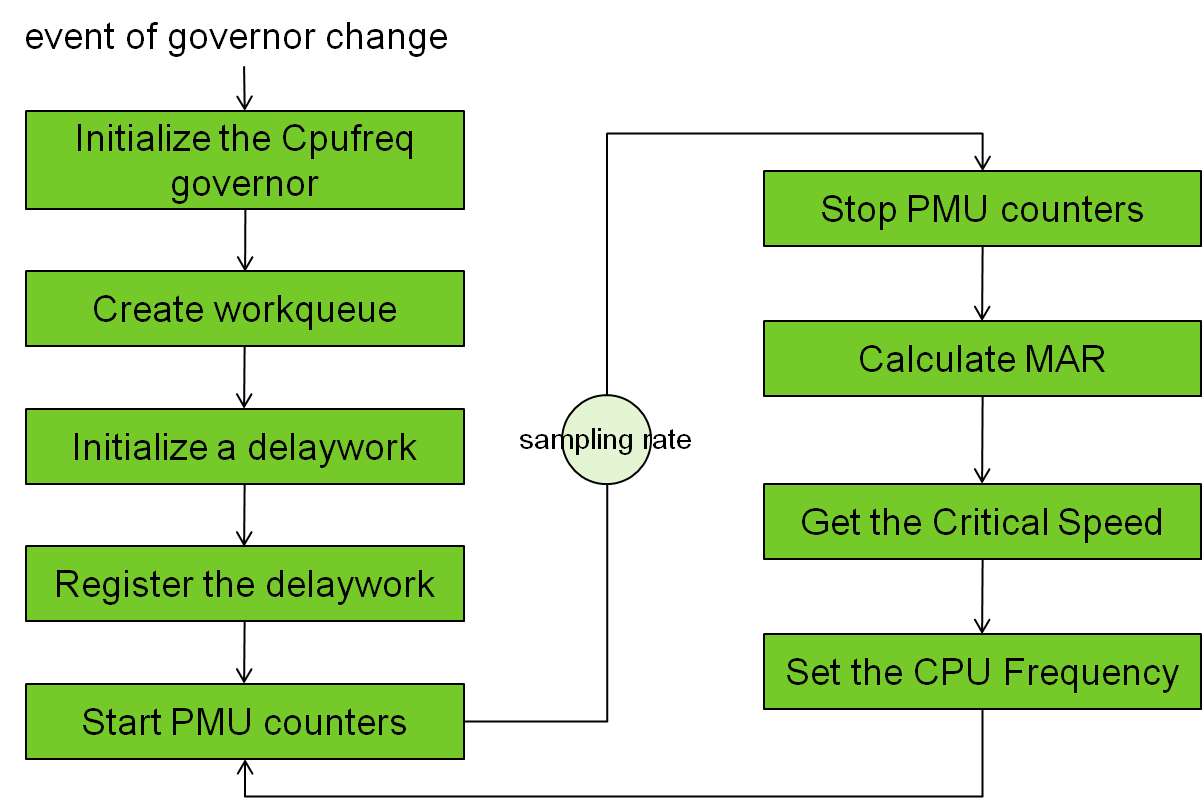
MAR-CSE. The frequency that will be applied to the CPU will be selected based on the critical speed. The whole process is repeated periodically with an

execution interval of 500ms.

**4.4 The Kernel-space Governor**

Based on the *cpufreq* subsystem structure, we have implemented a kernel space governor called KG- DVFS. It gets the MAR related information periodically from the PMU and sets a proper frequency which is close to the critical speed, according to the MAR-CSE equation. The sampling rate of our algorithm is by default set to 500ms. The flow chart of the algorithm is shown in the following

***Figure 4-4. The flow our kernel space governor***



***.***

When the KG-DVFS governor is initialized, a workqueue is created and a delayedwork is registered

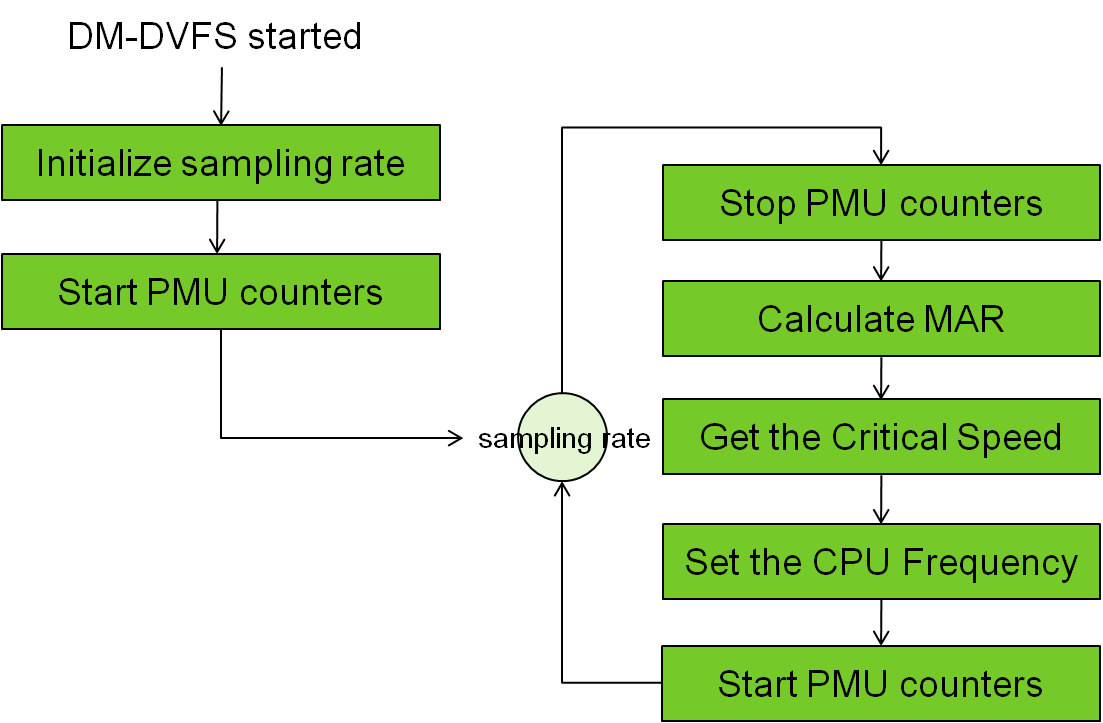
for retrieving the PMU data and setting the frequency periodically3. Each time the delayedwork is activated, the PMU information is collected to calculate the run-time MAR, which is applied to the MAR-CSE equation to predict an ideal frequency (critical speed.) Once the critical speed is found, a dual speed algorithm is used to find a pair of frequencies to approximate the ideal frequency.

**4.5 User-space Governors**

We have implemented two user space governors. The user space governors control the PMU hardware and the CPU frequency through the Linux sysfs interface that we have implemented in the kernel.

