UNIVERSITÉ DE NICE – SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice – Sophia Antipolis

Mention : Électronique

Présentée et soutenue par

Jabran KHAN JADOON

Evaluation of Power Management Strategies on Actual Multiprocessor Platforms

---

Thèse dirigée par Cécile BELLEUDY

Laboratoire LEAT, Université de Nice - Sophia Antipolis - CNRS

Soutenue en 25 Mars 2013

Jury:

Nathalie JULIEN	Professeur, Université de Bretagne-Sud	Rapporteurs
François PECHEUX	Maître de Conférence, Université Pierre et Marie Curie	
Smail NIAR	Professeur, Université de Valenciennes et Hainaut-Cambrésis	Examinateurs
Michel Auguin	Directeur de Recherche CNRS	
Cécile BELLEUDY	Maître de Conférence, Université de Nice-Sophia Antipolis	
Sébastien BILAVARN	Maître de Conférence, Université de Nice-Sophia Antipolis	

@khan

ALL RIGHTS RESERVED

UNIVERSITÉ DE NICE – SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice – Sophia Antipolis

Mention : Électronique

Présentée et soutenue par

Jabran KHAN JADOON

Evaluation of Power Management Strategies on Actual Multiprocessor Platforms

---

Thèse dirigée par Cécile BELLEUDY

Laboratoire LEAT, Université de Nice - Sophia Antipolis - CNRS

Soutenue en 25 Mars 2013

Jury:

Nathalie JULIEN	Professeur, Université de Bretagne-Sud	Rapporteurs
François PECHEUX	Maître de Conférence, Université Pierre et Marie Curie	
Smail NIAR	Professeur, Université de Valenciennes et Hainaut-Cambrésis	Examinateurs
Michel Auguin	Directeur de Recherche CNRS	
Cécile BELLEUDY	Maître de Conférence, Université de Nice-Sophia Antipolis	
Sébastien BILAVARN	Maître de Conférence, Université de Nice-Sophia Antipolis	

ABSTRACT

The purpose of this study is to investigate how power management strategies can be efficiently exploited in actual platforms. Primarily, the challenges in multicore based embedded systems lies in managing the energy expenditure, determining the scheduling behavior and establishing methods to monitor power and energy, so as to meet the demands of the battery life and load requirements. The work presented in this dissertation is a study of low power-aware strategies in the practical world for single and multiprocessor platforms. The approach used for this study is based on representative multiprocessor platforms (real or virtual) to identify the most influential parameters, at hardware as well as application level, unlike many existing works in which these parameters are often underestimated or sometimes even ignored. The work analyzes and compares in detail various experimentations with different power policies based on Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Switching (DPS) techniques, and investigates the conditions at which these policies are effective in terms of energy savings.

The results of these investigations reveal many interesting and notable conclusions that can serve as prerequisites for the efficient use of power management strategies. This work also shows the potential of advanced domain specific power strategies compared to real world available strategies that are general purpose based in their majority. Finally, some high level consumption models are derived from the different energy measurement results to let the estimation of power management benefits at early stages of a system development.

Dedicated to my beloved mother, father, brothers and sisters

and

to my beloved wife Mahrukh.

ACKNOWLEDGEMENTS

One of the joys of completion is to look over the journey past and remember all the friends and family who have helped and supported me along this long but fulfilling road. Completion of my PhD required countless selfless acts of support, generosity, and time by people in my personal and academic life. I can only attempt to humbly acknowledge and thank the people and institutions that have given me so much help throughout my PhD career and made this dissertation possible. I am thankful to the Higher Education Commission (HEC) of Pakistan for providing funding throughout my Masters and PhD career.

I am sincerely thankful to Miss. Cécile Belleudy, my advisor, for being a constant source of invaluable encouragement, aid, and expertise during my years at LEAT. I would also like to express my heartfelt gratitude to my co-supervisor Sébastien Bilavarn. He has provided insightful discussions about the research. I am very grateful for his scientific advice and knowledge and many insightful discussions and suggestions. The mentoring, friendship, and collegiality of both Cécile Belleudy and Sébastien Bilavarn enriched my academic life and have left a profound impression on how academic research and collaboration should ideally be conducted.

I would also like to thank Professor Nathalie Julien and François Pecheux for being my reviewers and thoroughly reading and acknowledging my thesis. In addition, I would like to thank Professor Michel Auguin and Professor Smail Niar to be part of my jury. It is no easy task, reviewing a thesis, and I am grateful for their detailed reviewing. To the many anonymous reviewers at the various international conferences, thank you for helping to shape and guide the direction of the work with your careful and instructive comments.

LEAT has provided a rich and fertile environment to study and explore new ideas. At LEAT, I would first like to thank Micheal Auguin who has been extremely supportive in allowing me to participate in lab activities whilst pursuing my PhD studies. I am grateful for the chance to be a part of the lab and thank for welcoming me as a friend and helping to develop the ideas in this thesis. Other colleagues who I owe gratitude for their support of my research or major PhD milestones include: Alain Pegatoquet, Bassem Ouni, Ons Mbarek, Khurram Bhatti, and Zeeshan Khan.

My family and friends have been an unending source of love and inspiration throughout my PhD career. I would not have contemplated this road if not for my parents, Hameedullah Khan and Yasmeen Akhtar, who instilled within me a love of creative pursuits, science and language, and believe in me, all of which finds a place in this thesis. To my parents, thank you. My wife, Mahrukh, has offered unconditional understanding and encouragement. My sisters, Madiha and Rawish, have kept me sane with their humor and understanding even from distance. My brothers, Ajram and Ahmer, have been great and selfless support to me throughout these years of my absence from home. My friends in French Riviera, Sharique, Usman, Imran, Sabir, Umer, Taimour, Ahmed, Wajahat, Waqas, and Asad have provided hours of enjoyable distraction from my work. I will always remember the time I have shared with them.

TABLE OF CONTENTS

ABSTRACT ..... **IV**

TABLE OF CONTENTS ..... **IV**

LIST OF FIGURES ..... **VIII**

LIST OF TABLES ..... **X**

CHAPTER 1 : INTRODUCTION **1**

1.1 Contributions Outline 1

1.2 List of Publications 3

CHAPTER 2 : POWER MANAGEMENT CHALLENGES **4**

2.1 State of the Art 6

2.1.1 Overview and Classification of Work 7

2.1.2 Dynamic Power Switching (DPS) 7

2.1.3 DVFS Techniques 12

2.1.4 Available Market Standards 13

2.1.4.1 Intel 14

2.1.4.2 AMD 16

2.1.4.3 ARM 17

2.1.4.4 Linux 19

2.1.4.5 Conclusion 23

2.1.5 Academic Research 23

2.1.5.1 Overview of Academic Research 23

2.1.5.2 Low Power Scheduling 25

2.1.6 Conclusion 28

2.1.7 Focus and Objectives 29
2.2 Work Context 31
2.2.1 Problem Statement 32
2.2.2 Platforms 33
2.2.2.1 ARM11 MPCore 34
2.2.2.2 ARM 1176JZF-S 35
2.2.2.3 QEMU_ARM1176 37
2.2.2.4 QEMU_CortexA9 38
2.2.3 Power Strategies 39
2.2.3.1 DVFS Video Power Strategy 39
2.2.3.2 Low Power DSF Scheduler 39
2.2.3.3 Low Power AsDPM Scheduler 40
CHAPTER 3 : DVFS VIDEO POWER STRATEGY 42
3.1 Introduction 42
3.1.1 Case study: H.264 Decoder 44
3.1.2 DVFS Video Strategy Description 44
3.2 DVFS Strategy Implementation and Experimentation 46
3.2.1 DVFS Strategy Implementation 46
3.2.2 Power and Frame rate profiles 47
3.2.3 Energy Consumption Analysis 50
3.3 Further Investigation of Energy Saving Conditions 51
3.3.1 Operating Point Set up on the Virtual Platform 52
3.3.2 Accuracy and Behavior of Virtual Platform Estimations 53
3.3.3 Results and Discussion 54
3.4 Conclusion 57

CHAPTER 4 : DSF POWER STRATEGY 60

4.1 Introduction 60

4.1.1 DSF Strategy 61

4.1.2 Application Examples 61

4.2 DSF Implementation and Experimentation 64

4.2.1 DSF Implementation 64

4.2.2 Experimentation on a Single Processor 65

4.2.3 Experimentation with Application Parameters 67

4.3 Results and Analysis 69

4.3.1 Multiprocessor Energy Savings 69

4.3.2 Analysis of Results 70

4.4 Conclusion 71

CHAPTER 5 : ASDPM POWER STRATEGY 74

5.1 Introduction 74

5.1.1 AsDPM Strategy 75

5.1.2 Application Examples 76

5.2 AsDPM Implementation and Experimentation 78

5.2.1 AsDPM Implementation 78

5.2.2 Energy Savings 79

5.2.3 Further Analysis of Results 82

5.3 Energy Gain Comparison of DSF and AsDPM 83

5.3.1 Energy Gains for DSF 83

5.3.2 Comparison and Analysis of Results 85

5.4 Conclusion 87

CHAPTER 6 : GLOBAL ANALYSIS AND CONCLUSION 90
6.1 Power Management Effectiveness 90
6.1.1 Characteristics of Operating Points 91
6.1.1.1 Operating Points Inefficiency 91
6.1.1.2 Operating Points Impact on Energy Savings 92
6.1.2 Latencies of Changing States 93
6.1.3 Application Level Conditions 94
6.1.4 Domain Specific Strategies 95
6.1.5 Efficiency of DVFS vs. DPS 96
6.1.5.1 DSF vs. AsDPM 96
6.1.5.2 DVFS vs. DPS 97
6.2 Power Models 97
6.2.1 DVFS Video Strategy 98
6.2.2 DSF 99
6.2.3 AsDPM 100
6.3 Conclusion and Perspectives 101
6.3.1 Perspectives 103
VITA 104
BIBLIOGRAPHY 105
APPENDIX A – ACRONYM INDEX 112

LIST OF FIGURES

Figure 2.1: Global System Power States and Transitions. [9] 9

Figure 2.2: Power management model with ACPI subsystem. [9] 11

Figure 2.3: Intelligent Energy Management Solution by ARM. [22] 19

Figure 3.1: Power and frame rate profiles for *adaptation_constraint* of 8 fps. 48

Figure 3.2: Power and frame rate profiles for *adaptation_constraint* of 9 fps. 48

Figure 3.3: Power and frame rate profiles for *adaptation_constraint* of 11 fps. 49

Figure 3.4: Power and frame rate profiles for *adaptation_constraint* of 16 fps. 49

Figure 3.5: Frequency and load power consumption of QEMU platform. 52

Figure 3.6: Energy consumption vs. Adaptation constraint. 55

Figure 3.7: Energy vs. adaptation_constraint for different platform configurations.....57

Figure 4.1: Thales Task model of H.264 Encoder. 63

Figure 4.2: Percentage Energy gains for Example 1 on different platforms. 66

Figure 4.3: Percentage of Energy Gain vs. Application parameters. 68

Figure 4.4: Energy gains on QEMU_ARM1176 (blue) and QEMU_CortexA9 (green) platforms for different applications and platform configurations. 69

Figure 5.1: Energy gains of Example 6 on QEMU platforms. 80

Figure 5.2: Energy gains of Example 7 on QEMU platforms. 80

Figure 5.3: Energy gains of *H.264Encoder* on QEMU platforms. 81

Figure 5.4: Energy Gains for DSF Strategy on QEMU platforms. 84

Figure 5.5: AsDPM vs. DSF energy gains for different examples on QEMU_ARM1176. 85

Figure 5.6: AsDPM vs. DSF energy gains for different examples on QEMU_CortexA9. 86

Figure 6.1: Energy Model for DVFS based Video Power strategy. 98
Figure 6.2: Energy Model for DSF power strategy. 99
Figure 6.3: Energy Model for AsDPM strategy. 100

LIST OF TABLES

Table 2.1: Supported Performance States for the Intel M-Processor [14] 13

Table 2.2: Supported Voltages and Frequencies for Low Power AMD-K6™2E+Processors[20] 17

Table 3.1: Performance Analysis table for ARM1176JZF-S 51

Table 3.2: Frequency vs. Energy for video Foreman (300 frames). 54

Table 3.3: Frequency and load power consumption for different configuration of QEMU platform. 55

Table 4.1: Examples used for experimentation of DSF strategy 62

Table 4.2: Energy consumption of *Example 1* on ARM1176JZF-S and QEMU_ARM1176 platform. 65

Table 4.3: Power profile of QEMU_ARM1176 and QEMU_CortexA9 platform. 71

Table 5.1: Examples used for the experimentation of AsDPM strategy. 76

Table 5.2: Energy consumption of *Example 5* on QEMU platforms. 79

Table 5.3: Energy Consumption and Gain for *H.264Encoder*. 82

Table 5.4: Idle vs. load power levels for QEMU platforms at maximum frequency. 83

This thesis is partially funded by COMCAS project (CA501), a project labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics; and partially by HEC PAKISTAN under program Overseas Scholarships for MS MS/MPHIL Leading to PhD in selected Fields (Phase-II, Batch-II).

Chapter 1 : INTRODUCTION

This introduction aims at presenting the reader with a global outline of this study. The introduction part provides a general discussion on trends of energy consumption in modern systems, presents general objectives of this work and provides an outline of this thesis. A detailed literature review, work context, as well as problem statement are presented later in the respective chapter.

In recent years, there has been a rapid and wide spread growth of nontraditional computing platforms, especially mobile and portable computing devices. It is a common experience that current mobile devices (laptops, PDAs, etc.) can operate just for few hours before the battery gets exhausted. As applications are becoming more and more complex and processing power is continuously increasing, there is a significant impact on battery life. Embedded systems due to the advancement made in fabrication of powerful tiny processors, with the ability of having multiple cores and variable frequencies can be integrated in small handheld devices. The small physical size and limited battery life of these devices impose several constraints on power and energy consumption of these devices. In this context, most systems should be designed in a way to adapt themselves according to environment needs, precisely they need to adapt at the lowest energy consumption for a given performance level.

These constraints require rethinking the design process with power and energy issues as major concerns. It is really a waste if the equipments are not designed to be energy efficient with the abilities to power down during non-operating hours. Energy consumption is becoming more of a concern and is getting an increasingly larger percentage of the overall development costs as well. Reducing the energy consumption in embedded systems has become a prime criterion, motivated by the limited lifetime of battery operated systems.

This thesis provides a characterization of real challenges encountered while providing advance power management solutions in actual embedded systems. This thesis also provides a detailed analysis of power management challenges and review power saving techniques available with today's system-on-chip (SoC). We analyze different power management strategies on single and multiprocessor platforms and explore how different power management policies, each with their own methodologies, can really provide energy savings. The real implementation of different power management strategies

coincide with different issues therefore this work present a realistic approach by experimenting with different power management strategies in real world. This thesis demonstrates experimentation and analysis of three different domain specific power management strategies with the objective of pointing out different factors effecting the energy consumption. In addition, we provide different experimentation results to evaluate the effectiveness of these strategies and provide conditions under which the addressed strategies are feasible in modern systems.

In next section, we present a global outline of our contribution to address the above issues.

1.1 Contributions Outline

Chapter 2: This chapter provides a detailed literature overview of our work. At the beginning of this chapter, we provide a detailed study of main power management solutions, industrial power management standards and academic research going in the field of power management. Afterwards, we introduce the focus and targets of our work to meet the power management challenges and consequently define our problem statement. The chapter then provides information about the target platforms and ends with the brief introduction to the different power strategies used to address the problem statement.

Chapter 3: This chapter provides a detailed experimentation about the first DVFS based video power strategy. The DVFS based strategies are shown to provide significant energy gains, where scaling down the frequency mostly provides decrease in power consumption. The experimentation and results provided in this chapter shows that various conditions should be met to achieve this energy savings. The chapter also highlights the importance of using domain or application specific strategies. At the end, the effect of operating points on the efficiency of a DVFS strategy and their influence on the energy savings is also investigated.

Chapter 4: This chapter provides experimentation of a second DVFS based *Dynamic Stretch to Fit* (DSF) strategy on single and multiprocessor platforms. DSF strategy is shown to provide significant energy gains in different platform configurations. The experimentations highlight the effects of load vs. idle power levels of the platforms. This chapter also provide certain application and platform limitations for using DSF strategy

as well as draw attention to the efficiency criteria of application or domain specific strategy.

Chapter 5: This chapter provides an analysis and experimentation of a Dynamic Power Switching (DPS) based power strategy for multiprocessor platforms called *Assertive Dynamic Power Management* (AsDPM). The application specific strategy is analyzed for its energy consumption and is further investigated for the conditions under which it is more efficient on certain platform than other. The chapter also provides the effects introduced by different application parameters, platform characteristics and state switching latencies on the efficiency of the AsDPM strategy. In addition, this chapter provides a comparative analysis of energy gains obtained by AsDPM with previous DSF strategy.

Chapter 6: This chapter provides major categorization of conditions obtained from the results of the above experimentations. First, it addresses power management effectiveness by classifying different application conditions, platform conditions, latencies of DVFS / DPS states and domain vs. general purpose strategies. Secondly, this chapter provides some high level power models derived from the results of energy gains obtained by our experimentations. At the end, this chapter provides a brief conclusion along with perspectives of furure work.

1.2 List of Publications

Personal publications in International Conferences:

1) **Khan, J**., Bilavarn, S., and Belleudy, C.: "Impact of operating points on DVFS power management," *In proceedings of 7th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS'12*, Tunisia, May 2012.

2) **Khan, J**., Bilavarn, S., and Belleudy, C.: "Energy analysis of a DVFS based power strategy on ARM platforms, " *In proceedings of FTFC 2012 : IEEE Faible Tension Faible Consommation, FTFC'12*, France, Jun 2012

3) **Khan, J.,** Bilavarn, S., Belleudy, C., and Bhatti, K.: "Energy analysis of a real-time multiprocessor control of idle states on ARM platforms," *In proceedings of Pervasive and Embedded Computing and CommunicationS PECCS'13*, Barcelona, Feb 2013.

Chapter 2 : POWER MANAGEMENT CHALLENGES

Moore's law states that the number of transistors in integrated circuits doubles approximately every two years [1]. As the technological advancement has proven this law to be correct, this decrease in size and increase in computational power impacts the power utilization of these highly complex components. Lower power consumption also means lesser heat dissipation, which increases system stability. However, this imposes major enhancements to the management of power for these new devices containing millions of transistors. The need for more complex and portable devices imposes new efforts on designing and implementing low power solutions. To increase the battery life of systems and give more reliability to the system, new techniques should be implemented to cope with power management issues. Power management challenges include considerations of overheating, energy cost as well as environmental concerns.

In the past, boot delay of a device was very long when it was switched on and people used to leave their devices turned on to avoid this delay hence wasting a lot of energy. This led to introduce new power features allowing devices not to fully turn off when unused. To reduce power utilization when a device is not active, consumer electronic devices introduced the concept of sleep or standby states. Old versions included manual handling of going from one state to another based on user inputs. However, as the processor execution speed is much higher to that of a user, relying only on the user does not provide enough power reduction. As the technological era changed, new power management schemes were introduced to tackle automatically the power management problem without the need of the user.

The power management in PCs and embedded systems has been handled historically with two different power managers i.e. Advanced Power Manager (APM) and Advance Configuration Power Interface (ACPI). Advanced Power Management (APM) represents the first stage, developed by Intel and Microsoft in 1992 [2]. Power management was performed through BIOS by turning off the display, disabling the hard disk after a preset period of idle time, or by idling CPU by entering in suspend state. APM lacked a lot of capabilities. For instance, power management was done as a background process by the BIOS instead of the Operating System (OS). For each different platform, this BIOS based APM was specific to the platform and had to be supplied by the manufacturer. The

second version named Advanced Configuration and Power Interface (ACPI) is still widely used in today's operating systems. ACPI is an open industry specification establishing a standard interface for OS direct configuration and power management of laptops, desktops, and servers. First released in December 1996, ACPI defines platform independent interfaces for hardware discovery, configuration, power management and monitoring. ACPI is superior to APM as it addresses the drawbacks of APM. Although ACPI is fully adopted in windows based platforms, however Linux support for ACPI is still evolving.

Besides these two major advancements of power managers, modern microprocessors also come with their own power management policies implemented in their BIOS. However, the evolution of these complex microprocessors does not fully compensate the energy demand and power optimization. Hence, operating systems may be limited by their general purpose software-based power management policies and could save further energy consumption for more demanding workloads like video and/or real-time processing for instance. In this work, we present a study of various power strategies on real-life applications and platforms. To do that, we investigate the conditions of energy gains of three distinct multiprocessor strategies: a Dynamic Voltage and Frequency Switching (DVFS) based strategy for video applications, a DVFS based deadline scheduling strategy and a Dynamic Power Switching (DPS) based deadline scheduling strategy. We also aim to investigate the conditions of efficiency and power reduction of these different strategies, so we promote a realistic investigation approach based on power measurements using actual multiprocessor platforms as much as possible.

We address all these issues in detail in the following sections. Section 2.1 provides an overview and discussion of existing works related to the general field of power management. An overview and classification of work are presented in section 2.1.1. In section 2.1.2 and section 2.1.3, we detail the two broadly used power management techniques, i.e. DVFS and DPS. We then move to section 2.1.4, to review market standards and their available power management solutions. We start with Intel platforms and their power management infrastructure in section 2.1.4.1. Then we cover AMD processors and their respective technologies in section 2.1.4.2. The ARM processors are famous for their low-power consumption; we thoroughly study their PM infrastructure in section 2.1.4.3. Afterwards in section 2.1.4.4, we move to the power management provided by the Linux operating system. At the end in section 2.1.4.5, a conclusion is

established on the discussion of these standards. An overview of academic research work is then shown in section 2.1.5 with a focus on low power scheduling techniques. In section 2.1.6, we provide a global conclusion about the advantages, shortcomings, as well as other possibilities of exploring the said techniques. In the end, section 2.1.7 puts light on our focus and objectives related to power management for multiprocessor platforms.

The second part of this chapter is to present our work context, why this work is carried out and how we can address the encountered shortcomings related to the power management on the representative platforms. Therefore, in section 2.2.1 we start by providing the problem statement. Afterwards, in section 2.2.2 we give information related to different platforms used in this work. We present the real platforms ARM11 MPCore and ARM1176JZF-S along with two QEMU based virtual platforms. At the end, we provide a brief introduction of various power management strategies in section 2.2.3, where the first dynamic video power strategy is described in subsection 2.2.3.1. In section 2.2.3.2, we introduce the DVFS based deadline scheduling technique and in section 2.2.3.3, we introduce the DPS based deadline scheduling technique.

2.1 State of the Art

This state-of-the-art section provides an overview of historical approaches used by systems. There are a lot of power management techniques that are proposed or implemented in embedded systems. However, power management is a large domain and cannot be summarized entirely. We start by differentiating and discussing the proposed classification of work in the field of power management. In the following sections, we focus on the two broadly used power management techniques, i.e. DVFS and DPS. The presence of a large number of processor manufacturers result in a variety of techniques, in spite of this we discuss the main approaches used by largely developed standards (ARM, Intel, AMD, etc.) and their respective platforms. We give a clear view of how these industrial and commercial available products manage power consumption. A detailed discussion on power management policies and infrastructure used by the Linux OS is also presented. The state-of-the-art section also provides an overview of ongoing research in the field of power management in academic domain. As this is a very large domain of research, we only focus on techniques used for processor level power management. We also discuss low power scheduling policies as they provide interesting

solutions for power management in modern multiprocessor devices. At the end, we provide a general analysis and conclusion of this state-of-the-art regarding the targets and goals of our work.

2.1.1 Overview and Classification of Work

In this section, we emphasize different power management techniques used in embedded systems. Power management can be done at different levels, i.e. operating system level, component level or application level. CPU power consumption is significant and is the main target of power management analysis in various studies. For example, numerous power aware models studied in [3-7] are shown and integrated into present performance simulators to investigate power consumption of the CPU, on a unit basis or for the processor as whole. Our focus is also on techniques which are related to processor level power management. Two popular techniques are mainly employed: Dynamic Power Switching (DPS) to switch off the power of a part of the circuit to idle states, and Dynamic Voltage and Frequency Scaling (DVFS) to tune a processor clock speed and voltage. In some works as in [8], DPS techniques are also referred to as Dynamic Power Management (DPM). DPM refers to an IBM and MontaVista initiative of developing a general and flexible dynamic power management architecture which is now outdated. We therefore use the term DPS instead of DPM in order to avoid confusion. In general, DVFS and DPS techniques are controlled by a power policy or power strategy. The next section provides a detailed discussion on these broadly used techniques.

2.1.2 Dynamic Power Switching (DPS)

Dynamic Power Switching (DPS) is a well-known technique that selectively shutdowns or puts to sleep some active components in embedded systems, to manage energy consumption and heat dissipation problems. In early stages, power switching was under the control of Advanced Power Management (APM). APM provided different power modes like *full-on, enable, standby, suspend,* and *off*. A brief description of these modes is given below:

In *full-on* mode all components are fully powered and there is no power management occurring. In *enable* mode CPU is slowed or stopped (depending on BIOS), all other devices still draw full power and recover time is instantaneous. In APM *standby* mode,

CPU may be stopped depending on operation or activity, most devices are in low power mode as well as monitor enters its first power management mode. An activity can trigger a return to *enable* or *full-on* mode depending on the system and activity. In APM *suspend* mode CPU is stopped, most power-managed devices are not powered (network card may stay on) and APM provides maximum power savings. Activity can trigger a return to *standby*, *enable* or *full-on* depending on the BIOS though recovery time is very large. In APM *Off* mode operational parameters are saved, system turn off and draw no power.

A decision to enter low-power modes was controlled by timers and the return to full power (or higher-power modes) is directly triggered by activity from the keyboard, mouse, modem, or network. Such activities generate interrupt requests, or IRQs, which signal the processor that it needs to respond to them. Even when a PC has powered down (but is not off), the BIOS still monitors IRQ activity. APM had a major drawback of its dependence on the BIOS. Another shortcoming was lack of intermediate power states and usually systems either uses full performance level or are always idle (inactive). ACPI addressed these issues and was later adopted and enhanced by all market sharers, and is still in use. ACPI allows direct power management by the Operating System. ACPI is a hierarchical power management technique having different system and device power states as shown in figure 2.1. ACPI naming convention divides the overall system components in set of manageable components like Global G-states, System S-states, Processor P-states, Busses B-states, Links L-states and Devices D-states. A numbering convention is also used where '0' is the main active state like G0, S0, D0, etc. The numbers from 1 to n represent the corresponding power levels. The higher number indicates the lower power consumption state in comparison to its previous ones. For example, P1 state consumes more power than P2, P3 or P4 state. ACPI subsystem uses these states to handle power management of individual devices, components or the overall system.

A summary of different states is discussed to give an idea of this dynamic power switching mechanism. Here, G-states are the top-level states for the overall system and reflect the user level perception. G-States are global operating states and are not configurable by the user. These are just used in documentation to specify certain system states such as on, off, sleeping. G0 is the working state, where the system is fully operational. G1 is a sleeping state where power consumption is small, and the system can be resumed without the need of booting. G2 is a soft off state in which the system

consumes a minimum amount of power, and the system must be restarted to be functional. G3 is a mechanical off state, where all hardware information is lost, and power consumption is zero. A hierarchical view of G-states with other system states is shown in figure 2.1.

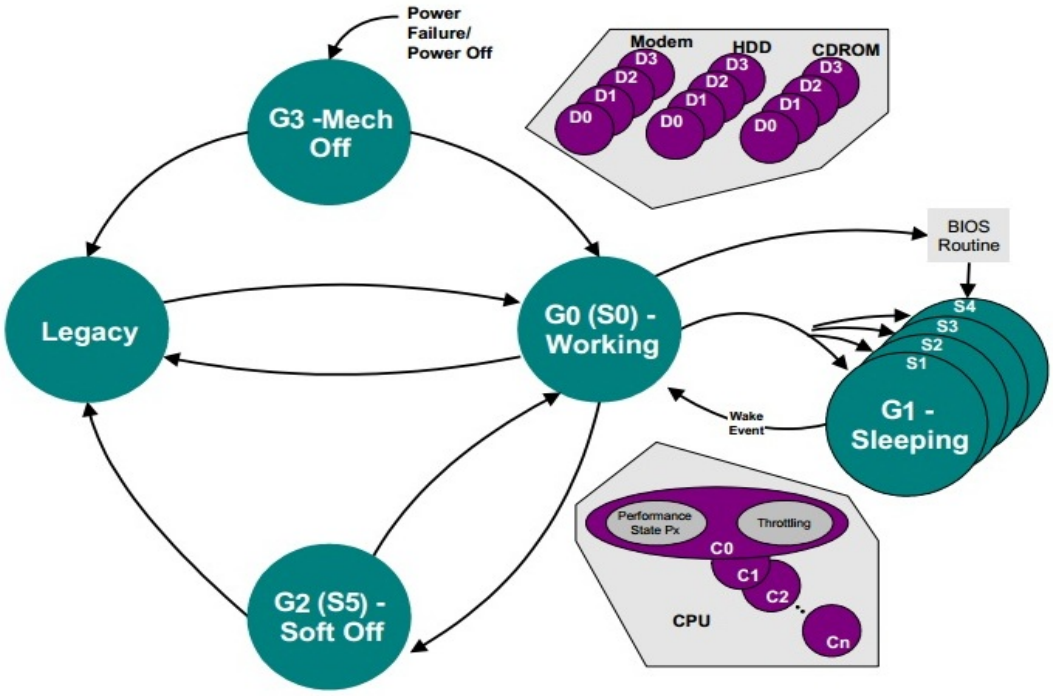

Figure 2.1: Global System Power States and Transitions. [9]

The S-states are types of sleeping states within the G1 sleeping state. These are set in the BIOS and then configured in the power option's control panel (timeout in minutes). The S-states allow the system to save a significant amount of power when not being used. S0 is the working or fully powered state. S1 and S2 are similar low wake-up latency states except that in S2, the CPU and cache context are lost. S3 is a suspend to RAM state where the memory image is maintained and powered, while the CPU chipset and I/O devices lose their content. S4 is a suspend to disk or hibernate state where all devices are powered off. This state has the longest wake-up latency. The RAM content is lost and the platform content is maintained on the hard drive. The soft-off state S5 is similar to

S4 but here, the system requires complete re-boot when waking up. A system in S-state can be triggered by a motion sensor, LAN or GPIO activity.

The CPU power state or C-states are the processor power consumption and thermal management states within the global state G0. It reduces power consumption by putting the processor to sleep when it does not have code to execute. The entry and exit from C-states are much faster in comparison to S-states. C-states require low processor power during idle light workloads. The C-states limits can be configured and set by the BIOS, and a processor can go to C-states several thousand times per second. These states can also be referred as operating, halt, sleep and deep sleep mode. The processor is in C0 state when it executes instructions. The C1 to Cn power states are processor sleeping states, where the processor consumes less power and dissipates less heat than in the C0 state. C1 is the lowest latency power state where the processor is in a non-executing state. C2 state offers improved power saving in comparison to C1. The contents of L1 cache are saved to L2 cache. C3 state also offers improved power scaling over C1 and C2, but the time latency of entrance and exit increases. In the C3 state, the core flushes the content of its L2 instruction cache and the shared L3 cache, while maintaining its architectural state. All core clocks stop at this point. The C-states are processor model specific, therefore some processors may contain further C4, C5, C6 states, etc.

While in the C0 state, ACPI allows the performance of the processor to be altered through a defined *throttling* process and through transitions into multiple performance states (P-states) as shown in figure 2.1. P0 provides maximum performance while consuming maximum power (at a higher frequency). Similarly, P1 is a state where the processor consumes less power than P0. The lesser power consumption also results in reducing the performance below the maximum. The P-states also depend upon the underlying hardware. The numbers of P-states are processor specific, where each state corresponds to a different frequency and power consumption level. Pn is the state where processor has lowest frequency and minimum power consumption. P-states are very useful to control non-critical workloads that do not require maximum frequency and power consumption throughout their execution.

ACPI also contains the device D-states (Figure 2.1) for managing power of peripheral devices. D0 is the active state and consumes maximum power. Devices often come with only on and off states. Therefore, D1 and D2 are rarely used. However, in devices like modems, D1 state provides functionality like modem controller to go into low power

mode, phone interface powered by the phone line, and speaker off. D3 is the state when a device is off. Like in case of a modem, the controller is switched off, phone interface turned off, speaker off. Additionally, it can be D3 hot or D3 cold depending on if primary power is removed or not.

The operating system uses these ACPI states with the help of a policy manager for power and energy management. An example of such a policy manager is OSPM as shown in figure 2.2. The policy manager defines a policy based on the different user, application or environmental parameters. The policy manager directs the rules for using efficiently the ACPI sleep states for the policy (or scheme). The ACPI sleep states are mostly selected by the policy manager when there is no workload on the processor. However, the processor is usually the most active element in embedded systems. Handling smartly the processor states has the potential for significant energy and power gains. Therefore, when a processor is active, the policy manager uses DVFS techniques to select the P-states for managing power, as detailed in the next section.

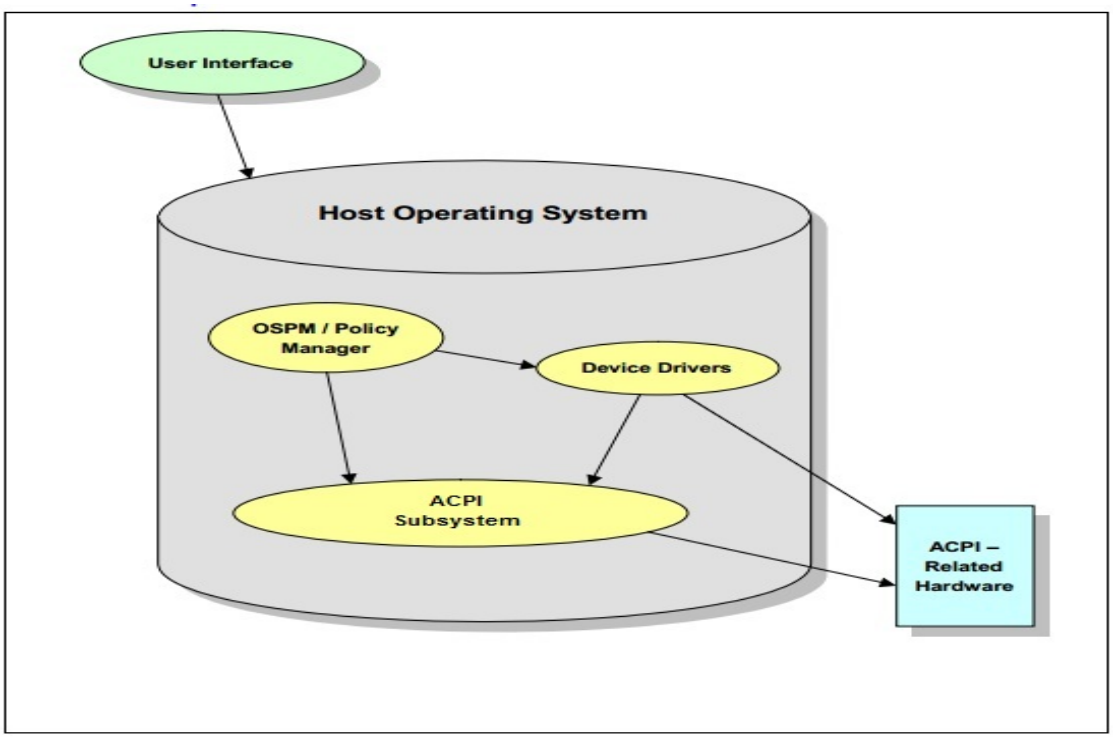

Figure 2.2: Power management model with ACPI subsystem. [9]

2.1.3 DVFS Techniques

Dynamic Voltage and Frequency Scaling (DVFS) is widely used for power management in modern processors, as it is an effective method for achieving low power consumption of CPU while meeting the performance requirements. The purpose of DVFS techniques is to scale dynamically the circuit speed and the supply voltage level of the processor, to process the system workload. The frequency and supply voltage are directly related to power consumption in CMOS technology as shown in [10]. Therefore, decreasing frequency and voltage will impact overall power consumption and will increase/decrease the total energy consumption. Modern microprocessors come with the built-in support for DVFS along with the support for DPS. A number of new microprocessors such as ARM1176JZF-S [11], ARM CortexA9 [12], AMD-K6^{TM}-2E+Processors[13], Intel M series [14], Transmeta Crusoe [15] are equipped with the DVFS functionality.

Microprocessors are also provided with power management software (policies) to make use of DVFS techniques. A detailed description of some of the industry standards, as well as their power management policies and DVFS techniques are detailed in section 2.1.4. DVFS in popular operating systems, like Windows and Linux, is supported by the help of ACPI. The ACPI C0-states can be further divided into performance P-states as described in section 2.1.2. Different policies are defined based on performance and power needs, to use these P-states efficiently. For example, the *OnDemand* policy *(*governor) in Linux allows adapting the frequency to the workload using P-states. This governor allows the CPU to achieve maximum clock frequency when the system load is high and minimum frequency when the system is idle. The detailed description of Linux governors is given in section 2.1.4.4. Similarly, Windows OS comes with policies like *Balanced*, *Max Power Saving*, *Max Performance*. These policies use the ACPI infrastructure that allows the processor to change its frequency and voltage according to the workload. For example, in case of a *Balanced* scheme, ACPI chooses the best P-state level based on total average workload. An example of P-states of an Intel M-processor using Enhanced Intel SpeedStep (EIST) technology is shown in table 2.1. The corresponding frequency and voltage levels are also shown for each P-state. According to ACPI, P0 is the state where frequency is maximum (1.6 GHz), and voltage is also maximum (1.484 V). Similarly, the lowest P-state is P5 where the frequency and voltages are at their lowest values as shown in table 2.1.