Daemon Implementation of the User-space Governor

One of user space governors is implemented as a Linux daemon process, named the DM- DVFS governor. Not like the kernel space governor, the daemon is running is user space. It gets PMU data through the sysfs interface periodically and sets a suitable frequency from the MAR-CSE equation. The following [Figure 4-4](#_bookmark1) shows the flow of the DM-DVFS governor.



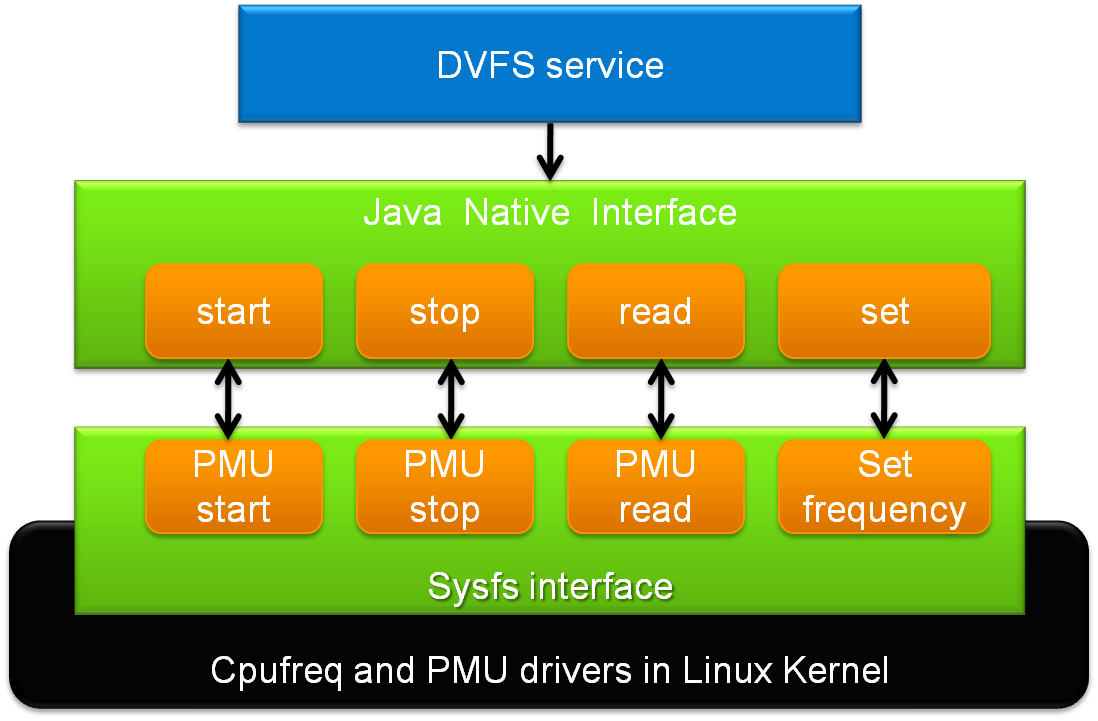
***Figure 4-5. The flow chart of DM-DVFS.***

Android Implementation of the User-space Governor

The other user space governor is implemented as an Android service on the Android operating system, called the AD-DVFS governor. Since the Android service is written in Java,theJavaNat

The Linux sysfs interface so as to access the PMU and to control the CPU frequency.

JNI is a native programming interface for the Java language. It allows the Java code that runs inside a Java Virtual Machine (VM) to interoperate with applications or libraries developed in other programming languages such as C, C++, and assembly. These applications or libraries are called native programs with respect to the Java programs. Programmers can write one version of a native code and expect it to work with all Java VMs supporting JNI. [25]



***Figure 4-5. Architecture of AD-DVFS.***

We have developed a JNI library for the AD-DVFS governor. The JNI library implements the native functions. It controls the CPU frequency and the PMU through the sysfs interface. Four major native functions listed below are supported by the library for these purposes.

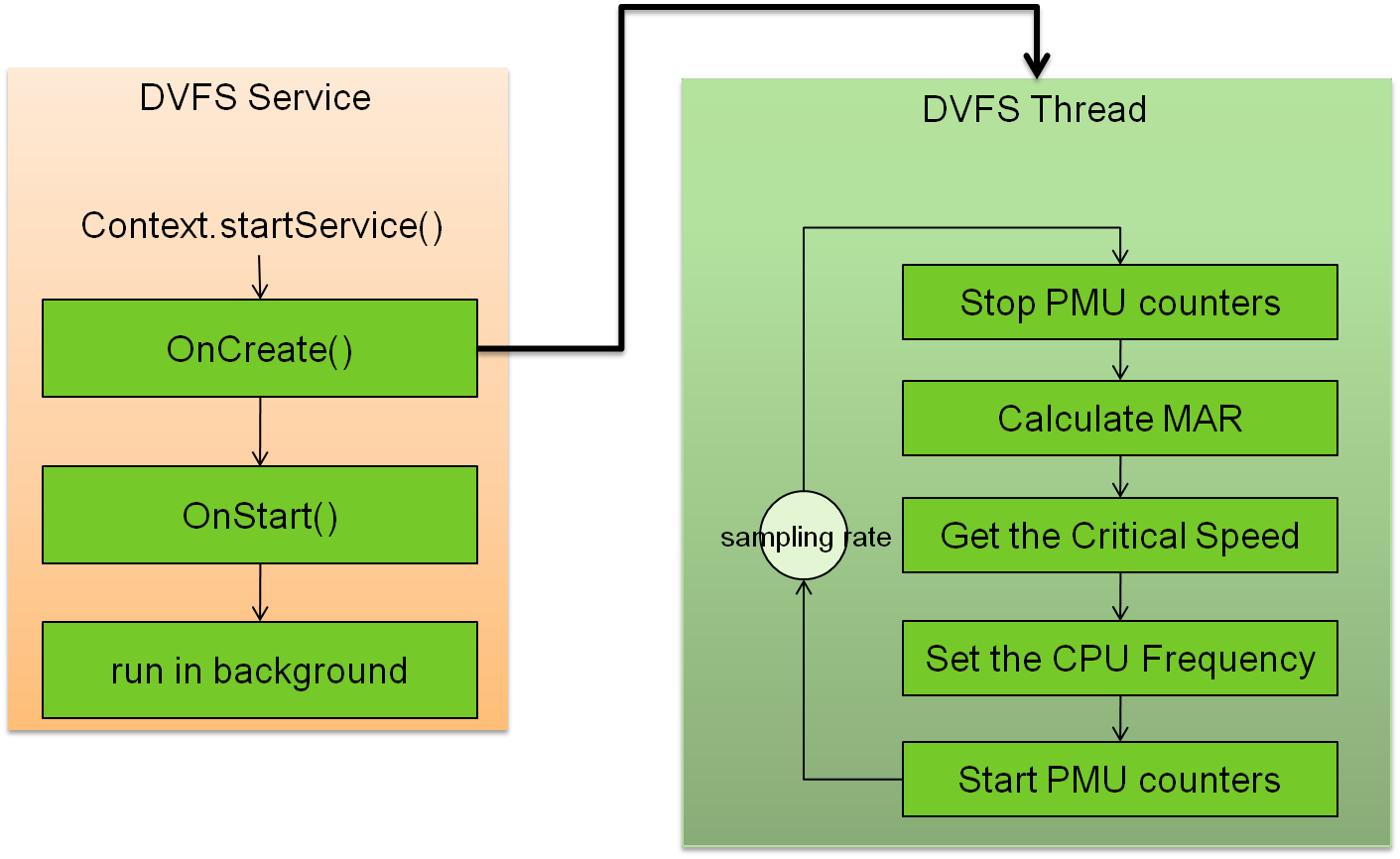
PMU\_Start(): This function starts the PMU counters.