Table 2.1: Supported Performance States for the Intel M-Processor [14]

P-States	Frequency	Voltage
P0 (HFM)	1.6 GHz	1.484 V
P1	1.4 GHz	1.420 V
P2	1.2 GHz	1.276 V
P3	1.0 GHz	1.164 V
P4	800 MHz	1.036 V
P5 (LFM)	600 MHz	0.956 V

Another important aspect of implementing a DVFS based strategy (policy) is its dependency on the operating points. Operating points in the context of DVFS are the frequency/voltage levels that a processor support (equivalent to P-states shown in table 2.1). DVFS based policies should be defined in a way to use these operating points efficiently. For instance, if the desired frequency for an application or workload is between two discrete levels, the one with higher value should be used. In this way, we can guarantee the temporal requirements. Researchers in [16], have proposed algorithms to map such required continuous frequency/voltage levels to discrete values, to utilize DVFS technology. Thanks to the advancements in micro architectures, modern processors are able to operate in a wide range of the frequency/voltage spectrum. This allows DVFS based policies to exploit a wide range of available P-states. For example, Intel SpeedStep technology evolved from changing frequency by taking steps of 200 MHz in its first version (*SpeedStep* I, Table 2.1), to steps of 100 MHz in the latest version (SpeedStep III).

2.1.4 Available Market Standards

Traditionally, first technologies in power management were based on dynamically switching on and off the circuitry or components to achieve an overall lower power utilization [17]. Power management for computer systems has focused on regulating the power consumption in static modes (such as sleep, suspend). These de-activating states often require a user action to re-activate the system. Latencies and overheads are usually significant for entering and exiting of these states. In desktop and server systems, a firmware layer is typically added to support these modes. However, many architectures

provide the equivalent of a halt instruction that reduces CPU power during idle periods, with only a few clock cycles of latency.

With the advancement in technology and constraints such as size, cost and environmental effects, no design is complete without a thorough analysis of the power-supply architecture. Modern systems are therefore equipped with facilities such as dynamically changing the frequency and voltage, flexible power management modes, separate power domains, intelligent voltage regulators, etc. New micro architectures are designed with CMOS technology to consume less power. Power consumption at the processor level is generally divided into static and dynamic power. Accordingly, we have to decrease both the static and dynamic power to decrease the total power consumption. In the past, static power consumption has been small in comparison to dynamic power. However, factors contributing to leakage power, including quantum effects (such as gate-oxide layer tunneling) are becoming increasingly important with shrinking silicon feature sizes as discussed in [18].

Voltage has a quadratic relationship with power for dynamic power consumption, therefore decreasing voltage can significantly affect dynamic power utilization. With the availability of techniques like DVFS, new policies and power management methods are possible at different levels (i.e. OS, Hardware, etc.). ACPI is widely used to handle power management as detailed in the above section 2.1.2. Besides ACPI, modern microprocessors come with their own power management hardware and software to handle power consumption. Examples of such processors include Intel *SpeedStep*[14], AMD *Cool'n'Quiet* [19] and *PowerNow* [20], IBM *EnergyScale* [21] and ARM *IEM* [22]. A brief overview of power management policies used by Intel, AMD and ARM processors, as well as power management in Linux is detailed in the following sections.

2.1.4.1 Intel

There are several Intel® power management technologies that can be used by embedded developers to manage the balance between power consumption and performance. First, there are power states that define distinct “sleep” modes as well as different fully functional operating modes. Second, Enhanced Intel *SpeedStep*® Technology enables optimal performance at the lowest power by allowing the operating system to change the processor frequency and supply voltage. Third, Intel® Turbo Boost Technology provides additional processor frequency bins, above the base operating frequency (i.e. faster). An

embedded system only runs at the full throttle when workload demand is high, therefore, energy is mostly saved during non-peak times. Intel power management technologies give software developers granular control over the system operation.

Intel *SpeedStep*, that is a trademark for a series of dynamic frequency scaling technologies (including *SpeedStep* I, *SpeedStep* II, and *SpeedStep* III) allow the clock speed of the processor to be changed dynamically. This allows the processor to meet the performance needs for the operation being performed, while minimizing power drawn and heat dissipation. Power management in new Intel series processors, is provided by the built in *Enhanced Intel SpeedStep (EIST) technology*. *EIST* allows the processor performance and power consumption levels to be modified while a system is functioning. This is accomplished via application software, which changes the bus-to-core frequency ratio and the processor core voltage (Vcc). A variety of inputs such as the system power source, processor thermal state, or OS power policy is used to determine the proper operating state.

There are various versions of EIST present for Intel processors. The EIST V1.1 is used by second generation Pentium III processors. It enables the CPU to switch between predefined modes: the top and bottom modes are commonly known as high-frequency mode (HFM) and low-frequency mode (LFM). The frequencies and voltages (operating points) are stored within a read-only processor model specific register (MSR). This MSR ensures BIOS will not allow transitions to invalid states above the HFM or below the LFM. The other four operating points are stored within the BIOS code in a drop voltage table provided by vendors. An example of P-states for Intel M processor at 1.6 GHz is shown in table 2.1 in section 2.1.3, where the processor frequency is modeled to have steps of 200 MHz. Using EIST, Pentium III processors consume about 20 Watts at 1 GHz and it can be reduced to 6 Watts at 600 MHz. EIST V2.1 (Enhanced *SpeedStep*) is used in Pentium III-Mobile processors and is similar to the previous version. EIST V2.2 is adapted for Pentium 4-Mobile processors. With EIST V2.2, a 1.8 GHz Pentium 4-M consuming about 30 Watts can lower its frequency to 1.2 GHz, thus reducing power consumption to about 20 Watts. EIST V3.1 is used in the first and second generation of Pentium M processors (Banias and Dothan cores, used in Centrino platforms). With this technology, the CPU varies its frequency (and voltage) between about 40% and 100% of its base frequency in increments of 100 MHz (for Banias core) or 133 MHz (for Dothan core).

2.1.4.2 AMD

AMD provides several power management technologies that can be used by software developers to deal with power and thermal issues. To manage power at the CPU level, the two most efficient technological standards provided by AMD are *Cool'n'Quiet* and *PowerNow*. *Cool'n'Quiet* is a CPU speed throttling and power saving technology introduced in the *Athlon64* processor line. The technology reduces the overall power consumption and lower heat generation, allowing for slower cooling fan operation. The objectives of cooler and quieter system execution result in the name *Cool'n'Quiet*. The technology is similar to Intel *SpeedStep* and AMD *PowerNow*. Due to its different usage, *Cool'n'Quiet* refers to the desktop and server chips, while *PowerNow* is used for laptops and mobile chips. The technology is also introduced on *e-stepping* Opterons; however, it is called Optimized Power Management (*OPM*), which is a re-tooled *Cool'n'Quiet* scheme designed to work with registered memory.

Cool'n'Quiet technology controls the processor performance automatically by dynamically adjusting the operating frequency and voltage up to 30 times per second to the task at hand. It reduces the power in two ways. First, it reduces the leakage power at idle. The *Athlon64* puts itself to sleep when the HLT/STPGNT instruction is sent. At this stage, only the leakage power is drawn. The power is further cut down significantly by reducing the frequency and voltage. Secondly, it reduces the power needed during light and medium load. When the processor can cope with the work at a low frequency/voltage level, it stays at this level. The performance penalty of this solution is negligible. All *Athlon64* and all *A64* based *Semprons* greater than 1.8 GHz support this feature. Power can be saved significantly when an application does not require full performance. The processor can also respond to increased workloads, allowing the system to deliver a responsive computing experience.

Like *Cool'n'Quiet*, AMD *PowerNow* is also a speed throttling and power saving technology of AMD processors used in laptops. All *AMD-K6-2E+* and *AMD-K6-IIIE+* low power embedded processors support *PowerNow*. The processor clock speed and core voltages are automatically decreased (based on power policy) when the computer is under low load or idle, to save battery power and to reduce heat (noise). The AMD *PowerNow* technology supports a wide range of operating voltages ranging from 0.0925 V to 2 V (allowing 32 different core voltages with a step as small as 25 mV or 50 mV). The technology also supports frequency starting from as low as 133 MHz or

200 MHz (allowing steps of 33 MHz or 50 MHz) depending on the processor model. Table 2.2 shows an example voltage and frequency levels using *PowerNow* technology. The allowed voltage step is 0.1 volts with the respective frequency switching of 50 MHz.

Table 2.2: Supported Voltages and Frequencies for Low Power AMD-K6™2E+Processors[20]

Processor	Core Voltage	Range of Supported Operating Frequencies	Active Power
AMD-K6-2E+/450APZ	1.7 V	450–200 MHz	8.70–4.90 W
	1.6 V	400–200 MHz	6.90–4.20 W
	1.5 V	350–200 MHz	5.60–3.70 W
	1.4 V	300–200 MHz	4.30–2.95 W
AMD-K6-2E+/400xTZ	1.6 V	400–200 MHz	6.90–4.20 W
	1.5 V	350–200 MHz	5.60–3.70 W
	1.4 V	300–200 MHz	4.30–2.95 W
AMD-K6-2E+/350xUZ	1.5 V	350–200 MHz	5.60–3.70 W
	1.4 V	300–200 MHz	4.30–2.95 W

Beside DVFS support, *PowerNow* technology provides different operating modes like *High performance, Power saver* and *Automatic* mode defined for specific performance and power requirements. The AMD *PowerNow* technology contains *Enhanced Power Management (EPM)* Block that can be accessed by the OS to change operating frequency and voltages. The OS based power management policies are responsible for carrying the burden of invoking EPM block. However, locking of processor from external interrupts, snoops, etc. during transition is handled automatically.

AMD has also provided other power management technologies like *AMD PowerCap Manager* that allows IT data center managers to fix power consumption on server processors. Similarly, *Advanced Platform Management Link* (APML) technology provides new controls to monitor power and cooling. Detailed description of these technologies can be seen in [13].

2.1.4.3 ARM

ARM is very well known for developing low-power processor IP technology. The ARM power management kit contains a collection of standard cells specifically designed to allow the implementation of various low power techniques. Almost all ARM processors

contain support for low power states (idle states); additionally, cores like ARM1176JZF-S, CortexA9, CortexA15, etc. also contain support for DVFS. To allow DPS, the typical ARM processor allows support for four sleep states which are *Active* or *Run*, *Standby*, *Dormant* and *Shutdown*. The processor is in *Run* mode when everything is clocked and powered up. In a *Standby* state, the CPU clock is stopped by executing a *Wait For Interrupt (WFI)* instruction. *WFI Standby* mode disables most of the clocks in a processor, while keeping its logic powered up. This reduces the power drawn to the static leakage current, leaving a tiny clock power overhead requirement to enable the device to wake up. The *Dormant* state enables the processor to be powered down while leaving the caches powered up and maintaining their state. The *Shutdown* state powers down the entire device and all states, including the cache content which must be saved by an external software. The Power Management Controller (PMC) determines whether the processor should be put into *Active, Standby, Dormant,* or *Shutdown* state. However, the choice to use these states is controlled by a policy manager. For an example, Linux OS with the help of ACPI infrastructure contains a *CPUIdle* driver, to direct PMC to use one of the predefined low power states. The PMC informs the processor about the nature of reset that has occurred. Whether the reset occurred due to exiting *Dormant* mode or *Shutdow* mode, the processor is branched to the correct state restore routine.

To allow DVFS when in active state, ARM offers a system-level power management scheme called *Intelligent Energy Management* (*IEM*) for energy and power management. This is a combination of hardware and software technology that allows DVFS to reduce energy consumption. The *IEM* solution is designed primarily for battery-powered equipment, where the requirement is to have a long battery life. The *IEM* solution is also ideal for portable devices, for example, smart phones, feature phones, Personal Digital Assistants (PDA), handheld game consoles and media players, etc. A typical *IEM* System on Chip (SoC) solution is shown in figure 2.3. A complete *IEM* solution is made up of a number of hardware and software components. The *IEM* software component uses information (based on workload) from the operating system to build up a historical view of the application execution on the system. A number of different software algorithms are applied to classify the types of activity and to analyze their processor utilization patterns. The results of each analysis are combined to make a global prediction about the future performance requirement for the system. This prediction is used by the *Intelligent Energy Controller* IEC.

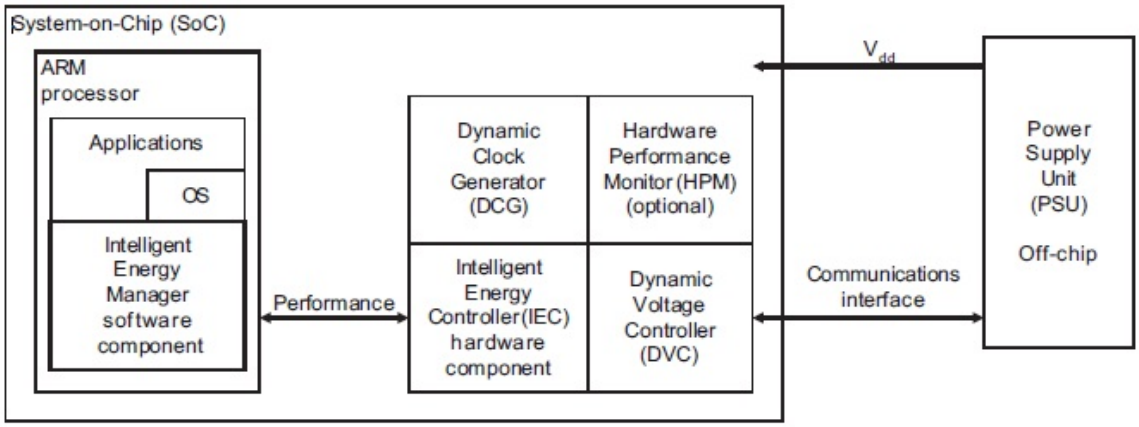

Figure 2.3: Intelligent Energy Management Solution by ARM. [22]

The IEC uses prediction performance level requests from the *IEM* software. The performance setting is communicated to the IEC, so that the platform scaling hardware can be controlled to bring the system to the required performance level. Battery life is extended by lowering the operating frequency and voltage of SoC components (such as the processor) which consequently reduces energy consumption. The IEC also measures the work done in the system to ensure that the software deadlines are not going to be missed. Additionally, the IEC also supports a maximum performance request feature. Further information regarding *IEM* and IEC can be seen in [22] and [23].

In the following section, we discuss the power management provided by Linux, to further understand how efficiently power can be saved. In addition, we also provide a discussion of academic works on power strategies, to highlight other opportunities for power management than workload based approaches.

2.1.4.4 Linux

Besides available market standards and their power management solutions, popular operating systems like Linux, Windows, Mac OS also provide support for better power solutions to deal with rapid technological advancements. Linux is an open-source operating system that is widely used in modern devices. Linux supports two implementations of power management: Advanced Power Management (APM) and Advanced Configuration and Power Interface (ACPI). Detailed description of processor sleep states available in APM and ACPI is presented in section 2.1.2. In the following, we focus on the mechanisms and strategies used for proper exploitation of these states in Linux. At system level, ACPI defines mechanisms for putting the computer as a whole

in and out of system sleeping states. It also provides a general mechanism for any device to wake the computer. ACPI tables also describe motherboard devices, their power states, the power plans the devices are connected to, and controls for putting devices into different power states. This enables the OS to put devices into low-power states based on application overall power usage. According to the description of the ACPI states provided in section 2.1.2, each state introduces greater power savings but requires commensurately more time to awaken and begin performing work. Linux device drivers are generally responsible for saving device states before putting and restoring the device in low power modes. Generally, applications are not involved in power management state transitions.

Linux contains an infrastructure *pm_dev* to maintain information about every power management event. Device drivers must have to register themselves with Linux power management subsystem. They do this by calling *pm_register* that contains necessary device information like its type, identity (ID) and functionality. The Linux *pm.h* file contains various types and IDs used by these drivers. When a power management event occurs, arguments like device name, device state, device data, etc. are passed to device drivers to perform specific tasks. For example, in case the device is a liquid crystal display (LCD), an event could be *pm_suspend* to save the device state and turn off the display. Similarly, if a *pm_resume* event occurs, the LCD driver should restore the saved state back. When the device is not used anymore, its driver should unregister itself from the power management infrastructure. When a device is unregistered, the power management does not involve that device in future power management events.

At CPU level, power is controlled by using processor's idle power states (C-states) or by changing the CPU frequency (P-states). The number of C-states and P-states depend upon processor and can vary independently. These are typically implemented with the help of ACPI as described in section 2.1.2. However, special infrastructures are present to make use of these idle and performance states. The *CPUIdle* subsystem provides the functionality of separating the layers to make use of C-states. The *CPUIdle* drivers are found in architecture-specific ACPI code. On the other hand, the decision of choosing which idle state is decided by the policy (power strategy). Linux contains *CPUIdle governors* that allow the implementation of different policies for distinct needs. It should be noted that deeper sleep (C-states) saves more power, but the downside is that they have higher latency (the time the CPU needs to go back to C0).

Processor in operating state (C0 state) can be in one of different P-states. The *CPUFreq* is used to set a static or dynamic power policy for the system. It uses the *CPUFreq* driver to dynamically scale the processor frequencies at runtime. The *CPUFreq* provides a common interface to the various low-level technologies and high-level policies. The Linux in-kernel *governors* (policy governors that can change the CPU frequency based on different criteria), and CPU specific drivers (*CPUFreq*) are used to implement the technology for the specific processor. The processor consumes less power while still doing work, and the tradeoff comes between power and performance, rather than power and latency. Each governor has its own unique behavior, purpose, and suitability in terms of workload. Generally, there are five different types of *CPUFreq* governors, i.e. *Performance*, *Powersave*, *OnDemand*, *Conservative* and *Userspace.* A brief description of these governors is given below:

Performance

The *Performance* governor forces the CPU to use the highest possible clock frequency. This frequency is statically set and will not change. As such, this particular governor offers no power saving benefit. It is usually suitable for hours of heavy workload, and only during times wherein the CPU is rarely (or never) idle.

Powersave

The *Powersave* governor forces the CPU to use the lowest possible clock frequency. This frequency is statically set and will not change. By itself, this specific governor offers maximum power savings, but at the cost of the lowest CPU performance. The *Powersave* governor is more a “speed limiter” for CPU than a “power saver”. It is most useful in systems and environments where overheating can be a problem.

OnDemand

The *OnDemand* governor is a dynamic governor that allows the CPU to achieve maximum clock frequency when the system load is high, and also minimum clock frequency when the system is idle. The *OnDemand* governor uses parameters such as the *sampling-rate*, *up-threshold*, *sampling-down-frequency*, *ignore-nice-load*, etc. to make the frequency switch decision. As an illustration, the *sampling-rate* is used to let the kernel decide how often (in microseconds) a transition is needed. *Up-threshold* is the average CPU usage (CPU %) during sampling time. Similarly, other parameters have

their own special functionality. While the *OnDemand* governor allows the system to adjust power consumption with respect to system load, it does so at the expense of latency related to frequency switching. For most systems, the *OnDemand* governor can provide the best compromise among heat emission, power consumption, performance, and manageability. When the system is only busy at specific times of the day, the *OnDemand* governor will automatically switch between maximum and minimum frequency depending on the load without any further intervention.

Conservative

Like the *OnDemand* governor, the *Conservative* governor also adjusts the clock frequency according to usage. However, while the *OnDemand* governor does so in a more aggressive manner (that is from maximum to minimum and back), the *Conservative* governor switches between frequencies more gracefully. This means that the *Conservative* governor will adjust to a clock frequency that it deems fitting for the load, rather than jumping to max speed the moment there is any load. A parameter called *freq_step* is used to define frequency steps to smoothly increase or decrease frequency. Conservative governor can possibly provide significant savings in power consumption however; it does so at an ever greater latency than the *OnDemand* governor.

Userspace

The *Userspace* governor allows user programs (or any process running as root) to set the frequency. This governor is normally used in conjunction with the cpuspeed daemon. A parameter called *Scaling_setspeed* is used to write desired frequency. Consequently, *Userspace* is the most customizable of all the governors and depending on how it is used, it can actually provide a dynamic balance between performance and power consumption for the system.

The above discussion summarizes our discussion on available technologies, market standards and power management done in Linux OS. It also puts light on specific power management techniques provided by different manufacturers. A conclusion highlighting the benefits, drawbacks and limitations of these standards is given in the next section.

2.1.4.5 Conclusion

Based on the above discussion, power management can be broadly divided into two categories. Firstly, OS based power management policies such as in Linux and secondly vendor specific power management policies like Intel's *EIST* or ARM's *IEM*. In both cases, such existing power management policies rely on the total workload of a system (which is more related to the hardware state), during a certain period to deal with power issues. These policies present the advantage of being applicable in all cases (general purpose), but the drawback is probably a certain level of inefficiency (since they do not really consider application knowledge).

Therefore, new power management solutions should be provided to overcome these shortcomings. The idea is to make a tradeoff between power and performance through power-aware algorithms. Currently, a lot of academic research is being done on accurately predicting the workload during a certain time period or to exploit other paradigms (e.g. Low power scheduling). Researchers are continuously exploring new techniques (based on application properties) to use the underlying hardware efficiently. Similarly, as some OS based policies give the user freedom to create its own custom defined policy, novel parameters (other than workload) should be used to define such custom policies, in order to further optimize energy and power consumption. The next section presents academic research on power strategies that have room for higher power savings.

2.1.5 Academic Research

2.1.5.1 Overview of Academic Research

Researchers have focused on how to estimate and minimize power in modern systems at various levels of design. At the system level, dynamic power switching (DPS) can be efficiently used to put active components in shutdown or low power states to conserve energy. The fundamental principle for the applicability of DPS is that systems (and their components) experience non-uniform workloads during operation time. Due to this distinct behavior of workload, DPS techniques exhibit different behavior. A survey of various DPS based strategies is shown in [17], where a system-level approach is used to manage power. A dynamic power strategy (or policy) is used to control when and which

of the low-power states to apply. Existing policies are usually based on workload predictions and a detailed comparison of the different algorithms has been provided in the survey. From this survey and others like in [24], dynamic power policies are categorized into two broad categories i.e. *predictive* and *stochastic* schemes. *Predictive* schemes typically exploit temporal correlation between the past history of the workload and it's near future, to predict the upcoming workloads. *Stochastic* techniques model the workload behavior as a controlled Markov process and find the optimal power management based on the model. *Predictive* techniques may cause extra penalties like power loss in transition, wake-up state energy consumption, etc. when the workload is varying widely. However, *stochastic* approaches have inaccuracies in modeling workload variations and complexity involved in solving the optimization problem.

These issues primarily limit the use of both schemes for systems where the workload is either very ir-regular or the workload model is not known a priori. Some advanced history based heuristics and *stochastic* schemes have been shown in [25], to predict varying workloads with high accuracy. However, their computational complexity severely limits their use and may not be suitable for applications having huge workload revisions. Besides system level dynamic power switching approach, another way to reduce power consumption is the use of application knowledge because the application knows the most about its future workload. An example of such work is shown in [26], where a limited application knowledge such as the size and type of the frames is used for applying DPS. These above methods have their advantages and are simple to design. However, relying purely on policies that use only low power states can be efficient but not always optimal or applicable. Therefore, researchers started to explore new methods by utilizing the support of dynamic voltage and frequency scaling (DVFS).

A lot of work has been carried out for implementing DVFS based policies at the processor level [16, 27-34]. Generally, these algorithms can be classified as static or dynamic algorithms. Static algorithms have a complete knowledge of the task set and its constraints, such as deadlines, computation times, precedence constraints, and future release times. Frequency is statically decided before the execution of a task set. We can obtain a single processor frequency that never changes, or obtain variable frequencies that are statically decided before execution of a task set. An example of such an algorithm is shown in [16], where Sys-Clock and PM-Clock algorithms determine the frequency needed to complete the task set. The algorithm determines single clock frequency at admission control and keep it constant until the taskset changes. Static

algorithms are based on the assumption that processor speed can vary continuously in a given range. In practical world, processors only have discrete frequency levels so choosing a higher-frequency level than calculated may result in granting performance, but it may result in extra power consumption. Some static algorithms use task parameters as selection criteria to obtain the needed frequency. Given a set of periodic tasks, the sequence of frequency changes to be performed on the processor during execution can be calculated offline. The task schedule is periodic and the voltage schedule computed by this way is also periodic and can be stored in a table. An example of such an assignment is shown in [35], where distinct speed is assigned to different tasks, based on the voltage schedule table obtained by offline scheduling. However, there is a major drawback of static assignment of frequency. If an activation of a task is missed, the whole frequency assignment is affected and may become inefficient.

Dynamic algorithms, on the other hand, have knowledge of current active tasks execution. The new task activations are not known to the algorithm when it is scheduling the present task set. Therefore, the task schedule changes over time. The reclamation of slack time resulting from the early completion of the current task set can be used to reduce the processor frequency, for the execution of the following tasks. For example, in [16] the authors propose an approach whereby each task is assigned a different speed. After a regular interval, a critical interval is defined, which is the maximum of all idle intervals. The tasks are scheduled at certain speeds by an EDF algorithm [36] which calculates the new deadlines and executes tasks at the new reduced speeds. Similarly, in [37], the algorithms rely on past and future predictions of idle periods, and accordingly, task's execution is stretched to lower speed. Dynamic techniques are often related to low power scheduling techniques, as they have a direct impact on the ordering and the execution of tasks. The following section describes more in detail such low power scheduling techniques

2.1.5.2 Low Power Scheduling

Scheduling is used in embedded devices to balance the load effectively or to achieve a desired quality of service. While scheduling tasks, processes or jobs, the scheduler mainly focuses on some core issues like turnaround time, response time, waiting time, deadlines, periods, etc. Low power scheduling refers to the scheduling of tasks keeping in view the power utilization. Different approaches are used for low power scheduling at

CPU level in real time systems. However, when one of the core issues is addressed others may be compromised. Therefore, an off-line scheduling analysis should always be done, regardless of whether the final runtime scheduling algorithm is static or dynamic. Mostly for some available resources, offline scheduling refers to identifying the maximum set of tasks to be scheduled with their worst-case execution time and applying a static scheduling algorithm to produce a fixed schedule. This schedule is used online with well understood properties. Tasks may be given preference based on operating system needs, application requirements or user needs. However, the timing constraints are the most important factor for all real time systems. Therefore, based on the extent of time importance, real time systems can be divided into two major classes. Real-time systems with hard timing constraints are known as hard real-time systems, whereas those with soft timing constraints are known as soft real-time systems.

There have been various studies on low power scheduling in real time systems [38-42]. Some authors categorize real time scheduling based on the clock driven, event driven or a hybrid of both [40]. Clock-driven schedulers are those in which the scheduling points are determined by the interrupts received from a clock. In event-driven schedulers, the scheduling points are defined by certain events, which preclude clock interrupts. Hybrid driven schedulers uses both clock interrupt as well as the event occurrence to define their scheduling points. Clock driven schedulers have an advantage of lower overhead; though they are not feasible to be used for tasks having non uniform periods (i.e. aperiodic or sporadic tasks). Event driven schedulers are more flexible than clock driven schedulers, as they can also handle aperiodic or sporadic tasks. However, a shortcoming of such algorithms is their complexity and task switching overhead. Scheduling algorithms can be further divided into *fixed priority* or *dynamic priority* algorithms. In a *fixed priority* algorithm, all jobs generated by a specific task have the same priority throughout their execution. An example of such an algorithm is the rate monotonic (RM) algorithm [38]. RM is a statically defined *fixed priority* algorithm that assigns priorities to tasks based on their rate of occurrence. It is known to be good among the *fixed priority* based algorithms for single processor platforms. A rate monotonic (RM) scheduler can be implemented simply by assigning each task a fixed priority level which is inversely proportional to its period. In a *dynamic priority* based algorithm, jobs generated by the same task may have different priorities (based on different parameters). Earliest Deadline First (EDF) is a well-known example of dynamic priority based algorithm [39]. However, implementing a dynamic scheme, like EDF, requires to keep track of all

deadlines and to perform an online mapping between absolute deadlines and priorities. When a scheduling event occurs, the task with the smallest deadline is taken to be executed. A scheduling test is also performed offline to check whether the overall tasks are schedulable under EDF or not. EDF has proven to be an optimal single processor scheduling algorithm by [43]. For a large number of tasks, EDF has a clear edge with respect to RM algorithm. The EDF and RM algorithms are extensively investigated for low power management of modern processors [43]. However, their use of power management is generally limited to single processor systems.

Some authors use combinations of both static and dynamic algorithms as in [31-34, 44]. These algorithms can be further divided into an inter-task [31, 32, 45] and intra-task [33, 34, 44] scheduling techniques. Inter-task scheduling algorithm redistributes the dynamic slack time at task boundaries to other tasks. Conversely, the available slack time is reallocated inside the same task in the intra-task scheduling algorithms. However, the most important step in implementing these strategies is the prediction of the future workload. This workload prediction is used to choose the minimum frequency (thus power) while satisfying performance. The intra-tasks strategies are difficult to manage because of their constant tracking, extreme analysis and sometimes the need of application source code modification. The number of excessive frequency switching may also decrease their efficiency. In contrast, the downside of the inter-task approach is that the interval boundaries are predetermined and independent of workload changes. Thus, they can be late in responding to large, severe activity swings.

Another important aspect of low power scheduling in modern systems is the applicability to multiprocessor platforms. Current microprocessors come with multiple cores, having variable frequency and voltage supports. Scheduling in multiprocessor environments is not an easy task, and this area has been an active topic of research in recent years. Some authors divide the multiprocessor scheduling problem into two classes i.e. *partitioned scheduling* and *global scheduling* [46, 47]. *Partitioned scheduling* assigns each task set to a specific processor, and a processor can only execute the tasks that are assigned to it. After partitioning, single processor scheduling algorithms (such as EDF, RM, etc.) can be employed to each processor. In *global scheduling,* all prepared tasks are put into a shared queue, and each idle processor fetches the next highest priority ready task from this queue for execution. It has been shown in [47], that simple global EDF or global RM could fail to schedule tasks having small execution time. In

[46], the authors use an extension of the RM algorithm, called RM first fit (RMFF), that can schedule any system of periodic tasks with total utilization bounded by $m(2^{1/2}-1)$, where m is the number of processors in the system. They also show a better bound to be $(m+1)\ (2^{1/(m+1)}-1)$ for RMFF in [47]. In addition, various other algorithms have been implemented to schedule tasks efficiently, and to reduce power consumption in multiprocessor systems [46-51]. However, the authors in [46, 47] have shown that there is no optimal online scheduling algorithm for multiprocessor systems. An extension of EDF is also shown in [52] for a multi processor environment, where authors agree that EDF is not the optimal algorithm for a multiprocessor environment. Even so, they still provide compelling reasons for the use of the EDF algorithm in a multiprocessor environment that are:

- The EDF scheduling algorithm has the least runtime complexity among job-level fixed-priority algorithms for scheduling tasks on multiprocessor architecture.
- EDF is an optimal scheduling algorithm for single-processor systems.
- EDF is considered as efficient from an implementation point of view.

The work presented in this thesis will also be considering such EDF based scheduling, as it seems to be a good basis for multiprocessor low power scheduling. The policies described above claim to be efficient, however if we explore the implementation results, we come to the conclusion that a very few are practically implemented in real development world. The work is validated using specific high level simulation tools, but not with actual platforms. Therefore the actual applicability of these works specially in a multiprocessor environment remains questionable.

2.1.6 Conclusion

Power management plays an increasing role in modern electronic devices. Much progress has been made on the technical side, for instance, latencies to switch on/off a device or to change a processor frequency have been significantly reduced. At the processor level, the availability of multiple voltage and frequencies has been really helpful in decreasing power consumption extensively. Similarly, many power management strategies have been proposed, both by the academia and industry, to manage the available technology more efficiently in order to decrease overall energy consumption.

However, there are certain limitations in implementing different power management strategies in practice. From a hardware point of view, modern processors have a support for multiple voltage and frequencies. Still, this number of discrete available frequencies and the corresponding voltages do not allow deployment of policies that use continuous frequency voltage mapping. Moreover, frequencies and related voltages vary for different platforms. For instance, an AMD K6-III uses 1.4 V for 300 MHz, whereas, an Intel M-processor uses 1.4 V for 1.4 GHz. Therefore, some power management solutions may work for a particular platform but not for others. Furthermore, existing vendor provided solutions are mostly workload based and may only work efficiently for applications having workload variations that are not too fast. However, we will see that there is still room for power savings using application or domain-specific strategies. Another important issue regarding vendor particular power management strategies is that they are non customizable. There is a very little room in modern platforms to define application or domain-specific policies.

From a research point of view, a lot of theoretical works with formal proof or simulation based verification has been done. However, there is very little work on implementing the proposed techniques on real platforms because of the complexity and development time implied. Consequently, a natural question arise that "Does the approach really works in a real world"? There are very few works that respond to this question by actually implementing the proposed techniques. The reason behind this is the complexity to develop low power strategies, in particular, deadline schedulers. In many works, the proposed strategies or policies often make simplified assumptions. They generally neglect certain important real implementation issues like neglecting the effect of state transition delay, limited number of available frequencies and power levels, continuous frequency values, etc.

2.1.7 Focus and Objectives

The work presented in this PhD is a study of low power management techniques on actual multiprocessor platforms. The work focuses on analyzing different power management strategies in both single and multiprocessor platforms. The approach shown in this work is built on experimentation, based on representative platforms (real or virtual), rather than most works that rarely addresses the problem of power management efficiency. The main goal is to investigate if and how power management strategies can

be efficiently exploited in real platforms. This is based on exploring the behavior of different parameters such as effect of supported operating points (frequencies and voltage couples), scheduling and transition delays, idle and load power, etc. of actual platforms. Similarly, as discussed above in section 2.1.2 and section 2.1.3, DPS and DVFS are two popular techniques used for power management in modern architectures. The management of these mechanisms through an operating system requires software (policy) that controls efficiently the behavior of the platform according to application requirements or processor parameters. This work focuses on experimenting with such policies (based on DVFS and DPS techniques) and explores the conditions at which a certain policy is effective in terms of energy savings.

A first point is to experiment with full implementations of low power schedulers on real platforms, and this will be based on the method described in [53]. Another problem of investigating power management policies in real platforms is the non-availability of power measurement and monitoring methods. Reliable power monitoring is a complex issue, especially for multiprocessor platforms supporting power management, and should be handled precisely. Mostly, the platforms provide methods for estimating the total energy (or total power) consumption and normally could not handle the processor energy consumption. The reason is the complexity of measurement of power consumed by the core at extremely low voltage and very fast frequency switching. This study is targeted to measure, monitor and handle the energy consumption purely related to the processor. The platforms used (ARM1176JZF-S, QEMU) have been specifically chosen to provide such access to the power consumption of cores.

In this work, we address different policies for energy management on ARM based platforms. We assess the behavior of these policies in multiprocessor platform as well. We provide a comparative analysis of a few DVFS, and DPS based strategies on different ARM platforms, in distinct configuration of active cores, operating points, application parameters, etc. At the end, we provide a global analysis result and discussion of the measures, and conclude on the energy efficiency of these power strategies.

2.2 Work Context

The work presented in this thesis is part of a European project named COMCAS (COmmunication-centric heterogeneous Multi-Core Architectures). COMCAS is a cross-industry research and development project consisting of many organizations, including *LEAT* as a university partner from France. The project is labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics. The COMCAS project aims at a breakthrough low-power design solutions for (data) communication-centric heterogeneous multi core architectures targeting 45 nm and 32 nm CMOS technologies. These architectures will be exploited in a number of future applications (i.e. the next generation of programmable multi-processor mobile phones and mobile digital entertainment devices). COMCAS investigations concern the complete low-power design hierarchy looking at all aspects from system-level choices, modeling of applications (algorithms, protocols) and architectures, the maximization of reuse of existing IPs using the most appropriate tool chains and minimal power design in technologies of 45 nm and beyond.

The topic on which our team is assigned to work is the processor level energy and power management. We explore different energy and power management policies for ARM based platforms. We have been working in close collaboration with the project partners: TIMA labs [8], that provided the virtual platforms used in this work for evaluation of power strategies, Thales Communication France and CEA LETI for applications. The new technologies combined with various lower power techniques developed in the COMCAS project allow for the design of smaller form factor devices capable of delivering HD video, high-resolution imaging, desktop-class 3D graphics. The work exposed here focuses specifically on software-based techniques to reduce the power consumption of multicore platforms. Therefore, we investigate DVFS and DPS based power strategies for multiprocessor ARM based platforms and their actual efficiency in improving the energy consumption. At the end of the project, two demonstrations have been made to show the actual achievements of this work within the goals of COMCAS.

In the next section, we start by defining the problem statement of this study and define the goals of these experimentations with different power management strategies.

2.2.1 Problem Statement

Most previously mentioned industrial techniques for power management rely on performance needs. The decision to adapt a low-power state or to change frequency and voltage are usually taken based on the overall workload in existing power management. When the workload needs higher performance, the system consumes maximum power. Similarly, if performance demand is low, power consumption is decreased. Likewise, vendor specific policies are for the most part general purpose, predefined and non customizable. Even though a user can choose a given policy, the infrastructure does not allow user defined power management for application-specific strategies. Therefore, users can only rely on the power management policies provided by their vendors. They are usually quite efficient for general-purpose applications and usage, but they can be inefficient for other specific applications. Likewise, power management approaches used by the operating systems are mostly workloads based. For example, the static policies provided by Linux governors (i.e. *Performance, Powersave)* set the frequency to maximum or minimum according to the workload, i.e. based on the processor activity. Consequently, when the system uses maximum frequency, it consumes maximum power which does not improve energy saving. Similarly, using minimum frequency allows lesser power utilization at the expense of larger time to process the workload. As a result, more energy is usually consumed even for lower power consumption at minimum frequency. The dynamic approaches (i.e. *OnDemand, Conservative)* change the frequency during run time based on the workload. For example, if a certain workload requires higher performance, the *governor* will switch to a higher frequency (dynamically) and remain at that value. This can provide power (as well as energy) savings for applications which do not have too fast workload variations. However, in case of applications having variation of workload on relative large scale, it cannot provide the maximum potential savings.