PMU\_Stop(): This function stops the PMU counters.

PMU\_Read(): This function reads the pmu counters and array Containing the counter values.

Set\_Frequency(int): This function sets the CPU frequency throught an integer parameter.

The AD-DVFS service works. Once Android operation system has finished its booting process, the system broadcast the “***ACTION\_BOOT\_COMPLETED***” message. A ***BroadcastReceiver4*** is implemented for listening to this message and for starting the Android DVFS service, i.e., the AD-DVFS governor, after receiving this message. When the service has been started, it creates a JAVA thread for calculating MAR and setting an appropriate frequency from MAR-CSE periodically. The necessary information retrieval for MAR calculation and the control of the PMU and the frequency change are performed to the corresponding hardware indirectly through the JNI interface and the underlying *sysfs* provided by the Linux operating system.



The library function PMU\_Start() and PMU\_Stop() are used to start and stop the PMU counters. The function PMU\_Read() is used to read the values of the PMU counters. The other function Set\_Frequency() is required to set the CPU frequency. Beneath the sysfs interface, we have also implemented the necessary kernel-level supporting code to control the hardware. The structure of the AD-DVFS implementation has one major advantage. That is,

the policy manager of the AD-DVFS governor which is implemented in the Android application layer can be easily ported to machines running the Android operating system, leaving hardware dependent parts to lower-level codes.

In the execution flow of the AD-DVFS governor, the predicted critical speed will be calculated from the MAR-CSE equation, which is a cubic equation involving floating-point operations. This equation cannot be efficiently computed in the Android application layer as a Java program, mainly because the Java program needs to be run on the Java virtual machine by way of interpretation. This is especially important when the program is designed to be run on an embedded system with less computing power and with a concern of extra energy consumption. To shorten the computation time, in our implementation, the MAR-CSE equation has first been converted into a pre-built lookup table, so that an approximated solution can be rapidly calculated by the interpolation method. The curve is divided into ten levels for ease of computation, and at the same time to reduce the size of the lookup table.

Once the critical speed has been calculated, the target frequency will be chosen in the next step of the execution flow. Since the critical speed is typically not one of the available frequencies supported by the processor, a dual speed method is instead used to find two neighboring frequencies to approximate the critical speed. Fig. 7 shows how this method works. During the execution interval, a pair of neighboring frequencies, including the least highest frequency Freqhigh above the critical speed Freqcs and the maximum frequency Freqlow below Freqcs, and a switch point , are calculated and applied. The processor first runs with the frequency Freqhigh until the switch point, then switches to the frequency Freqlow and uses this until the end of the execution interval.

**5.0 IMPLEMENTATION**

# MODULES

**Module-1**

In this module a welcome screen is first shown with the name of the application and an option to continue then a user interface is provided to for the user to interact and select the required video from the lit of video files he wishes to play. The list of video files is displayed in a plain, simple and an uncomplicated manner. This makes the whole experience for the user very comfortable and it can even be easily understood by kids.

The list of files is shown from the local storage of the device. Once the user selects the required file he wishes to play, the work of this module is done.

**HomePageActivity**

**package** android.intel.sdp.MediaPlayer;  
  
**import** android.app.Activity;  
**import** android.content.Intent;  
**import** android.os.Bundle;  
**import** android.view.Menu;  
**import** android.view.View;  
**import** android.view.View.OnClickListener;  
**import** android.widget.Button;  
  
**public class** HomePageActivity **extends** Activity {  
 **private** Button continueBtn;  
 **private** Intent intent;  
 **private long** ms = 0;  
 **private boolean** paused = **false**;  
 **private boolean** splashActive = **true**;  
 **private long** splashTime = 2000;  
  
 */\* renamed from: android.intel.sdp.MediaPlayer.HomePageActivity$1 \*/* **class** C00001 **extends** Thread {  
 C00001() {  
 }  
  
 */\* JADX WARNING: inconsistent code. \*/  
 /\* Code decompiled incorrectly, please refer to instructions dump. \*/* **public void** run() {  
 */\*  
 r6 = this;  
 r4 = 100;  
 L\_0x0002:  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r0 = r0.splashActive; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 if (r0 == 0) goto L\_0x001a;  
 L\_0x000a:  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r0 = r0.ms; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r2 = android.intel.sdp.MediaPlayer.HomePageActivity.this; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r2 = r2.splashTime; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r0 = (r0 > r2 ? 1 : (r0 == r2 ? 0 : -1));  
 if (r0 < 0) goto L\_0x0034;  
 L\_0x001a:  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r1 = new android.content.Intent;  
 r2 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r3 = android.intel.sdp.MediaPlayer.FileChoosingActivity.class;  
 r1.<init>(r2, r3);  
 r0.intent = r1;  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r1 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r1 = r1.intent;  
 r0.startActivity(r1);  
 L\_0x0033:  
 return;  
 L\_0x0034:  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r0 = r0.paused; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 if (r0 != 0) goto L\_0x0046;  
 L\_0x003c:  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r1 = r0.ms; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 r1 = r1 + r4;  
 r0.ms = r1; Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 L\_0x0046:  
 r0 = 100;  
 android.intel.sdp.MediaPlayer.HomePageActivity.C00001.sleep(r0); Catch:{ Exception -> 0x004c, all -> 0x0067 }  
 goto L\_0x0002;  
 L\_0x004c:  
 r0 = move-exception;  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r1 = new android.content.Intent;  
 r2 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r3 = android.intel.sdp.MediaPlayer.FileChoosingActivity.class;  
 r1.<init>(r2, r3);  
 r0.intent = r1;  
 r0 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r1 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r1 = r1.intent;  
 r0.startActivity(r1);  
 goto L\_0x0033;  
 L\_0x0067:  
 r0 = move-exception;  
 r1 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r2 = new android.content.Intent;  
 r3 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r4 = android.intel.sdp.MediaPlayer.FileChoosingActivity.class;  
 r2.<init>(r3, r4);  
 r1.intent = r2;  
 r1 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r2 = android.intel.sdp.MediaPlayer.HomePageActivity.this;  
 r2 = r2.intent;  
 r1.startActivity(r2);  
 throw r0;  
 \*/* **throw new** UnsupportedOperationException(**"Method not decompiled: android.intel.sdp.MediaPlayer.HomePageActivity.1.run():void"**);  
 }  
 }  
  