The academic research of section 2.1.5 reviewed various power management strategies using broadly DVFS and DPS techniques. Mostly, these techniques rely on past or future workload predictions of a certain application. However, this prediction is usually approximated and may not work as expected in a real implementation. The workload predictions also require constant tracking and extreme analysis that may introduce important overheads and further errors in approximation. It should be noted that most of these techniques have been validated by simulations rather than implementation in real life. In contrast to workload based, we can therefore explore techniques using

application parameters for power and energy management, and using more realistic validation approach. Using such techniques, we can experiment with user defined power management policies based on different application parameters (e.g. based on a quality of video request in case of a video) to save energy. The use of specific or custom power management strategies (in contrast to workload based) may provide extra room for energy gains by utilizing DPS and DVFS techniques more finely and more efficiently.

In addition, a lot of algorithms and techniques exist for single and multiprocessor power management. However, due to their complexity very few are actually implemented on real platforms. In this work, we experiment with different policies for power management that are fully operational on both single and multiprocessor platforms. The validation of power management policies in a multiprocessor environment is also an important issue. In this work, the validation is also based on considering representative application examples including the H.264 video processing standard. Video processing is an interesting example as it exposes high workload processing and variations that can be efficiently exploited at a power management level. For instance, we experiment a truly operating H.264 decoder and with different parallelization models of an H.264 encoder in various configurations of platforms (number of cores, operating points). We also explore the effects of different application parameters (frame rate, parallelism, slack time, etc.) on power and energy consumption for each configuration.

In the following section, we detail the platforms used to address the above problems.

2.2.2 Platforms

The power management policies used in this work can be implemented on any platform having Linux or Embedded Linux support. Mostly, the platforms only allow changing frequency of a processor but do not provide access to monitor the power consumed by the processor. Vendors have their intellectual property rights to their systems and do not allow such access. Hence, it is practically impossible to access and monitor the power utilization of the underlying core. The power policies used in this work are experimented to analyze the power and energy consumption on different platforms, access to the processor's power consumption is thus needed. Therefore, we rely on ARM based platforms as they provide direct access to power monitoring registers of the cores. ARM baseboards give user access to monitor the change in voltages and currents related to the

processor. By this mean, we can calculate the real instantaneous power, mean power as well as energy directly related to the underlying core. In this work, we rely on these ARM based platforms to implement the prescribed power management strategies. Another reason of using these platforms is to address the target architecture (i.e. ARM) in the *COMCAS* project.

At the beginning of this work, no platform supporting altogether multicore, DVFS and power monitoring facilities were available. Therefore, we started working on an ARM emulation baseboard containing an ARM11 MPCore processor but without DVFS support. Then we used a mono processor platform containing a single core ARM1176JZF-S, but supporting DVFS and power monitoring. Finally, we used virtual platforms that support DVFS, multicore and power monitoring functionality. The virtual platforms also permit to target perspectives of future architectures, based on Dual CortexA9, that was not available at the beginning of the work. In the following sections, we provide detailed information of all the platforms used in this work. We start by the MPCore emulation baseboard, followed by the Versatile Baseboard ARM1176JZF-S. We then detail the QEMU [54] based virtual platforms (QEMU_ARM1176, QEMU_CortexA9). Additionally, we detail the methods for power and energy monitoring on these platforms.

2.2.2.1 ARM11 MPCore

The study focuses on the experimentation and validation of different power management strategies in the real world. For this purpose, we start by using a previous contribution prior to this work, which investigated power strategies using an ARM emulation baseboard containing an ARM11 MPCore chip. This multiprocessor platform provided many results and the definition of a DVFS video strategy. The MPCore chip supports dynamic voltage scaling as well as static frequency scaling; however, lack of real DVFS support stopped us to further work on this platform. Therefore, we used another platform (PB ARM1176JZF-S) to pursue these investigations and further experiment with the video power strategy. The main features of the ARM emulation baseboard are briefly highlighted below.

The emulation baseboard is a highly integrated development board containing tile connectors to connect an ARM11 MPCore chip. It also contains a large FPGA (Xilinx Virtex-II XC2V6000), 256 MB of 32 bits wide DDR SDRAM, 4 MB of 16 bits wide

cellular RAM, 128 MB of 32 bits wide NOR flash, Ethernet support, USB 2.0 controller, VGA/DVI, Keyboard, Mouse connectors, SD Card slot, 4 UARTS and character LCD, switches, LEDs and GPIOs. The MPCore chip contains four ARM11 cores, where each processor has a 32 KB instruction memory and 32 KB data memory. The nominal frequency for each core is 200 MHz. The platform supports embedded Linux however, the support for Linux *CPUFreq* cannot be used due to lack of DVFS support by the MPCore chip. The emulation baseboard does not support ARM's *IEM or IEC* technology discussed in section 2.1.4.3.

2.2.2.2 ARM 1176JZF-S

We thus considered an ARM1176JZF-S platform to continue the study of power strategies, but for a mono processor platform in this case. The first two power strategies (based on DVFS) are experimented using an ARM1176JZF-S baseboard as it provides all the required functionalities. The investigations on this platform require a power monitoring module and a frequency switching mechanism. Therefore, we developed a kernel module to monitor the processor's power. Secondly, we created a user space program which is able to change the frequency with the help of the Linux *CPUFreq*. The main components of the ARM1176JZF-S platform are detailed below, followed by the frequency switch mechanism and power monitoring drivers.

The baseboard is a mono processor development platform having an ARM1176JZF-S core mounted on it. The basic system provides a good platform for developing systems supporting ARM11 processors that feature TrustZone® Technology, CoreSight™, DVFS and *Intelligent Energy Management* (IEM see section 2.1.4.3). The platform also contains 128 MB of 32 bits wide mobile DDR RAM, 8 MB of 32 bits wide static PSRAM, 2 x 64 MB of 32 bits wide NOR flash. The NOR flash is used to save a boot loader and an embedded Linux OS image for auto boot. Other peripheral's devices include connectors for VGA, Color LCD display, PCI, UART, GPIO, Keyboard/Mouse, Smart Card, USB, audio, MMC and Ethernet.

The platform supports embedded Linux configured with *CPUFreq*. The *CPUFreq* contains *CPU_Freq* driver that uses IEM to change frequency and adjust the desired voltages. *CPUFreq* also contains a *Userspace* governor that can be used to switch frequency as needed. We therefore created a user space program (using *Userspace* governor) to change and monitor the frequency of the processor as required. It should be

noted that the ARM1176JZF-S processor has a nominal frequency of 240 MHz, and it can switch to three other frequencies that are 160 MHz, 215 MHz and 265 MHz. To experiment with the PM strategies, the latest available kernel image (i.e. Linux 2.6.32) was used and cross compiled by *Code Sourcery* [55] development tools.

The platform also provides access to the processor's voltage and current consumption register. The register labeled **PWR_VOLTAGE_CTL0** is used to monitor power consumption. A kernel module is created to access this register at regular intervals of time. The **PWR_VOLTAGE_CTL0** register has two fields; the first [0:15] bits are used to read the voltage of the core and the other [16:31] bits are for the current consumed in the ARM1176JZF-S chip. The reading from the register must be converted to an absolute value by the formula:

$$Core_Voltage = (PWR_VOLTAGE_CTL0\ [15:0] * 18) / 655200 \quad (Volt) \qquad Eq\ (1)$$

$$Core_Current = (PWR_VOLTAGE_CTL0\ [31:16] * 18) / 655200 \quad (Amp) \qquad Eq\ (2)$$

The power is simply calculated as:

$$Core_Power = Core_Current * Core_Voltage \quad (Watt) \qquad Eq\ (3)$$

The power monitoring driver measures the instantaneous power every 200 milliseconds. It can also calculate the mean power and total energy consumed between two defined points. Experimentation results and analysis of the power strategies on ARM1176JZF-S platform with this measurement procedure is detailed in chapters 3 and chapter 4.

As the real goal of this work is to check the efficiency of power strategies for multiprocessor platforms, we use QEMU based virtual platforms for this intent. QEMU support was directly provided to us by TIMA laboratories in close cooperation during the COMCAS project. Another reason for using QEMU platforms is the flexibility to model different platforms (e.g. type of core, number of CPUs, operating points, idle/load power, etc.). For instance, the DVFS based power strategies depend greatly upon the characteristics of operating points. Therefore, we need to change the operating points to further analyze the strategies behavior. Since, ARM1176JZF-S platform is not a multiprocessor platform, and due to the unavailability of a real multiprocessor platform with all necessary features, we use QEMU based virtual platforms for onward evaluation and experimentations.

2.2.2.3 QEMU_ARM1176

We rely on QEMU [54, 56] based platforms for the experimentation of multiprocessor power strategies. QEMU is a generic and open source machine emulator and virtualizer. When used as a machine emulator, QEMU can run an OS and programs made for one machine (e.g. an ARM baseboard) on a different machine (e.g. PC). We rely thus on QEMU cycle accurate simulations to provide reliable performance and power estimations of power strategies for ARM1176 and CortexA9 based multiprocessor platforms. We start by using an ARM11 model based version of QEMU, where we have introduced the operating points of the ARM1176JZF-S in order to compare QEMU estimations with real ARM1176JZF-S measures. In rest of this document work, we refer ARM11 based version of QEMU as QEMU_ARM1176. We configure this platform in different operating point configurations detailed in chapter 3, for further evaluation of a DVFS video strategy. The same platform is used for the implementation and experimentation of the second DSF low power strategy. The following section puts light on the main platform characteristics as well as the different features used.

The QEMU_ARM1176 virtual platform has almost the same capabilities as the real ARM1176JZF-S. It supports embedded Linux and provides specific drivers to use DVFS functionality. Additionally, we can configure it for single and multiprocessor configuration. We can have a configuration up to eight processors using the QEMU models. We are not bound to a fix number of available frequencies using virtual platforms. The platform parameters (such as the number of available frequencies, power levels, etc.) are flexible and can be modified. However, we limit ourselves to the frequencies and power levels used by actual hardware platform (ARM1176JZF-S) to provide realistic measures for the various power management strategies. Different drivers, functions and scripts are created to use the DVFS and power monitoring features of QEMU. A special function *Mean_Power* is used to measure the average power consumed by the processor between two time intervals. The number of processors to be used must be configured before starting the QEMU_ARM1176 platform. Once the platform is started, it acts much like a real hardware platform and starts with the initialized number of processors at nominal frequency (i.e. 240 MHz).

We completed experimentations and analysis of the first two strategies with the QEMU_ARM1176 platform. It should be noted here that we work with the QEMU_ARM1176 virtual platform as a starting point, keeping in mind that the final

target architecture was not yet available. Around halfway through the COMCAS project, we were provided with the CortexA9 based model of QEMU (referred as QEMU_CortexA9 in the following) by TIMA and Thales, which is presented in the next section.

2.2.2.4 QEMU_CortexA9

We experiment with the power strategies using QEMU_CortexA9 platform to analyze the energy savings on the new multiprocessor platform. In addition, this experimentation allows us to check the efficiency of the PM strategies and further characterization of platform parameters affecting the energy gains. We use the QEMU_CortexA9 platform to analyze the energy gains of two strategies, DSF (DVFS based) and AsDPM (DPS based), in the perspective of future multiprocessor architectures targeted in the COMCAS project. It should be noted that the use of this virtual platform was a way to target multicore CortexA9 based application processors that were not available as silicon at this stage of the project. The CortexA9 version of the virtual platform is developed by TIMA with inputs on the CortexA9 power and performance model provided by Thales Communication France. The following summarizes the platform characterization and power monitoring.

The new platform supports much higher frequencies (i.e. 300, 600, 1000 MHz) in comparison to those of QEMU_ARM1176. The QEMU infrastructure of implementation quickly allows us to export the already experimented power strategies (DSF and AsDPM) on the latest QEMU_CortexA9 platform. The implementation of the strategies is almost identical to the QEMU_ARM1176 platform with only few modifications in power monitoring and frequency drivers. The corresponding power levels for each frequency are also much more efficient in terms of idle and load power in comparison to that of QEMU_ARM1176 platform.

With the help of this platform, we can check the efficiency of power strategies in the perspective of CortexA9 multicore architectures which has very different characteristics in terms of static and dynamic power, due to different integration technologies. In addition, this also permits to compare the results obtained with previous architecture generations based on ARM11 cores (in terms of amount of energy gains) and to identify some conditions of efficiency at hardware level.

In the following section, we briefly introduce the power management strategies used for this purpose.

2.2.3 Power Strategies

We experiment a total of three different power management strategies in this work. The strategies are chosen depending upon the nature of applications used in our work, platform characteristics and for the reasons explained in section 2.1.7 and 2.2.1. The first two power management strategies are based on DVFS techniques, while the last strategy relies on DPS to exploit processor sleeping states. By this way, we could later provide a global evaluation and comparison of different PM strategies, along with the characterization of influential parameters affecting their efficiency in actual and realistic multiprocessor platforms.

2.2.3.1 DVFS Video Power Strategy

First, a DVFS based power strategy dedicated to video processing with an application to an H.264 decoder is used. The power strategy is based on the exploitation of frame rate variations of a video. Previous works [57] have shown variations in the frame decoding time of around 40%. These variations are exploited to adapt the processor speed to match the amount of processing for decoding frames. The DVFS based video strategy uses average frame rate on a time window to decide the best suitable speed for the processor. The adaptation strategy controls the decoder speed around a frame rate constraint slightly slower than the average performance. The operating frequency of the processor is decreased this way to fit the desired frame rate. By this way, processor power consumption is decreased which in turn also reduces the overall energy expenditure of video. In chapter 3, we present a detailed analysis on the effectiveness of this strategy using ARM1176JZF-S and QEMU_ARM1176 platforms.

2.2.3.2 Low Power DSF Scheduler

DSF is a scheduler targeting multiprocessor execution of applications that exploits DVFS techniques. Its principle is detailed in section 4.1.1 of chapter 4. This algorithm has been defined at LEAT in the work described in [45]. This scheduler is one of the low power strategies provided to the COMCAS project that has to be validated on the actual

target architectures (ARM1176 and CortexA9 based). This actual implementation is a Linux based implementation described in [53] which allows to execute and experiment this scheduler on any platform supporting Linux. The results of experimentations with the DSF strategy in different configurations of multiprocessor platforms are detailed in chapter 4.

2.2.3.3 Low Power AsDPM Scheduler

Assertive Dynamic Power Management (AsDPM) is also a low power scheduler for multiprocessor platforms. The AsDPM take advantage of DPS techniques and is based on a similar principle as DSF, using a Linux userspace scheduler. This scheduler is used to execute the strategy on any multiprocessor platform supporting Linux. The principle of this strategy is detailed in section 5.1.1 of chapter 5. This scheduler is developed at LEAT on work described in [58]. In this work, we validate the implementation of this scheduler on representative target platforms and analyze the obtained energy gains. The results of experimentations with the AsDPM strategy are provided in chapter 5.

In the following, we thus experiment, analyze and discuss the results of above described power strategies, using the different platforms mentioned: ARM11 MPCore, ARM1176JZF-S, QEMU_ARM1176 and QEMU_CortexA9.

We first start with the DVFS based video strategy on ARM11 MPCore and ARM1176JZF-S, then with the DFS strategy on QEMU_ARM1176 and QEMU_CortexA9 and next by the AsDPM strategy on QEMU_ARM1176 and QEMU_CortexA9. The numerous amounts of power measurement results will be extensively analyzed in order to extract relevant meaningful conclusions on the efficiency of power strategies in the real world. It should be noted that these experimentation have been the subject of two demonstrators developed in the scope of the COMCAS project that are fully part of this work and has been underlined by the CATRENE evaluation committee.

Chapter 3 : DVFS VIDEO POWER STRATEGY

3.1 Introduction

Dynamic Voltage and Frequency Scaling (DVFS) is widely used for power management in modern processors as stated in section 2.1.3. It is an effective method for achieving low power consumption of CPU while meeting the performance requirements. In this chapter, we present a DVFS based power strategy dedicated to video processing, with an application to H.264 decoder. We analyze the potential of this DVFS power strategy across different platforms with recent generations of ARM processors as discussed in section 2.2.1.

Allowing OS and related software to gain control over power consumption is really gaining more and more interest these days since energy reduction is one of the prime concern in embedded systems. The management of power saving techniques through an operating system requires software that must be able first, to identify the various operating points of the processor(s) and second, to assess requirements derived from the application and its environmental context. As seen in section 2.1.4.4, Linux operating system lets the opportunity to define custom power management strategies resulting from the availability of power management APIs (*CPUFreq*). Concerning DVFS, Linux has different governors (i.e. *Conservative, OnDemand, Userspace, Powersave and Performance*) as detailed in section 2.1.4.4, to adjust power levels statically or dynamically according to the processor workload. In the strategy description and implementation, we therefore rely on Linux for the implementation of this custom DVFS based video power strategy. The use of application specific power management techniques provides extra room for power saving by utilizing the application parameters, as defined in section 2.1.7. The strategy discussed in this chapter is dedicated to video decoding and uses DVFS technique to control the frame rate by the CPU frequency, and thus to reduce power consumption. The new adaptive strategy can be used for future implementation of adaptive systems to achieve performance goals as discussed in [59], and for the reasons explained previously in section 2.1.6 and section 2.1.7. It is implemented under the latest available kernel at the time of implementation (i.e. Linux 2.6.33) with *CPUFreq* support.

It should be noted that this work mainly focuses on experimenting with the strategy in actual implementation world as discussed in section 2.1.7. We see the effects of different application parameters, as well as platform parameters that impacts power consumption. Therefore rather than focusing on the description of a new power management strategy, the work is mainly focused on experimenting with the DVFS based video strategy in the real world, how efficiently it works for different platforms (real or virtual), which platform parameters effect the energy savings, how much energy can be saved, and if it is efficient for multiprocessor environments.

The application specific strategy can provide significant power and energy gains in certain conditions. We analyze the effectiveness of this strategy using an ARM1176JZF-S platform. We also analyze the effects of changing platform characteristics (power levels associated with corresponding frequencies, Number of processors etc.) by implementing the DVFS power strategy on the virtual QEMU platform. The results reveal that in certain conditions, reducing the frequency is effective but in others it actually increases energy consumption. So this study shows that due to different power consumption levels related to a frequency for different platforms, decreasing the CPU frequency may not always reduce the energy consumption. The results also provide percentage of energy gains up to 57% by choosing correct adaptation constraints. However, the experimentation shows that the efficiency of a DVFS strategy depends upon the characteristics of operating points (frequency/voltage couples). In particular, attention must be paid to the power level gap between consecutive frequencies, to compensate the increase in execution time resulting from frequency downscaling.