 */\* renamed from: android.intel.sdp.MediaPlayer.HomePageActivity$2 \*/* **class** C00012 **implements** OnClickListener {  
 C00012() {  
 }  
  
 **public void** onClick(View v) {  
 HomePageActivity.**this**.intent = **new** Intent(HomePageActivity.**this**, FileChoosingActivity.**class**);  
 HomePageActivity.**this**.startActivity(HomePageActivity.**this**.intent);  
 }  
 }  
  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(C0002R.layout.activity\_home\_page);  
 **this**.continueBtn = (Button) findViewById(C0002R.id.continueBtn);  
 **new** C00001().start();  
 **this**.continueBtn.setOnClickListener(**new** C00012());  
 }  
  
 **public boolean** onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(C0002R.menu.home\_page, menu);  
 **return true**;  
 }  
}

# Module-2

In this module, application sends the video file selected towards the proposed DVFS where a number of steps are performed on the file where the least CPU frequency required by the device to play the file while maintaining the QoS(Quality of Service) is calculated. The QOS constraints are not compromised for energy efficiency.

FileChoosingActivity

**package** android.intel.sdp.MediaPlayer;  
  
**import** android.app.Activity;  
**import** android.content.Intent;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
**import** android.widget.Toast;  
  
**public class** FileChoosingActivity **extends** Activity {  
 **static** Uri uri;  
  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 getVideo();  
 }  
  
 **private void** getVideo() {  
 Intent pickMedia = **new** Intent(**"android.intent.action.GET\_CONTENT"**);  
 pickMedia.setType(**"video/\*"**);  
 startActivityForResult(pickMedia, 12345);  
 }  
  
 **public void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);  
 **if** (requestCode == 12345 && resultCode == -1) {  
 Uri selectedVideoLocation = data.getData();  
 Toast.makeText(**this**, selectedVideoLocation.toString(), 1).show();  
 uri = selectedVideoLocation;  
 startActivity(**new** Intent(**this**, MediaPlayerActivity.**class**));  
 }  
 }  
}

# Module-3

In this module, the file which was chosen by the user after having gone through the DVFS steps where the minimum CPU Frequency is calculated is placed into the buffer. A buffer is a temporary storage area, usually in RAM. The purpose of most buffers is to act as a holding area, enabling the CPU to manipulate data before transferring it to a device.

The processes of reading and writing data to a disk are relatively slow; many programs keep track of data changes in a buffer and then copy the buffer to a disk. This helps in maintaining efficiency.

# Module-4

In this module, finally the file placed in the buffer is fetched by the application and played.

The video is played till the user wants it to be played. The user is provided with three options; play, pause and stop. User can use any of these three given options to perform actions on the selected file. These options provide user with flexibility.

VideoPlayback

**package** android.intel.sdp.MediaPlayer;  
  
**import** android.app.Activity;  
**import** android.app.ActivityManager;  
**import** android.media.MediaPlayer;  
**import** android.media.MediaPlayer.OnPreparedListener;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
**import** android.os.Handler;  
**import** android.os.SystemClock;  
**import** android.util.Log;  
**import** android.view.SurfaceHolder;  
**import** android.view.SurfaceView;  
**import** android.view.View;  
**import** android.view.View.OnClickListener;  
**import** android.widget.ImageButton;  
**import** android.widget.TextView;  
**import** java.text.DecimalFormat;  
  
**public class** VideoPlayback **extends** Activity **implements** OnPreparedListener {  
 **private static final** String TAG = **"VideoPlayback"**;  
 **private long** mClipStartTime;  
 **private** Handler mHandler = **new** Handler();  
 **private** String mHeader;  
 **private** SurfaceHolder mHolder;  
 **private** String mIndicatorText;  
 **private** TextView mIndicator\_BottomLeft;  
 **private** TextView mIndicator\_BottomRight;  
 **private** TextView mIndicator\_TopLeft;  
 **private** TextView mIndicator\_TopRight;  
 **private** MediaPlayer mMediaPlayer;  
 **private** ImageButton mPause;  
 **private** ImageButton mPlay;  
 **private** SurfaceView mPreview;  
 **private long** mStartTime;  
 **private** ImageButton mStop;  
 **private** Runnable mUpdateTimeTask = **new** C00031();  
 **private** SDPUtility mUtility;  
 **private boolean** videoStopped = **true**;  
 **protected** Uri videoUri;  
  
 */\* renamed from: android.intel.sdp.MediaPlayer.VideoPlayback$1 \*/* **class** C00031 **implements** Runnable {  
 C00031() {  
 }  
  
 **public void** run() {  
 **if** (!VideoPlayback.**this**.videoStopped) {  
 **long** elapsedtime = SystemClock.elapsedRealtime() - VideoPlayback.**this**.mClipStartTime;  
 **long** playerElapsedTime = (**long**) VideoPlayback.**this**.mMediaPlayer.getCurrentPosition();  
 DecimalFormat df = **new** DecimalFormat(**"#.##"**);  
 String cpuUsage = df.format((**double**) (VideoPlayback.**this**.mUtility.readUsage() \* 100.0f)) + **"%"**;  
 String memUsage = **new** StringBuilder(String.valueOf(Long.toString(VideoPlayback.**this**.mUtility.readMem((ActivityManager) VideoPlayback.**this**.getSystemService(**"activity"**))))).append(**"M"**).toString();  
 VideoPlayback.**this**.mIndicatorText = **new** StringBuilder(String.valueOf(VideoPlayback.**this**.mHeader)).append(**"CPU usage: "**).append(cpuUsage).append(**"\nMemory usage: "**).append(memUsage).append(**"\nFPS: "**).append(df.format((**double**) ((23.0f \* ((**float**) playerElapsedTime)) / ((**float**) elapsedtime)))).toString();  
 VideoPlayback.**this**.updateIndicator();  
 }  
 VideoPlayback.**this**.mHandler.postAtTime(**this**, SystemClock.uptimeMillis() + 2000);  
 }  
 }  
  
 */\* renamed from: android.intel.sdp.MediaPlayer.VideoPlayback$2 \*/* **class** C00042 **implements** OnClickListener {  
 C00042() {  
 }  
  
 **public void** onClick(View view) {  
 **if** (VideoPlayback.**this**.mMediaPlayer == **null**) {  
 VideoPlayback.**this**.prepareVideo();  
 }  
 **if** (!VideoPlayback.**this**.mMediaPlayer.isPlaying()) {  
 VideoPlayback.**this**.mMediaPlayer.start();  
 VideoPlayback.**this**.videoStopped = **false**;  
 }  
 }  
 }  
  
 */\* renamed from: android.intel.sdp.MediaPlayer.VideoPlayback$3 \*/* **class** C00053 **implements** OnClickListener {  
 C00053() {  
 }  
  
 **public void** onClick(View view) {  
 **if** (VideoPlayback.**this**.mMediaPlayer != **null**) {  
 VideoPlayback.**this**.mMediaPlayer.pause();  
 }  
 }  
 }  
  
 */\* renamed from: android.intel.sdp.MediaPlayer.VideoPlayback$4 \*/* **class** C00064 **implements** OnClickListener {  
 C00064() {  
 }  
  
 **public void** onClick(View view) {  
 **if** (VideoPlayback.**this**.mMediaPlayer != **null**) {  
 VideoPlayback.**this**.stopVideo();  
 }  
 }  
 }  
  
 **protected** Uri getVideoUri() {  
 **return null**;  
 }  
  
 **public void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(C0002R.layout.media);  
 **this**.mPreview = (SurfaceView) findViewById(C0002R.id.surface);  
 **this**.mIndicator\_TopLeft = (TextView) findViewById(C0002R.id.textbox\_topleft);  
 **this**.mIndicator\_TopRight = (TextView) findViewById(C0002R.id.textbox\_topright);  
 **this**.mIndicator\_BottomLeft = (TextView) findViewById(C0002R.id.textbox\_bottomleft);  
 **this**.mIndicator\_BottomRight = (TextView) findViewById(C0002R.id.textbox\_bottomright);  
 **this**.mHolder = **this**.mPreview.getHolder();  
 **this**.mHolder.setType(3);  
 **this**.mIndicator\_TopLeft.setVisibility(4);  
 **this**.mIndicator\_BottomLeft.setVisibility(4);  
 **this**.mIndicator\_BottomRight.setVisibility(4);  
 **this**.mUtility = **new** SDPUtility();  
 **if** (**this**.mStartTime == 0) {  
 **this**.mStartTime = SystemClock.uptimeMillis();  
 **this**.mHandler.removeCallbacks(**this**.mUpdateTimeTask);  
 **this**.mHandler.postDelayed(**this**.mUpdateTimeTask, 100);  
 }  
 **this**.mHeader = **"Performance Indicators:\n"**;  
 **this**.mClipStartTime = SystemClock.elapsedRealtime() + 100;  
 **this**.mPreview = (SurfaceView) findViewById(C0002R.id.surface);  
 **this**.mPlay = (ImageButton) findViewById(C0002R.id.play);  
 **this**.mStop = (ImageButton) findViewById(C0002R.id.stop);  
 **this**.mPause = (ImageButton) findViewById(C0002R.id.pause);  
 **this**.mPlay.setOnClickListener(**new** C00042());  
 **this**.mPause.setOnClickListener(**new** C00053());  
 **this**.mStop.setOnClickListener(**new** C00064());  
 }  
  
 **private void** stopVideo() {  
 **this**.videoStopped = **true**;  
 **if** (**this**.mMediaPlayer != **null**) {  
 **this**.mMediaPlayer.stop();  
 **this**.mMediaPlayer.release();  
 **this**.mMediaPlayer = **null**;  
 }  
 }  
  
 **private void** prepareVideo() {  
 **try** {  
 **this**.videoUri = getVideoUri();  
 **this**.mMediaPlayer = **new** MediaPlayer();  
 **this**.mMediaPlayer.setDataSource(**this**, **this**.videoUri);  
 **this**.mMediaPlayer.setDisplay(**this**.mHolder);  
 **this**.mMediaPlayer.prepare();  
 **this**.mMediaPlayer.setOnPreparedListener(**this**);  
 **this**.mMediaPlayer.setAudioStreamType(3);  
 } **catch** (Exception e) {  
 Log.e(TAG, **"error: "** + e.getMessage(), e);  
 finish();  
 }  
 **this**.videoStopped = **false**;  
 }  
  
 **public void** onPrepared(MediaPlayer arg0) {  
 **this**.mMediaPlayer.start();  
 **this**.videoStopped = **false**;  
 }  
  
 **private void** updateIndicator() {  
 **this**.mIndicator\_TopRight.setText(**this**.mIndicatorText);  
 }  
}

**6.0 TESTING & Validation**

**6.1 INTRODUCTION:**

Testing is the debugging program is one of the most critical aspects of the computer programming triggers, without programming that works, the system would never produce an output of which it was designed. Testing is best performed when user development is asked to assist in identifying all errors and bugs. The sample data are used for testing. It is not quantity but quality of the data used the matters of testing. Testing is aimed at ensuring that the system was accurately an efficiently before live operation commands.

Testing objectives:

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say, testing is a process of executing a program with intent of finding an error.

A successful test is one that uncovers an as yet undiscovered error.

A good test case is one that has probability of finding an error, if it exists.

The test is inadequate to detect possibly present errors.

The software more or less confirms to the quality and reliable standards.

**6.2. Levels of Testing:**

Code testing:

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified.

Specification Testing:

Executing this specification starting what the program should do and how it should performed under various conditions. Test cases for various situation and combination of conditions in all the modules are tested.

Unit testing:

In the unit testing we test each module individually and integrate with the overall system. Unit testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during programming stage itself. In the testing step each module is found to work satisfactorily as regard to expected output from the module. There are some validation checks for fields also. For example the validation check is done for varying the user input given by the user which validity of the data entered. It is very easy to find error debut the system.

Each Module can be tested using the following two Strategies:

Black Box Testing

White Box Testing

**6.3 BLACK BOX TESTING**

What is Black Box Testing?

Black box testing is a software testing techniques in which functionality of the software under test (SUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.

In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.



The above Black Box can be any software system you want to test. For example : an operating system like Windows, a website like Google ,a database like Oracle or even your own custom application. Under Black Box Testing , you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Black box testing - Steps

Here are the generic steps followed to carry out any type of Black Box Testing.

Initially requirements and specifications of the system are examined.

Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.

Tester determines expected outputs for all those inputs.

Software tester constructs test cases with the selected inputs.

The test cases are executed.

Software tester compares the actual outputs with the expected outputs.

Defects if any are fixed and re-tested.

Types of Black Box Testing

There are many types of Black Box Testing but following are the prominent ones -

Functional testing – This black box testing type is related to functional requirements of a system; it is done by software testers.

Non-functional testing – This type of black box testing is not related to testing of a specific functionality, but non-functional requirements  such as performance, scalability, usability.