The outcome of this chapter is the following. Section 3.1.1 discusses a previous case study on H.264 video decoding on MP Core emulation baseboard. Based on the case study, a description of the DVFS based video power strategy is presented in section 3.1.2. We then provide the detailed implementation and experimentations for the video PM strategy in section 3.2. We provide DSF implementation in section 3.2.1, power and performance profiles of the H.264 based video sequence on the ARM1176JZF-S platform in section 3.2.2 and in section 3.2.3 we present an analysis of energy consumption. In section 3.3, we further examine the energy saving conditions by investigating the operating point effects on the efficiency of the DVFS strategy. For this purpose, operating point setup using QEMU platform is presented in section 3.3.1. The accuracy and behavior of virtual platform estimations is addressed in section 3.3.2 and

section 3.3.3 provides results and discussion of experimenting with the PM strategy in different operating point configurations. In section 3.4, we present our conclusions on the experimentations with the DVFS video strategy.

3.1.1 Case study: H.264 Decoder

A previous case study of application mapping of a H.264 decoder is detailed in [57]. Several H.264 videos (variable sizes) have been experimented and analyzed to observe their behavior in different situations using MPCore emulation baseboard. The videos have been divided into different slices versions, where each slice was taken as a thread. A slice represents an independent zone of a frame and can refer previous frame for decoding. Therefore decoding one slice of a frame is independent from another slice of same frame. The slice is handled by a POSIX thread and this way decoder can process different slices of a frame in parallel. The advantage of slice decomposition is its regularity and homogeneity which is suited for SMP implementations and makes balancing the workload between processors very easy. A detailed analysis of different slices versions of H.264 video decoder (1, 2, 4 and 8 slices) for four different videos is provided in this case study for different platform configurations (1, 2, 3 and 4 processors). The study pointed out that the frame rate (frames per second *fps*) is very sensitive to motion properties in the video, with variations of about +/- 20%. This can be exploited to define the dynamic video power strategy based on a frame rate adaptation using frequency scaling, which is discussed in next section 3.1.2. Furthermore, the case study also presents the power and energy behavior by trying to implement the power strategy. However, the platform used (MPCore Emulation Baseboard) does not support DVFS, it only supports static frequency scaling or dynamic voltage scaling. Therefore the results do not provide real energy and power consumption results of the dynamic power strategy. To overcome this, we experiment the strategy on ARM1176JZF-S platform for real implementation results. In the next section, we present the DVFS based video power strategy in detail.

3.1.2 DVFS Video Strategy Description

The principle of the adaptation strategy is to control the decoder speed around a frame rate constraint slightly slower than the average performance in nominal conditions. By this way, we can set dynamically the minimum processor frequency needed to comply with the minimum frame rate constraint and save actual amounts of energy, instead of

operating at the worst case maximum frequency. The adaptation is based on changing the frequency when the frame rate is between two defined thresholds. The number and values of these thresholds depend on the operating points (frequency/voltage) and on the performance constraint to satisfy. In the following, we therefore define the thresholds and the PM strategy for the ARM1176JZF-S platform.

In nominal conditions, the frequency of the ARM1176JZF-S core is 240 MHz. The supported frequencies are 160 MHz, 215 MHz, 240 MHz and 265 MHz as stated in section 2.2.2.2. To implement the DVFS video strategy on the ARM1176JZF-S, we calculate four thresholds derived from the above supported frequencies. Each threshold '$thresh_i$' is associated with a given frequency 'f_i' which is computed as follows:

$$thresh_i = adaptation_constraint * f_{nom} / f_i \qquad \text{Eq (4)}$$

In equation (7), f_{nom} is the nominal frequency (240 MHz). The decoder speed is controlled by the *adaptation_constraint*. We set an *adaptation_constraint* lower than the average frame per second decoded at nominal frequency, the video quality is slightly decreased alternatively the frequency and power consumption also decreases. This affects the overall energy consumption of the application. We therefore experiment with the DVFS strategy for different values of adaptation constraints.

Another important factor of the video PM strategy is to minimize the number of clock switching as it produces delay and extra energy overhead. We have defined two variables namely T_{eval} and $T_{monitor}$ to control the frequency switching. The frequency for the strategy is switched every (T_{eval} * $T_{monitor}$) images. The value of these variables is statically set by the user before executing the video strategy. The frame rate samples are taken at regular time intervals (every $T_{monitor}$ images). T_{eval} is the number of samples needed to decide about switching the frequency or not, based on the average frame rate on these samples. The record of T_{eval} number of decoded frames is stored in a table namely *PM_Table*. The average fps *avg_fps* of the *PM_Table* value for the last (T_{eval} * $T_{monitor}$) frames is calculated. This *avg_fps* value is then compared with the pre-calculated $thresh_i$ levels. Based on this value, a decision whether to change the frequency or not is taken. When the average decoder speed for (T_{eval} * $T_{monitor}$) images remains in a zone delimited by two consecutive thresholds $thresh_i$, the processor frequency is switched to the value associated with this zone.

We used the Linux *CPUFreq* to control the frequency of the processor. We have thus developed a DVFS strategy on top of Linux *CPUFreq* with the following characteristics. It handles the switching of operating points (frequency/voltage) with respect to the decoding speed. It samples the frame rate at regular time intervals (every $T_{monitor}$ images), and makes the decision of switching or not the operating point every (T_{eval}*$T_{monitor}$) images. In the next section we detail the implementation of the video strategy on the ARM1176JZF-S platform followed by the experimentation of an H.264 video sequence example.

3.2 DVFS Strategy Implementation and Experimentation

The ARM1176JZF-S platform support for embedded Linux and detailed knowledge about hardware and power monitoring registers is already given in section 2.2.2.2. However, we still have to connect the PB ARM1176JZF-S to the host computer and to load the developed programs on the platform. Section 3.2.1 details the implementation procedure of DVFS video strategy and section 3.2.2 provide the power and frame rate profiles of DVFS strategy on the multithreaded H.264 decoder. Subsequently, in section 3.2.3 we analyze the first obtained results.

3.2.1 DVFS Strategy Implementation

We first connect the PB ARM1176JZF-S platform to a host computer through serial cable RS232. The commands are given to the platform through the standard Linux *terminal* using the *Minicom* utility [60]. The host computer also contains a shared folder (*Arm)* that contains the cross compiled executable files and sample videos for execution on the ARM1176JZF-S platform. The *Arm* folder also contains a script that automates all the measurement procedure and functions.

At the start, the script adds the developed power monitoring driver *consumption_ARM11* to the ARM1176JZF-S platform. The *consumption_ARM11* contains functions to read the current and voltage values from the Virtex-4 FPGA controlling the programmable power supply in the ARM1176JZF-S platform. The registers used for this purpose as well as the formula to calculate power is explained in section 2.2.2.2. The *consumption_ARM11* reads and calculates the processor power consumption from these

registers. This value is transferred to the Linux terminal console on the host computer to be saved in a file for further analysis. The driver also calculates the mean power from the start to end of its execution. Secondly, we have written a PM frequency function (*PM_driver)* containing user functions to change the frequency dynamically with the help of Linux *CPUFreq*. The *CPUFreq* contains a governor *(Userspace)* that can be used to dynamically change CPU frequency. The *PM_driver* also contains functions to initialize the frequency to the nominal as well to a user given value. However, the values should correspond to the built-in available frequencies of the ARM1176JZF-S, otherwise it does not change the frequency and give an error message. Thirdly, we use *PM_scheduler* containing code for the power strategy described in section 3.1.2. The *PM_scheduler* uses the *PM_driver* to change frequency when required based on the *PM_Table*. The number of frequency switching is saved in a file for analysis along with the total energy consumption.

3.2.2 Power and Frame rate profiles

We start the experimentations by monitoring the performance of a 1 minute 20 seconds video sequence at nominal condition on ARM176JZF-S platform (at 240 MHz). The video sequence is decoded at an average speed of 11.6 fps as shown in figure 3.3 (blue dotted line). The video sequence consumes 52 Joules. Afterwards, we changed the *adaptation_constraint* as stated in section 3.2.1. The value of T_{eval} is chosen as 5 and $T_{monitor}$ is initialized to 50. Therefore, a decision to either switch frequency or not is taken after 250 (5 x 50) frames. The power and frame rate profiles of the video sequences are given in figure 3.1 to figure 3.4, for *adaptation_constraint* of 8, 9, 11 and 16 fps respectively. The figures show traces of the original (full red lines) and regulated (dotted blue lines) frame rate, as well as the trace of power consumption. On the power profiles (lower part of each figure), we can clearly observe different frequency domains, thus the DVFS switches that can be identified by distinct zones of stable values.

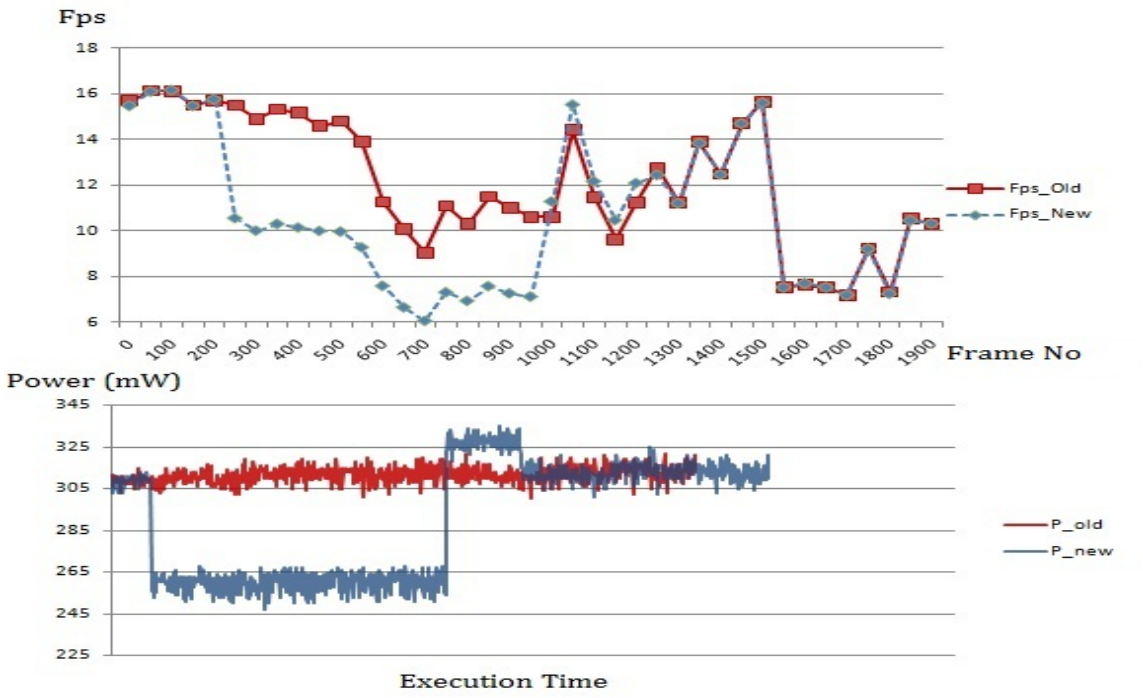

Figure 3.1: Power and frame rate profiles for *adaptation_constraint* of 8 fps.

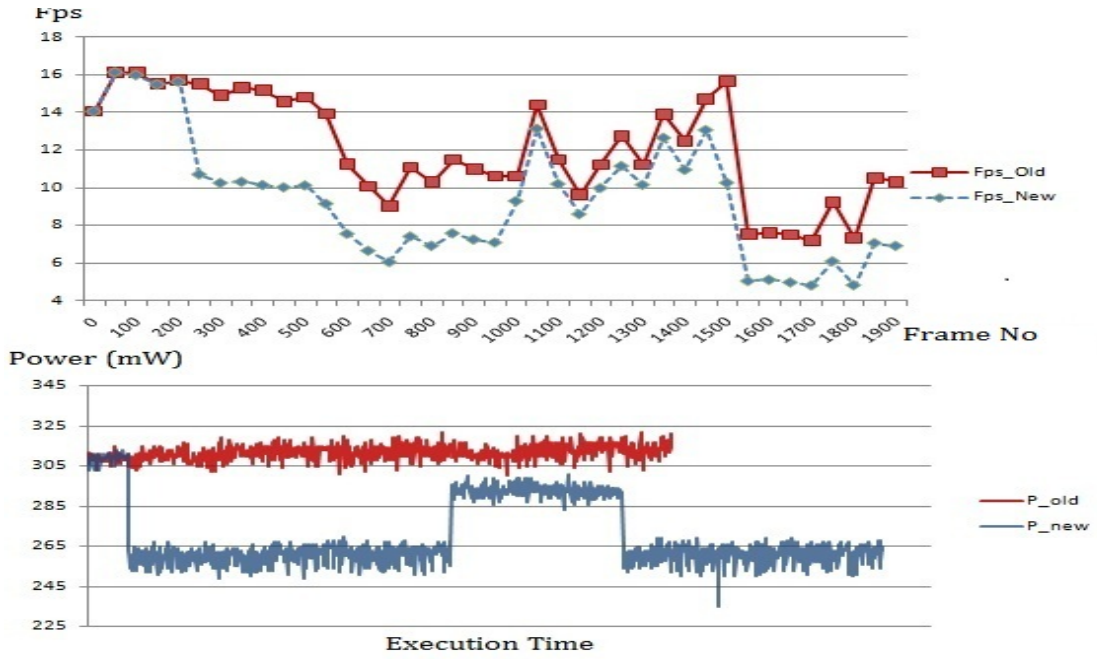

Figure 3.2: Power and frame rate profiles for *adaptation_constraint* of 9 fps.

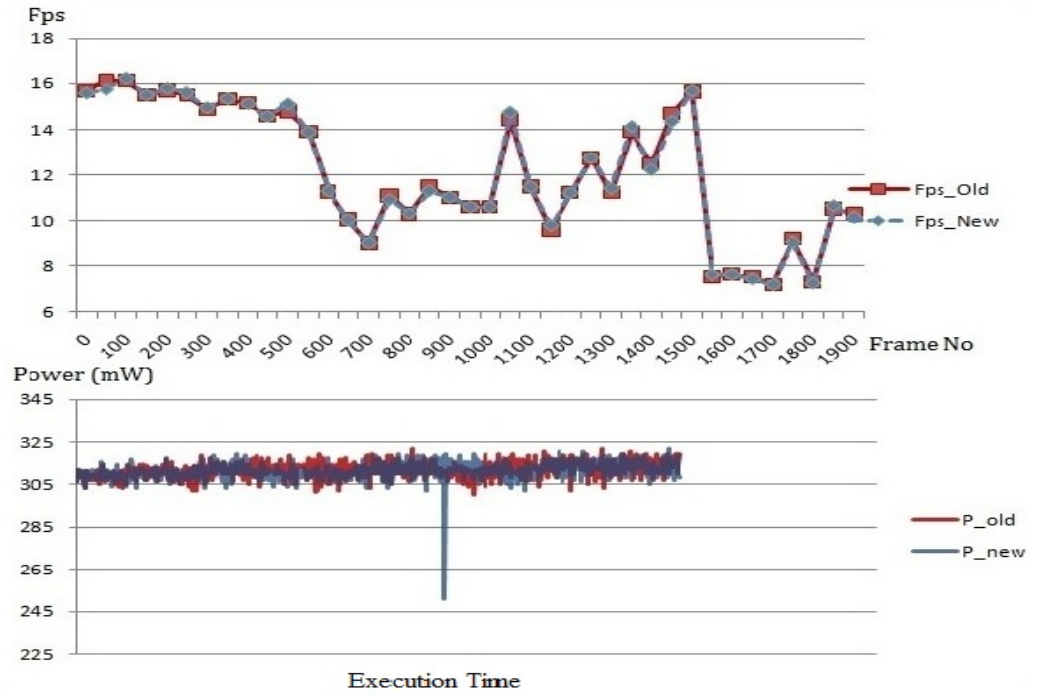

Figure 3.3: Power and frame rate profiles for *adaptation_constraint* of 11 fps.

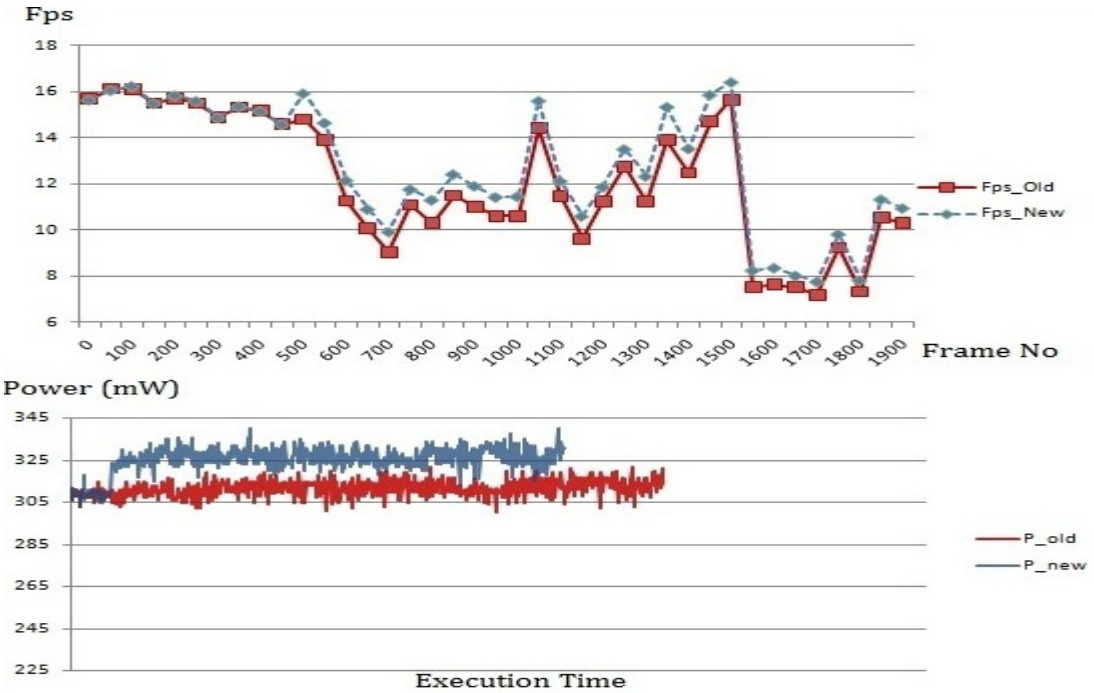

Figure 3.4: Power and frame rate profiles for *adaptation_constraint* of 16 fps.

Figure 3.1 shows the power profile for *adaptation_constraint* of 8, where video is decoded in 228 seconds. For an adaptation constraint of 8 fps, we have the following thresholds: 12.00, 8.93, 8.00 and 7.25 fps defined respectively for 160, 215, 240 and 265 MHz. When the decoder speed remains in a zone delimited by two consecutive thresholds during the last 250 (5 x 50) frames, the processor frequency is switched to the value associated with this zone. As an illustration, if the average frame rate is between 7.25 and 8.00 fps during last 250 frames, the operating point is set to 160 MHz. When the decoder speed is between 8 to 8.93 fps, frequency is switched to 215 MHz. In figure 3.1, we can see three frequency switches along with the corresponding switch in power consumption profiles.