Regression testing – Regression testing is done  after code fixes , upgrades or any other system maintenance to check the new code has not affected the existing code.

**6.4 WHITE BOX TESTING**

White Box Testing is the testing of a software solution's internal coding and infrastructure.It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability.White box testing is also known as clear, open, structural, and glass box testing.

It is one of two parts of the "box testing" approach of software testing. Its counter-part, blackbox testing, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing. The term "whitebox" was used because of the see-through box concept. The clear box or whitebox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "black box testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested

What do you verify on white box testing?

White box testing involves the testing of the software code for the following:

Internal security holes

Broken or poorly structured paths in the coding processes

The flow of specific inputs through the code

Expected output

The functionality of conditional loops

Testing of each statement, object and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

**How do you perform White Box Testing?**

  To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

**STEP 1) UNDERSTAND THE SOURCE CODE**

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

**Step 2) CREATE TEST CASES AND EXECUTE**

The second basic step to white box testing involves testing the application’s source code for proper flow and structure. One way is by writing more code to test the application’s source code. The tester will develop little tests for each process or series of processes in the application. This  method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include manual testing, trial and error testing and the use of testing tools as we will explain further on in this article.

**6.5 System testing:**

Once the individual module testing is completed, modules are assembled and integrated to perform as a system. The top down testing, which began from upper level to lower level module, was carried out to check whether the entire system is performing satisfactorily.

There are three main kinds of System testing:

Alpha Testing

Beta Testing

Acceptance Testing

Alpha Testing:

This refers to the system testing that is carried out by the test team with the Organization.

Beta Testing:

This refers to the system testing that is performed by a selected group of friendly customers

Acceptance Testing:

This refers to the system testing that is performed by the customer to determine whether or not to accept the delivery of the system.

Integration Testing:

Data can be lost across an interface, one module can have an adverse effort on the other sub functions, when combined, may not produce the desired major functions. Integrated testing is the systematic testing for constructing the uncover errors within the interface. The testing was done with sample data. The developed system has run successfully for this sample data. The need for integrated test is to find the overall system performance.

Output testing:

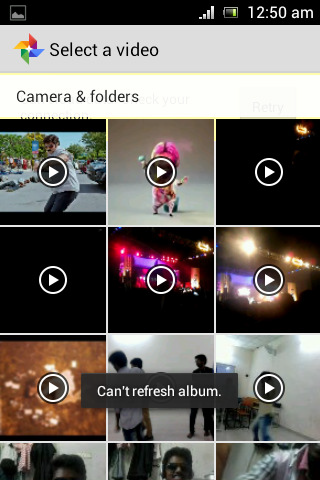
After performance of the validation testing, the next step is output testing. The output displayed or generated by the system under consideration is tested by asking the user about the format required by system. The output format on the screen is found to be correct as format was designed in the system phase according to the user needs. Hence the output testing does not result in any correction in the system.

Test plan:

The test-plan is basically a list of testcases that need to be run on the system. Some of the testcases can be run independently for some components (report generation from the database, for example, can be tested independently) and some of the testcases require the whole system to be ready for their execution. It is better to test each component as and when it is ready before integrating the components. It is important to note that the testcases cover all the aspects of the system (ie, all the requirements stated in the RS document).

**7.0 SCREENSHOTS**

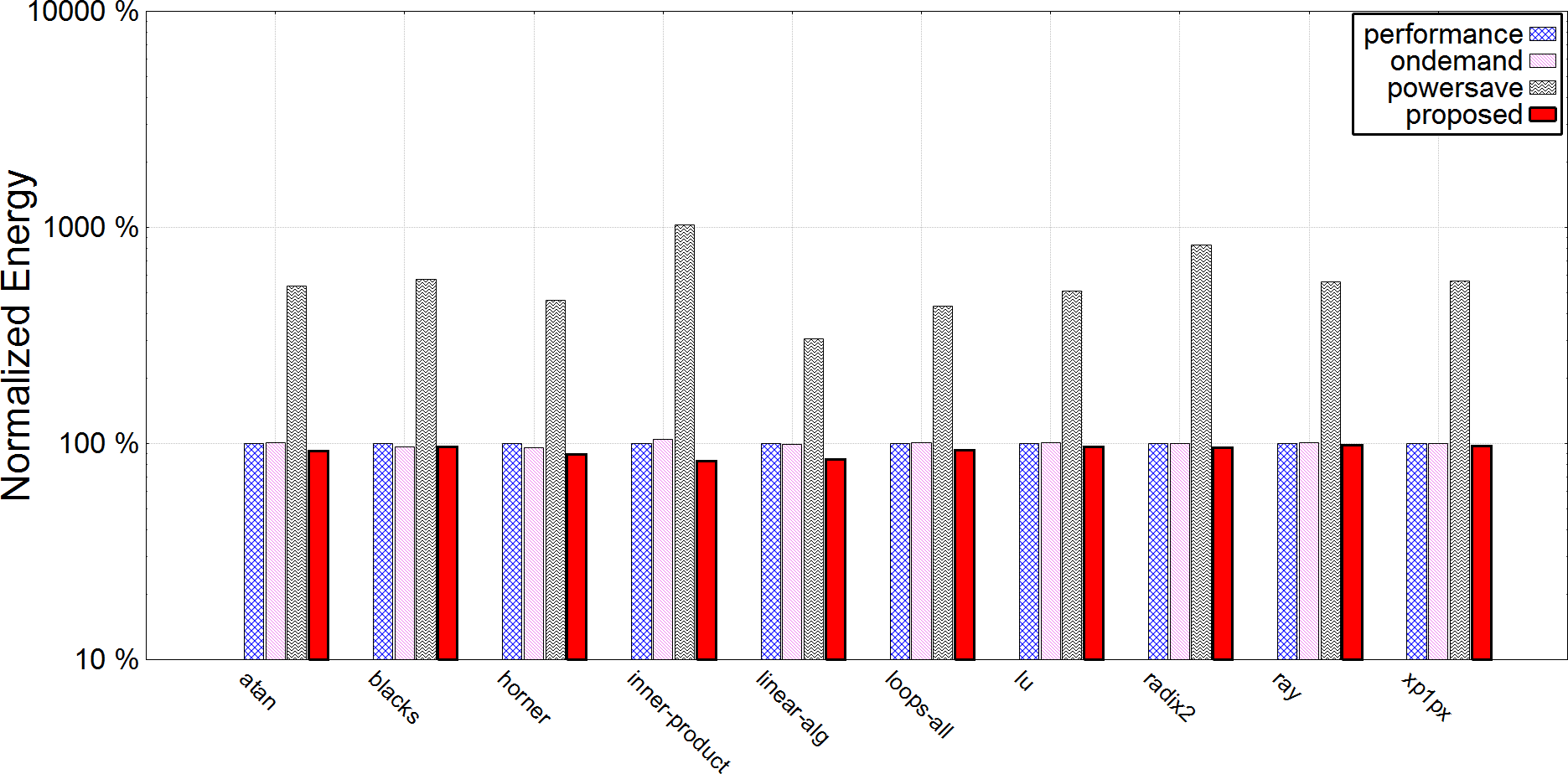
****

****

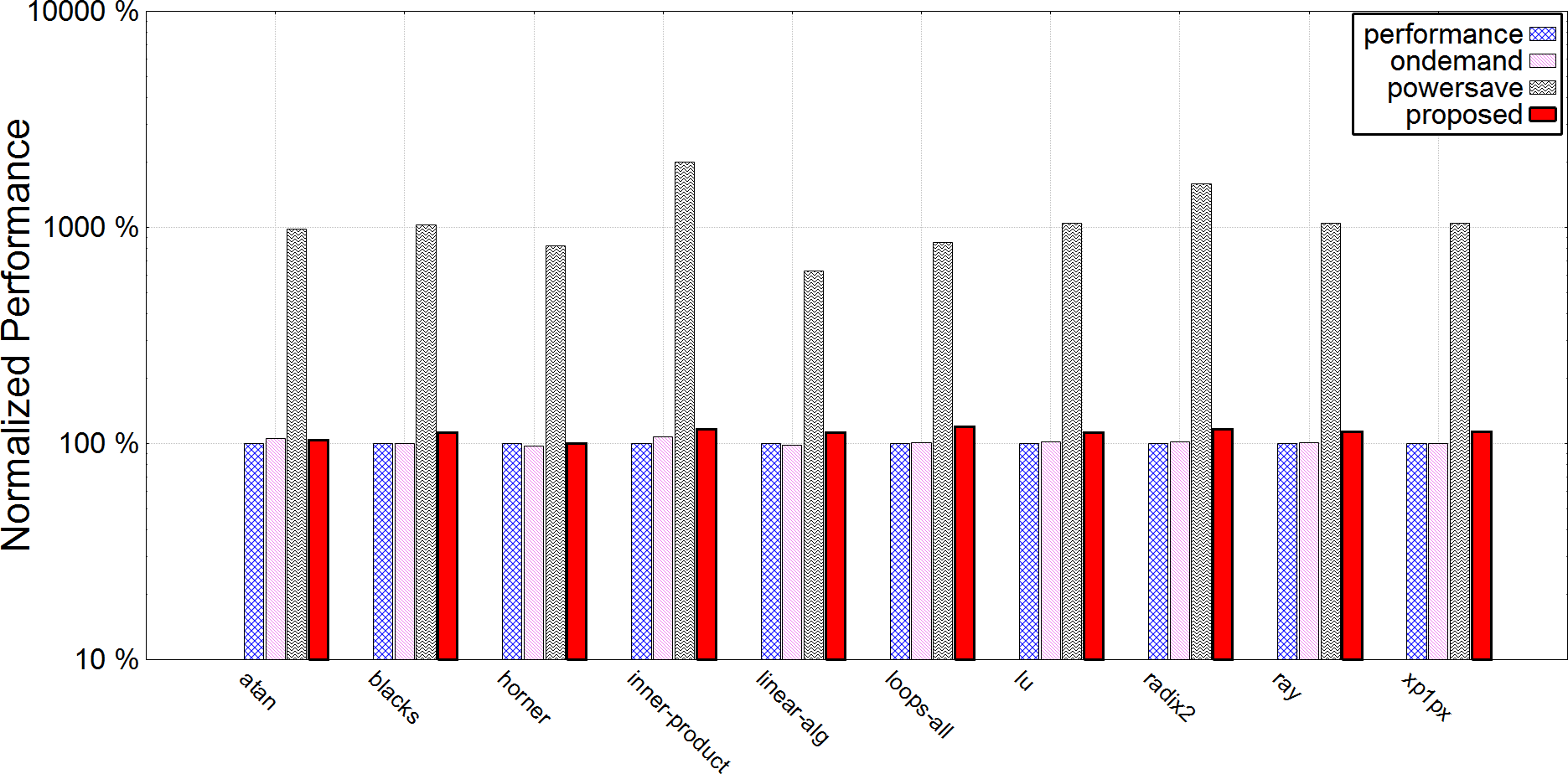
****

**8.0 CONCLUSION**

We propose an OI-based DVFS algorithm for ARM embedded systems. It controls the optimal frequency based on information that includes memory activity of the applications. To implement the proposed algorithm, we used bus information, based on PMU, instead of cache-related monitoring regis- ters. Unfortunately, bus information is only supported by the latest ARM cores. We explained the relationship between OI and energy, and implemen- tation of the algorithm on the basis of the information based on the OI.



(a)



(b)

**Fig. 4.** The result of proposed OI-based DVFS algorithm.

(a) Energy, (b) Performance

The OI-based DVFS algorithm shows that the energy consumption was, on average, 9% lower compared with a performance governor. In the case of a memory-bound application, the difference can be up to 17% in energy.

**9.0 Future Enhancements**

Adding advanced features like:

* 1. Volume control
  2. Brightness control
  3. Slide to scrub
  4. Play back through cloud storage

# References

[1] K. Choi, R. Soma, and M. Pedarm, “Dynamic Voltage and Frequency Scaling based on Workload Decomposition,” in Proc. 2004 Int. Symp. Low Power Electronics and Design, Aug. 2004, pp. 174-179, 2004. [

2] K. Choi, R. Soma, and M. Pedram, “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 Proc. 2004 Design, Automation and Test in Europe, vol. 1, p. 10004, 2004.

D. Rajan, R. Zuck, and C. Poellabauer, “Workload-Aware Dual-Speed Dynamic Voltage Scaling,” in Proc. 12th IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications, pp. 251-256. 2006..

D. Rajan, R. Zuck, and C. Poellabauer, “A Dual Speed Approach to Workload-Aware Voltage Scaling,” Technical Report TR-2006-05, University of Notre Dame, 2006..

R. Jejurikar and R. Gupta, “Dynamic Voltage Scaling for Systemwide Energy Minimization in Real- Time Embedded Systems,” in Proc. of 2004 Int. Symp. Low Power Electronics and Design, pp. 78-81, 2004.

W.-Y. Liang; S.-C. Chen; Y.-L. Chang; J.-P. Fang, “Memory-Aware Dynamic Voltage and Frequency Prediction for Portable Devices,” in Proc. 14th IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications, pp. 229-236, 2008.

Android Developers website, [Online]. Available at: [http://developer.android.com](http://developer.android.com/) [Accessed: Dec. 2010].