Figure 3.2, figure 3.3 and figure 3.4 show the respective frame rate and power profiles for *adaptation_constraint* of 9, 11 and 16 fps respectively. The values of $thresh_i$ are calculated separately for each case and a frequency switch occurs based on these values. Figure 3.2 shows three frequency switches and three similar levels for corresponding power switch. In figure 3.3, we have an *adaptation_constraint* equal to average frame rate at nominal frequency, therefore, there is no switch. In case of figure 3.4, we have chosen larger *adaptation_constraint* to observe the behavior of the PM video strategy. In this case, there is one frequency switch and a corresponding single shift in power consumption level. In the next section, a detailed analysis of these results along with the energy consumption is given.

3.2.3 Energy Consumption Analysis

The performance profiles of figure 3.1 to figure 3.4 allow comparing the evolution of the decoder performances at 240 MHz (full red line) versus scaling frequency (dotted blue line) at different adaptation constraints of 8, 9, 11 and 16 respectively. In table 3.1, a summary of total energy consumption for different values of *adaptation_constraint* for the above video sequence is given. We also record the total decoding time, mean power and time per frames to better analyze the behavior of the PM strategy.

Table 3.1: Performance Analysis table for ARM1176JZF-S

Adaptation constraint	Mean Power (mW)	Total Energy (Joules)	Time / Frame (ms)	Time (s)	Fps	Energy Gain (%)
No	311	52	86.4	168	11.6	0
4	262	64	126.1	244	7.9	- 23
8	270	62	117.6	228	8.5	- 19
9	290	56	101.4	196	9.9	- 8
11	311	52	86.5	168	11.6	0
16	325	50	81.2	157	12.3	4

Surprisingly, these results show an increase in energy utilization when the adaptation constraint is 4, which is the lowest performance level. As an illustration, the decoder speed for an adaptation constraint of 4 limits the decoder to stay in a zone limited by thresholds of 3.62 to 4.0 fps, so that the decoder uses the lowest available frequency (thus power level). Hence a decrease in energy utilization is expected, but it is not the case because the execution time has also increased in great proportion and this is directly linked to the operating point's characteristics.

In the next section, we investigate the above hypothesis by experimenting with the video power strategy on a QEMU platform, in order to analyze the effect of changing the operating points.

3.3 Further Investigation of Energy Saving Conditions

To complement previous results on the ARM1176JZF-S platform, we implement the video power strategy on the QEMU platform. The negative energy gains obtained are probably due to the different power level consumption associated with the respective frequencies. The differences of power level in consecutive operating points are small for the target platform we have used. The ARM1176JZF-S platform does not allow us to change the operating points, as they are provided by the vendor. Therefore, in order to evaluate the surprising energy consumption behavior, we analyze the video strategy by changing the power levels with the help of the virtual platform. In the following section, we start this investigation by setting the virtual platform in different operating point configurations.

3.3.1 Operating Point Set up on the Virtual Platform

The purpose of using the virtual platform is to define more efficient operating points and to check the impact on energy. Figure 3.5 shows the power levels associated with each frequency for ARM1176JZF-S platform as well as different configurations of QEMU platform. We must emphasize here that the measures are made on a versatile platform baseboard, which is for the early prototyping purpose and has performance limitations (especially concerning operating frequencies). A production device will have different characteristics, so the conclusions concerning the efficiency of operating points on ARM1176JZF-S cannot be generalized.

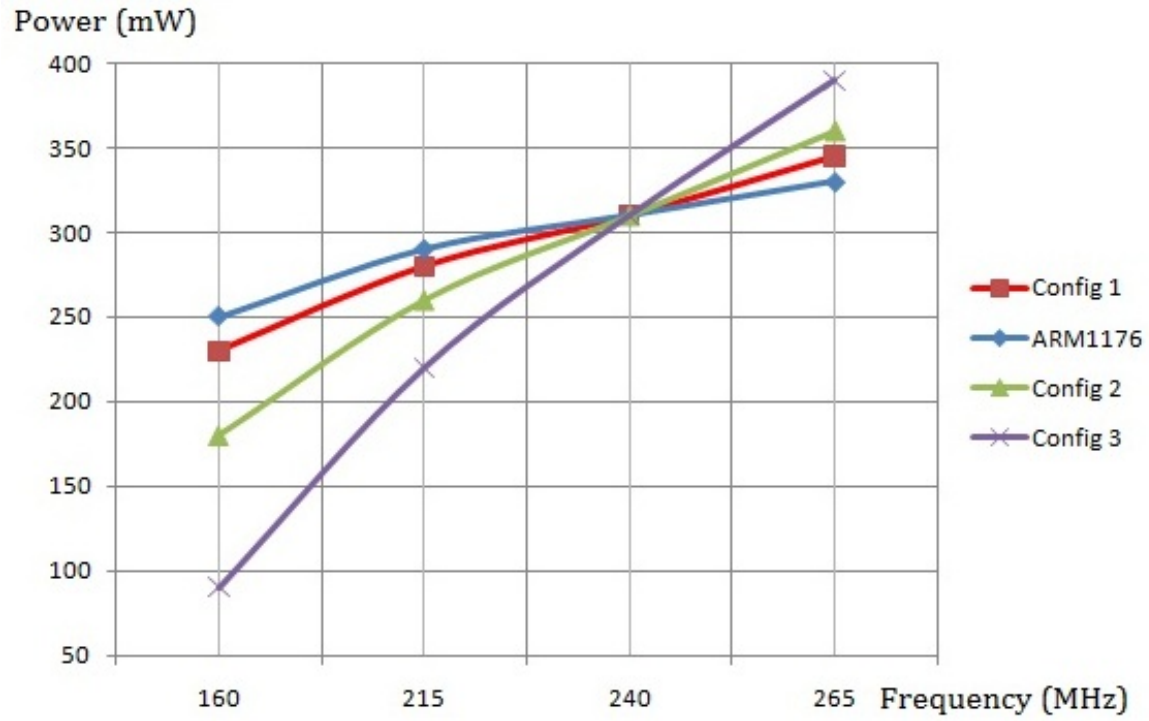

Figure 3.5: Frequency and load power consumption of QEMU platform.

We set up the QEMU platform in several configurations with respect to different operating points. A configuration for the ARM1176JZF-S is a set of four operating points, one for each frequency and its corresponding power consumption levels. As an illustration, for each available frequency of 160, 215, 240 and 265 MHz, the corresponding power levels with and without load are defined in the QEMU configuration file. These will be the operating points for the video power strategy. We have thus set three other configurations that are reported in figure 3.5. The choice of

these points depends upon the power level associated with each frequency. The power strategy relies on nominal frequency (240 MHz in this case), so we fix this point as reference and increase the power level gap between the subsequent upper and lower frequencies. We have set three configurations derived from the original ARM1176JZF-S operating points, but with increased power levels.

3.3.2 Accuracy and Behavior of Virtual Platform Estimations

Several different videos are allowed to run on both platforms to check the performance and energy consumption behavior. We analyze the energy and performance behavior for every frequency individually. Similarly, the energy consumption for the video sequence (*flavors_cut*) is also considered for comparison of these platforms. The energy consumption for video sequence (*flavors_cut*) on ARM1176JZF-S is 52273 mJ and on the virtual QEMU platform is 52109 mJ at a nominal frequency (240 MHz). The time taken to decode on real and virtual platform are 167.4 and 168 ms respectively. These results and simulations for several test videos show that both the platforms perform, approximately, similar behavior in terms of energy consumption and performance. There is a negligible error of 0.035% in timing analysis. Similarly, the energy consumption is also approximately, similar on both platforms with an error of 0.31%.

To better understand the effect of different operating point configurations, we also analyze the platforms behavior statically (fixed frequencies) and without using the DVFS power strategy. The total energy consumption by decoding a short video sequence of 300 frames (namely *foreman*) for each available frequencies (160, 215, 240 and 265 MHz) is shown in table 3.2. The energy consumption in case of *Config1* is 14 J at 265 MHz and 15.3 J at 160 MHz. This implies that when frequency is decreased, the energy consumed by the application is increased. Similarly, in case of *ARM1176,* we have an energy consumption of 13.4 J at 265 MHz and 16.6 J at 160 MHz (same behavior as previous *Config1*). In both cases lowering the frequency increases the energy consumption.

Table 3.2: Frequency vs. Energy for video Foreman (300 frames).

Frequency (MHz)	Energy (Joules)			
	Config1	*Arm1176*	*Config2*	*Config3*
160	15.3	16.6	12	6.2
215	14.4	14.6	13.1	11.1
240	14.2	14.2	14.2	14.2
265	14	13.4	14.6	15.8

However in case of *Config2* , the energy consumption measured at 265 MHz is 14.6 J and at frequency of 160 MHz is 12 J. This implies a decrease in energy consumption for a decrease in frequency. Similarly, for C*onfig3* we have energy consumption of 15.8 J and 6.2 J respectively, for the frequency of 265 MHz and 160 MHz. In both cases, lowering the frequency decreases energy consumption. From these observations we can conclude that for platform configurations *Config1* and *ARM1176,* lowering the frequency does not result in energy gains. Conversely, *Config2* and *Config3* provide energy gains. In the following analysis, we have further investigated this behavior and its causes but using the dynamic video power strategy.

3.3.3 Results and Discussion

We start by evaluating the DVFS video power strategy with the different operating point configurations and observe the overall performance and energy consumption for different *adaptation_constraint*. Figure 3.6 shows energy consumption for the video sequence (*flavors_cut*) at different adaptation constraints under different platform configuration. In figure 3.6, we can clearly remark that the energy consumption is increasing for platform configurations *Config1* and *ARM1176*, while decreasing for *Config2* and *Config3*. This is contradictory because the level of video quality decreases the frequencies at which the processor operates and would logically succeed in saving energy. This is not the case and the negative energy gains obtained are due to the fact that differences of power consumption in consecutive operating points are too small for the case of *ARM1176* and *Config1* as shown in table 3.3. Because of this, power reduction does not compensate the increase of execution time resulting from frequency reduction.

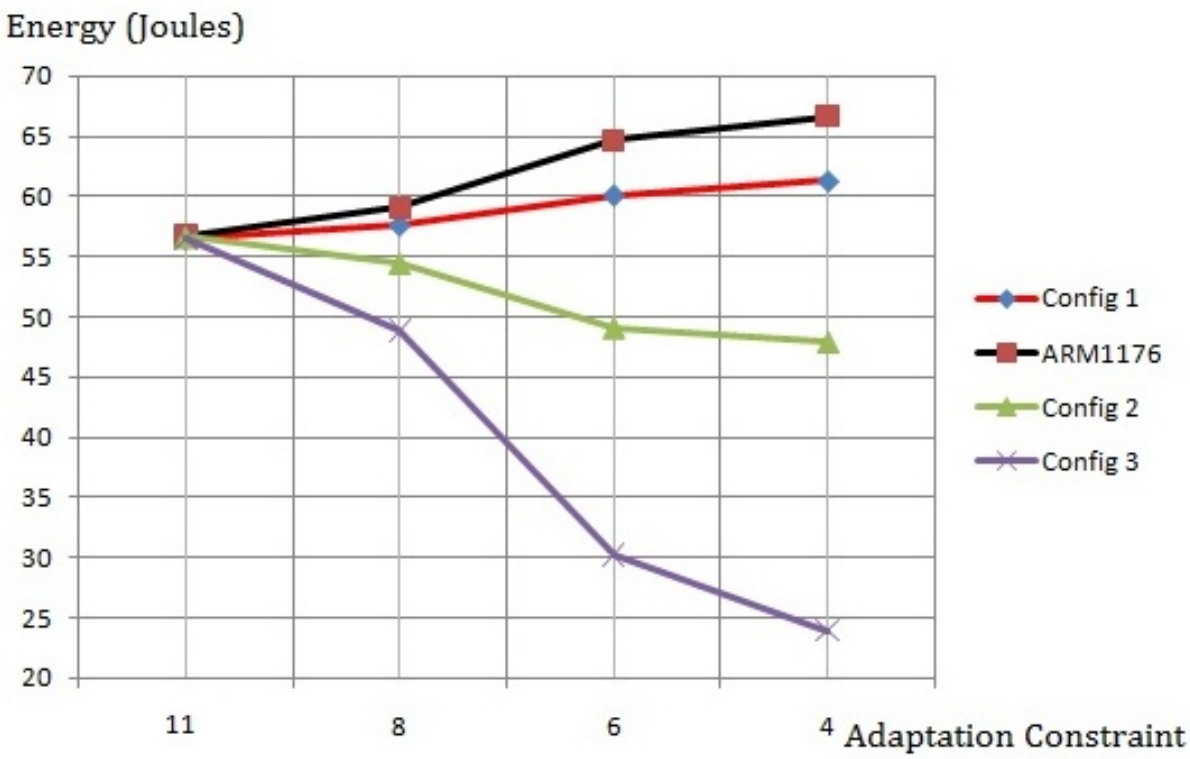

Figure 3.6: Energy consumption vs. Adaptation constraint.

As an illustration, the differences of power level between the consecutive frequencies of 240 MHz and 215 MHz for *Config1* and *ARM1176* are 30 mW and 20 mW respectively. This power level gap is not enough to compensate the execution time increase implied from 35 MHz downscaling. On the contrary, the same differences of power level for *Config2* and *Config3* are 50 mW and 90 mW respectively as indicated in table 3.3. Therefore we can see the energy gain is more significant as this gap increases as shown in figure 3.6.

Table 3.3: Frequency and load power consumption for different configuration of QEMU platform.

Frequency (MHz)	**Load Power (mW)**			
	Config 1	*ARM1176*	*Config 2*	*Config 3*
160	230	250	180	90
215	280	290	260	220
240	310	310	310	310
265	345	330	360	390

Similarly, the power level gap between frequencies 160 MHz and 215 MHz for *Config1* and *ARM1176* is 50 mW and 40 mW respectively, whereas the corresponding gap for *Config2* and *Config3* is 80 mW and 120 mW respectively. Again the energy is gained in case of *Config2* and *Config3* as shown in figure 3.6. This also means that not all platforms will provide energy gains when reducing dynamically the frequency and that the characteristics of operating points have a key impact on the efficiency of a DVFS strategy. This is the case on the Versatile Baseboard ARM1176JZF-S used for the experiments, for which decreasing the frequency increases the energy consumption. Thus the differences of power level between consecutive frequencies of the processor have important effects on energy consumption.

Secondly, the energy gain depends on the request of video quality (in terms of decoding speed required); the more quality, the less energy gains. In this case the *adaptation_constraint* controls the decoder speed, at low speed the decoder consumes less energy. Figure 3.7 summarizes the corresponding energy consumption in joules for the H.264 decoder in different platform configurations as well as at different *adaptation_constraint*. Figure 3.7 shows that the power strategy provides gains that are:

1. -2.05%, -6.46% and -8.54 % for *Config1*,
2. -4.1%, -14% and -17.5% for *ARM1176*,
3. +3.82%, +13.28% and +15.39% for *Config2*,
4. +13.6%, +46.55% and +57.68% for *Config3*.

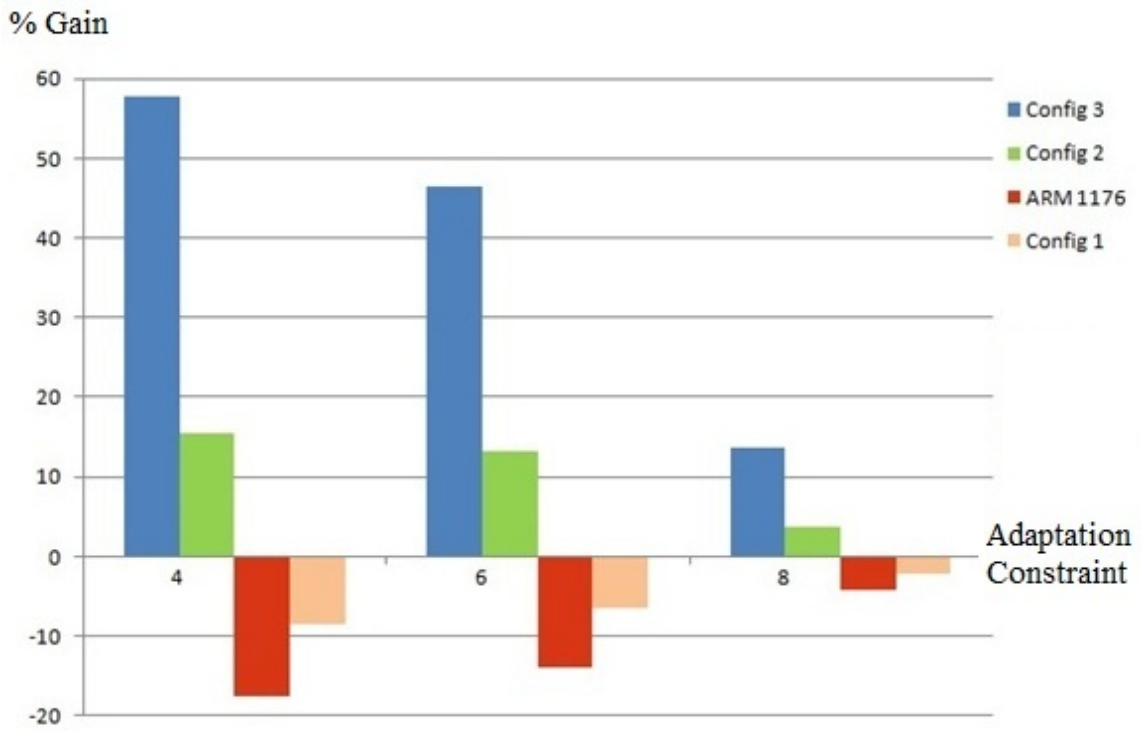

Figure 3.7: Energy vs. adaptation_constraint for different platform configurations.

The gain depends upon different *adaptation_constraint* (4, 6 and 8 respectively) for each different platform configurations. The results are coherent with the previous works and the results of the same DVFS strategy on a multi-core platform (MPCore Emulation Baseboard) that have reported energy gains from real measures, between 9.7% and 30.6% [57].

3.4 Conclusion

We have presented and analyzed the effectiveness of a DVFS power strategy dedicated to video decoding on an ARM1176JZF-S platform. Putting aside the energy gains, these results show that DVFS can be effective under certain conditions. In some cases decreasing frequency can actually increase the energy consumption. The operating points play an important role in the efficiency of a DVFS based power strategy. The differences of power between consecutive frequencies of the processor have important effects on energy consumption. In case of the Versatile Baseboard ARM1176JZF-S used for the experiments, the differences of power level between consecutive frequencies are

too small and result in an increase of energy consumption when scaling down the frequency. However, we have shown that if this gap is larger for two consecutive frequencies, there is a point where the energy gain becomes effective.

The research presented in this chapter is implemented on the real ARM1176JZF-S and on the virtual QEMU platform, to be able to consider different characteristics (matching those of ARM processors). The values provided on real and virtual platforms were equal in energy consumption (only 0.3% deviation). Using the virtual platform, we thus explored the effect of operating point characteristics on the efficiency of a DVFS strategy. It has been shown that the power level gaps should compensate the execution time increase resulting from a frequency decrease, in order to grant the energy gains. Under these conditions, the power strategy provides gain that ranges up to 57.68% depending upon the adaptation constraints.

The above work summarizes our discussion of the first DVFS based video power strategy. We would now switch to analyze the second DVFS based power strategies (i.e. DSF) on respective platform (i.e. ARM1176JZF-S, QEMU) in order to explore its effectiveness in the real development world. In addition, we further analyze the impact of operating points for the second DVFS strategy, and explore if generalization is possible and at which conditions.

Chapter 4 : DSF POWER STRATEGY

4.1 Introduction

In real-time systems, the use of variable frequency and voltage of a processor has a direct impact on processor's speed and consequently, on the ordering and the execution of tasks. Hence, scheduling techniques and voltage/frequency selection mechanisms are tightly coupled and should be addressed together to ensure the feasibility of application tasks under timing constraints. Real-time applications potentially exhibit variations in their Actual Execution Time (AET) and as a result, often finish earlier than their estimated Worst-Case Execution Time (WCET). DVFS techniques can exploit these variations to adjust the frequency and voltage of processors to reduce power and energy consumption (section 2.1.5). However, one of the challenges of these techniques is to preserve the feasibility of scheduling and provide deadline guarantees.

Experimentation with a second DVFS based strategy called Dynamic Stretch to Fit (DSF) is detailed in this chapter. DSF strategy is shown to be able to make significant energy savings while providing the required efficiency (real time scheduling). We detail the effects of this DVFS based power strategy on both single and multicore platforms. The experimentations are carried out with mainly three objectives in mind: (a) to check the effectiveness of the DSF power strategy on the ARM1176JZF-S platform and the two virtual platforms, (b) to characterize the range of energy gains from application parameters and (c) to analyze the effect of platform characteristics on the efficiency of the DSF strategy. The corresponding energy gains are presented in the results section.

We present the analysis study of the DSF strategy in the following manner. Section 4.1.1 describes the DSF strategy in detail. Section 4.1.2 provides information about the examples used for the experimentation. Section 4.2 is divided into three parts: section 4.2.1 explains the implementation of the DSF strategy, section 4.2.2 focuses on the experimentation of DSF using ARM1176JZF-S and in section 4.2.3 we put light on the effects of varying different application parameters on the energy savings. Afterwards, in section 4.3.1 we show the results of energy gains by experimenting with the DSF strategy in different multiprocessor configurations and in section 4.3.2 we give an analysis of these results. At the end, we present our conclusions in section 4.4.

4.1.1 DSF Strategy

The DSF low power technique uses Dynamic Slack Reclamation algorithm (DSR) as the principal slack reclamation algorithm. The AET of tasks within an application is always less or equal to the WCET. These differences produce time slacks by the current tasks to be used by the following tasks. The DSR algorithm assigns dynamic slack produced by the current tasks to the next priority ready tasks. This allocation provides more time for the new ready tasks, therefore the processor speed can be changed. Using DSR algorithm, the slack is fully consumed on the same processor by the task to which it is once attributed. Such allocation allows the DSR algorithm to have a large slowdown factor for scaling frequency and voltage for a single task, which eventually results in improved energy savings. DSR works in conjunction with global scheduling algorithms on symmetrical multiprocessor real time systems. The algorithm exploits the fact that distinct scheduling events have a different impact on an application's schedule. For instance, a terminating job may produce dynamic slack, but it does not increase concurrent utilization of the platform's resources and therefore, can only update the priority order of remaining ready tasks. A release event, on the other hand, increases the simultaneous platform utilization and may cause preemptions as well. This difference in the impact of scheduling events is exploited by DSR. At every scheduling event, the dynamic slack (difference between the AET and WCET) produced by the current tasks, is added to the WCET of the next ready tasks. By this way revised WCET for the new tasks (on the same processors) are calculated. Hence, novel frequency is calculated based on the updated WCET, and the tasks are allowed to run with the lowered frequency. This decreases power consumption and impacts the total energy consumption.

The next section presents a description of various examples used in our work.

4.1.2 Application Examples

To experiment with the DSF strategy, we created five distinct application examples for four different platform configurations. *Example 1* (see Table 4.1) is used to test the scheduling accuracy and energy savings of the DSF strategy for the ARM1176JZF-S and QEMU_ARM1176 platforms (single processor configuration). *Example 1* is also used for the evaluation of energy savings on the QEMU_CortexA9 platform. Afterwards, we

use three other application examples (*Example 2, Example 3 and Example 4*) to experiment with the DSF strategy in a multiprocessor configuration. In the end, we also experiment the DSF strategy on an *H.264Encoder* which is the application use case of the COMCAS project. The main parameters used in these examples are listed in table 4.1.

Table 4.1: Examples used for experimentation of DSF strategy

	Example 1		*Example 2*		*Example 3*		*Example 4*		*H.264Encoder*	
Task	2		4		6		8		4	
	WCET	BCET	WCET	BCET	WCET	BCET	WCET	BCET	WCET	BCET
T1	5.5	2	5	2	5	2	4	1	20.63	5.65
T2	4.5	3	4	3	6	3	5	1.5	20.63	5.65
T3			3.2	1.5	7	4	6	2	8.25	3.38
T4			3.4	2.5	4	2	7	2.5	5.78	1.81
T5					4.2	3	3	1		
T6					4.5	4	3.2	1.5		
T7							3.4	2		
T8							3.5	2.5		

Note: Here time is in 10^{-1}seconds.

Example 1 is composed of two tasks with the parameters specified in table 4.1. It is defined specifically for a single processor configuration. The AET of task T1 is taken between BCET and WCET with different values (i.e. 2, 2.6, 3.3, 4, 5, 5.5 10^{-1}s) to characterize the range of energy gains. The AET of task T2 is fixed at an average value between BCET and WCET. By this way, we have a variation of slack produced by the difference of AET and WCET of task T1 that allows to measure the range of energy consumption. For ease of implementation, we choose deadlines of tasks equal to their periods in most cases.

Example 2 consists of four tasks for the execution on a configuration of two processors. We choose different AET for tasks T1 and T2 in a similar way as for the task T1 of *Example 1*. The difference between the values of AETs and WCETs for the corresponding tasks T1 and T2 produce different slacks. These slacks are added to the WCET of tasks T3 and T4 respectively by the DSF strategy. *Example 3* contains six tasks for the execution on three processors and *Example 4* contains eight tasks for the execution on four processors. In *Example 3* we vary the AET for the tasks T1, T2 and T3

that produce slacks for the tasks T4, T5 and T6. Similarly in *Example 4,* we change AET of tasks T1, T2, T3 and T4 that produce different time slacks for the tasks T5, T6, T7 and T8.

The DSF strategy is then applied to the video *H.264Encoder* example. It should be noted that the *H.264Encoder* is not the real encoder application. It is a task model provided by Thales group for our experimentations. It consists of four tasks T1, T2, T3 and T4, where T1 and T2 are tasks used for motion estimation. T3 is used for intra prediction, texture encoding and syntax writing. T4 is a post processing filter. A graphical presentation of this task model is presented in figure 4.1. The first two tasks should be completed in parallel, however tasks T3 and T4 can be completed afterwards. This is controlled by the deadline of these tasks as shown in figure 4.1. Here T1 and T2 have deadline of 21 ms, T3 has a deadline of 31 ms and for T4 is 40 ms. The period is fixed to 40 ms for all tasks.

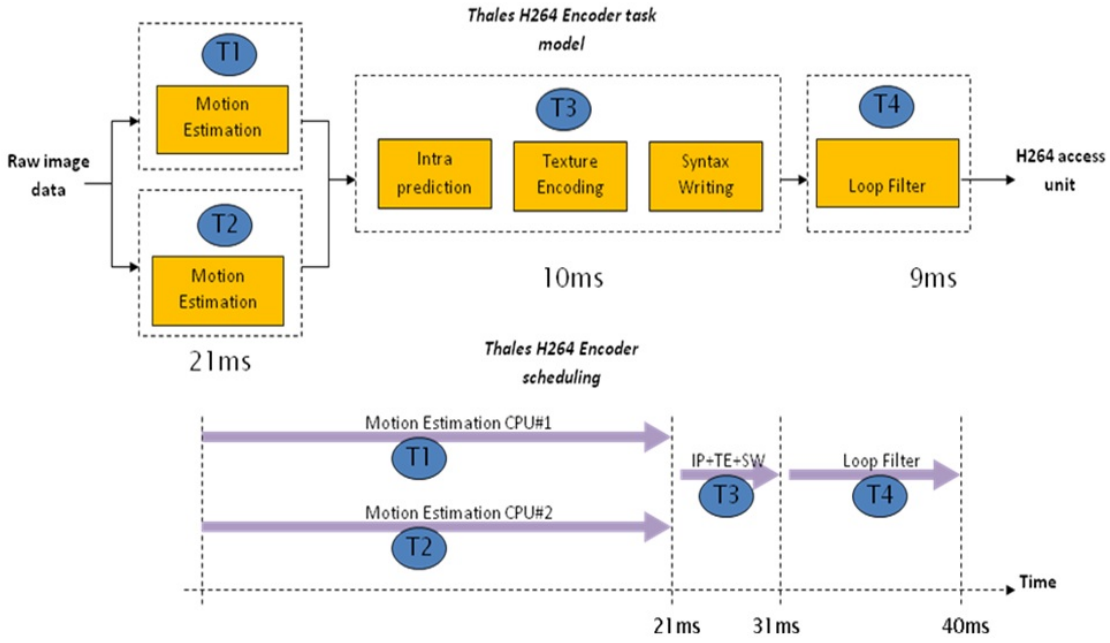

Figure 4.1: Thales Task model of H.264 Encoder.

In normal cases, tasks within an application execute with random values of AET between their BCET and WCET. However, we use static values of the AET for the execution of these examples. This allows us to have a same sequence of scheduling

events with and without the DSF strategy. In addition, the total execution time of the application is also not affected and provides the required deadline guarantees as well. This is needed in order to compute the relative energy savings with and without using the strategy, in the most comparable conditions of execution. In the next section, we present the actual implementation and experimentation of the DSF power strategy.

4.2 DSF Implementation and Experimentation

This section describes the actual experimentation of the DSF strategy on ARM1176JZF-S and QEMU platforms. We start by describing the DSF strategy implementation for these platforms in section 4.2.1. The accuracy of virtual platforms has already been presented in chapter 3 for several test applications. In section 4.2.2, we experiment with the DSF strategy to observe the energy gains for a single processor configuration. Afterwards in section 4.2.3, we analyze the effects of different application parameters (slack time, simulation duration, time granularity of tasks) on energy consumption and correctness of scheduling results.

4.2.1 DSF Implementation

We implement the strategy using the *DSF_Scheduler* code. It is based on a user space Linux scheduler developed at LEAT and described in [53]. It uses Linux and POSIX thread APIs to perform a schedule of tasks under deadline and priority constraints. With the association of Linux DVFS drivers (*CPUFreq*), this scheduler is able to apply a multitask, multiprocessor DSF strategy on any execution platform, provided it supports Linux and the required APIs (POSIX, *CPUFreq*, CPU affinity).

The *DSF_Scheduler* uses the respective *PM_Driver* for each platform to change the frequency based on the DSR algorithm. We use the previously defined *PM_Driver* used for the video strategy implementation. In case of QEMU_CortexA9, we modify the QEMU *PM_Driver* to use the frequencies of the CortexA9 processor. The *DSF_Scheduler* also uses the previously defined power driver (*consumption_ARM11*) to monitor the power of the ARM1176JZF-S core. The scheduler loads the *consumption_ARM11* driver at the start of the application and unloads it after its completion. The *consumption_ARM11* returns the mean power and time taken by the application during this period. The energy consumed by the application is calculated by

the *DSF_Scheduler* and is sent to the console for analysis. In case of QEMU platforms, the *DSF_Scheduler* relies on QEMU built-in functions for power monitoring and energy calculation (section 2.2.2.3 and section 2.2.2.4).

4.2.2 Experimentation on a Single Processor

The experimentation results of the DSF strategy using *Example 1* are shown in table 4.2 and in figure 4.2. The QEMU platforms are configured for a single processor. We first focus on providing comparative gains for the ARM1176JZF-S and QEMU_ARM1176 platforms. The percentage of energy gain is calculated by comparing the energy consumed by the application with and without the use of the DSF strategy. The energy consumption, mean power, execution time and the percentage of energy gain for different values of AET on both platforms are reported in table 4.2. These values are obtained by changing the AET of task T1 between BCET and WCET as stated in section 4.1.2. This allows the processor to switch between different frequencies, providing different value of energy consumptions.

Table 4.2: Energy consumption of *Example 1* on ARM1176JZF-S and QEMU_ARM1176 platform.

	ARM1176JZF-S				QEMU_ARM1176			
AET	Energy (mJ)	Mean Power (mW)	Time (sec)	% Gain	Energy (mJ)	Mean Power (mW)	Time (sec)	% Gain
Conf 1	2730.72	323	8.809	17.73	2722.29	309	8.81	17.80
	2246.48	256	8.81		2237.74	254	8.81	
Conf 2	2792.58	323	8.809	11.04	2783.96	316	8.81	11.39
	2484.3	281	8.81		2466.8	280	8.808	
Conf 3	2801.42	323	8.809	11.10	2792.77	317	8.81	11.15
	2490.47	287	8.81		2481.47	282	8.809	
Conf 4	2810.25	323	8.809	7.83	2801.58	318	8.81	7.86
	2590.11	294	8.81		2581.33	293	8.81	
Conf 5	2819.09	323	8.809	4.06	2810.39	319	8.81	4.08
	2704.64	307	8.81		2695.86	306	8.81	
Conf 6	2845.6	323	8.809	0.00	2836.82	322	8.809	0.00
	2845.6	323	8.809		2836.82	322	8.81	

For each configuration $Conf_n$, the upper value represents the energy consumption of the application itself without power strategy, while the lower value is the energy using DSF. The results show that we can have a gain varying between 4.06% and 17.73% for the ARM1176JZF-S platform and 4.08% to 17.81% for the QEMU_ARM1176 platform. It should be noted that we use operating points of ARM1176JZF-S for the QEMU_ARM1176 platform throughout the experimentation.

We then implement the DSF strategy on the QEMU_CortexA9 platform and start experimenting with *Example 1*. Figure 4.2 summarizes the percentage of energy gain for various values of AET and for all the three platforms.

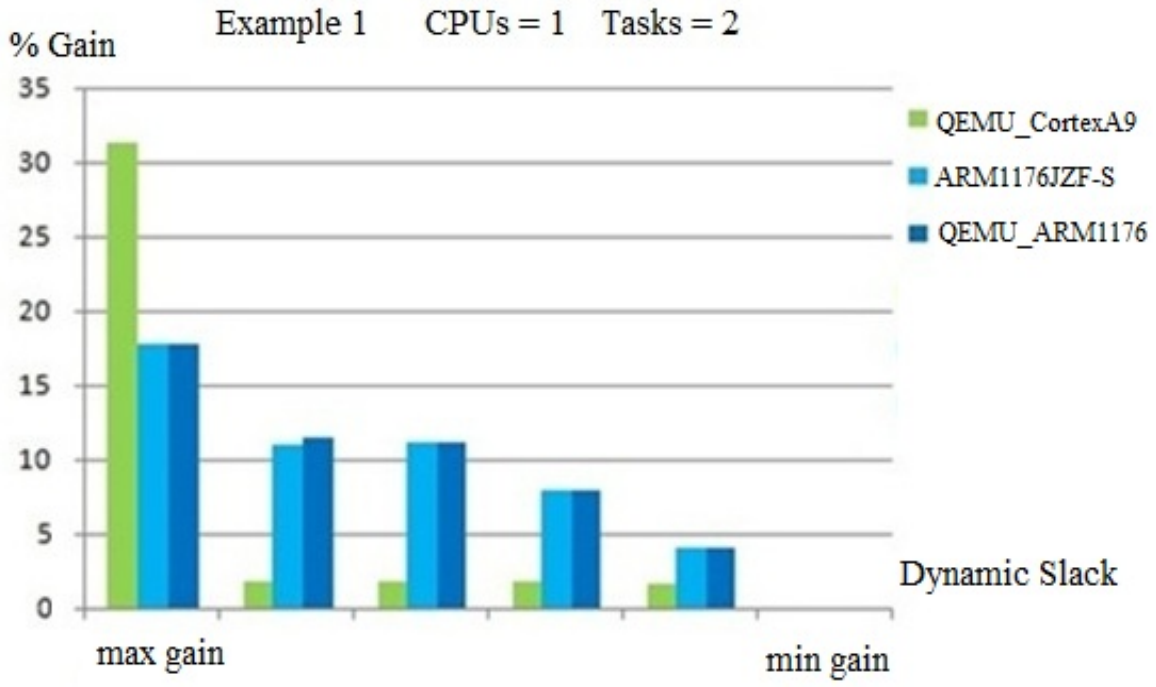

Figure 4.2: Percentage Energy gains for Example 1 on different platforms.

The energy consumptions (as well as the percentage of gains) are not the same for the CortexA9 and ARM1176. This distinct energy behavior is probably related to different operating points of these different platforms and is further analyzed in section 4.3.2. The results also show that we can have a gain varying between 1.56% and 31.27% for the QEMU_CortexA9. It should be noted that the percentage of energy gains for the ARM1176JZF-S baseboard and QEMU_ARM1176 platform are almost equal, having a negligible error of 0.3% as seen previously (hence this further confirm the virtual platform accuracy).

4.2.3 Experimentation with Application Parameters

Before analyzing the results of the DSF strategy on a multiprocessor example, we first explore the effects of varying application's parameters on a single processor. There are many parameters that can vary from one application's execution to another. These parameters may affect the scheduling behavior and/or the energy consumption by the applications. For instance, such a parameter in our examples is simulation time which is used for calculating the energy consumption by an application. Certain applications may require more time to complete their execution than others (like video sample of 100 frames vs. sample of 1000 frames), therefore changing simulation time may affect their energy consumption. Moreover, this can also cause influence on energy savings as larger simulation time may introduce additional frequency switching and computational overhead by the power strategy. Therefore, we first explore the effect of changing the total simulation time for *Example 1* that also helps us to evaluate the DSF strategy for other applications. Secondly, as the switching of frequency also imposes actual time penalties in real systems, therefore we define the application's parameters using different units of time (i.e. 10^{-3}, 10^{-2}, 10^{-1}, 10^{0} seconds) to see the influence of these solutions of task execution time (referred as *time granularity of tasks* in the following) on the efficiency of strategy.

We explore the effect of changing the total simulation time by the following procedure. First, we calculate maximum percentage of energy gain for *Example 1*, by fixing the AET of task T1 to BCET and note down the energy consumption with and without the DSF strategy. Secondly, we use a variable *loop* to control the number of iterations of the tasks T1 and T2, so that the total simulation time of the application can be changed. Thirdly, we note the percentage of energy gains for different values of *loop* on both ARM1176JZF-S and QEMU_ARM1176 platform for *Example 1*.