P. Venkatesh and S. Alexey, “The Ondemand Governor” in Proc. Linux Symposium, vol. 2, pp. 223- 238, 2006, Available at: <http://www.linuxinsight.com/files/ols2006/pallipadi-reprint.pdf>

**APPENDIX I**

**RELEVANCE OF PROJECT TO POs and PSOs**

|  |  |
| --- | --- |
| **PROJECT TITLE** | **CATEGORY** |
|
| Energy Efficient Video Player for Android | UTILITY |

**ABSTRACT:**

Media player support all modern electronic devices nowadays. However, power management is one of the most critical issues in the design of today’s Media player. The goal of power management is to maximize performance within a given power budget The main aim of this project is to reduce the electrical energy costs of operating data centers of media player. A common way of cost reduction is to perform dynamic voltage and frequency scaling (DVFS), thereby matching the CPU’s performance and power level to incoming workloads. Another power saving technique is CPU consolidation, which uses the minimum number of CPUs necessary to meet the service request demands and turns off the remaining unused CPUs. DVFS has been already extensively studied and verified its effectiveness. On the other hand, it is necessary to study more about effectiveness of CPU consolidation. Key questions that must be answered are how effectively the CPU consolidation improves the energy efficiency and how to maximize the improvement. These questions are addressed in this paper. After understanding modern power management techniques and developing an appropriate power model, this paper provides an extensive set of hardware-based experimental results and makes suggestions about how to maximize energy efficiency improvement through CPU consolidation. In addition, CPU consolidation algorithms, which reduce the energy delay product up to 13% compared to the Linux default DVFS algorithm

However, power management is one of the most critical issues in the design of today’s Media player. The goal of power management is to maximize performance within a given power budget. Power management techniques must balance between the demanding needs for higher performance/throughput and the impact of aggressive power consumption and negative thermal effects. Many techniques have been proposed in this area, and some of them have been implemented such as the well-known DVFS technique which is used in nearly all modern Media player. This paper explores the concepts of multi-core, trending research areas in the field of Media player and then concentrates on power management issues in multi-core architectures. Moreover, it proposes a new technique for power management in Media player based on that survey.

.

**Table [put table no] Mapping to POs: (Mapping Scale[1-3]: 1-Slight 2-Moderate**

**3-Strong)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | | **PO12** |
| **3** | **2** | **3** | **2** | **2** | **3** | **1** | **3** | **1** | **3** | **1** | **2** | |

**Table [put table no] Mapping to PSOs: (Mapping Scale[1-3]: 1-Slight 2-Moderate**

**3-Strong)**

|  |  |  |
| --- | --- | --- |
| **PSO1** | **PSO2** | **PSO3** |
| **3** | **2** | **3** |

|  |  |
| --- | --- |
| **RELEVANCE DETAILS** | |
| **Implementation details**  The project is implemented using Languages: java, html, xml in android studio and eclipse | |
| **PO and PSO Justification** | |
| **PO1** | It is strongly mapped as the basic concepts of mathematics, physics and engineering fundamentals are used for designing a solution to the complex usage of cpu and battery encountered problems |
| **PO2** | It is moderately mapped mapped as low transmission rate of data pf video file from memory to data buffer consumes time. |
| **PO3** | It is strongly mapped as risks, responsiveness and efficiency are considered to cater to the societal needs, health and public safety. |
| **PO4** | It is moderately mapped as many testing’s conducted on different devices showed varying results but with the beneficial factor of reduced battery usage. Critical thinking and the ability to analyse and solve complex real-world problems is done for developing the product. |
| **PO5** | It is moderately mapped as media player with DVFS algorithm Is not widely used but widely and well known for cutting down the cpu consumption. |
| **PO6** | It is strongly mapped as the use of DVFS helps cutting down cpu consumption in turn cuts down battery usage which is beneficial for society. |
| **PO7** | It is weakly mapped as the project need to be sustainable to the environment but most of the projects designed in the market aim at less battery usage |
| **PO8** | It is strongly mapped because the plagiarism tool can be used to check the authenticity and originality of the work done by the students |
| **PO9** | Projects are used to inculcate group work and to manage a team for promoting knowledge ,conceptualization and delivering same with varied complexity ,therefore the mapping is strong. |
| **PO10** | Demonstrate versatile and effective communication skills, both verbal and written with team members and present the product to the audience in comprehensive manner. Therefore the mapping is moderate. |
| **PO11** | Project Management tools like CPM and Pert etc can be used to propose and work accordingly, hence the mapping is moderate. |
| **PO12** | The mapping is moderate as projects are executed based on the self-learning or self-efforts put in by the group |
| **PSO1** | It is strongly mapped as understanding of the principles and working of the hardware and software aspects of computer systems is required to decomposed the system into phases and workflows. |
| **PSO2** | It is moderately mapped as cases are explored to ensure reliability of the product ,optimal quality and tested thoroughly to validate the ongoing process |
| **PSO3** | It is strongly mapped as self learning skills are applied for a real world problem by using content knowledge and acquired intellectual skills. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Title** | Energy Efficient Video Player for Android |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Project Supervisor** | Mrs. GOURI PAIL |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Start Date** | 4-Apr-18 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **End Date** | 12-Apr-19 | Start | End | Days | Status |  | | | | | | | | | | |
| **1** | **Project and Batch Registration** | **4-Apr-18** | **27-Apr-18** | **23** | **Completed** |
| 2 | Problem Definition | 28-Apr-18 | 7-May-18 | 9 | **Completed** |
| 3 | Abstract | 20-Jun-18 | 9-Aug-18 | 20 | **Completed** |
| 4 | Literature Survey | 9-Aug-18 | 11-Sep-18 | 33 | **Completed** |
| 5 | Requirements Analysis (including H/w and S/w) | 9-Aug-18 | 11-Sep-18 | 33 | **Completed** |
| **6** | **Project Seminar** | **18-Aug-18** | **16-Sep-18** |  | **Completed** |
| **7** | **Project Review 1** | **6-Oct-18** | **10-Oct-18** | **2** | **Completed** |
| 8 | Feasibility Study | 11-Sep-18 | 29-Sep-18 | 18 | **Completed** |
| 9 | Detailed Design / Module Identification | 16-Sep-18 | 30-Sep-18 | 14 | **Completed** |
| 11 | Implementation | 25-Sep-18 | 10-mar-18 | 166 | **Completed** |
| 12 | Testing | 1-jan-19 | 10-Feb-19 | 68 | **Completed** |
| 13 | Result Analysis | 10-Feb-19 | 10-mar-19 | 37 | **Completed** |
| **14** | **Project Review2** | **11-Mar-19** | **16-Mar-19** | 5 | **Completed** |
| 15 | Report Compilation |  |  | 11 | **Completed** |
| **16** | Viva Voice | 15-Apr-19 | 27-Apr-19 | 12 | **---** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | | | | | | | | | | | | | |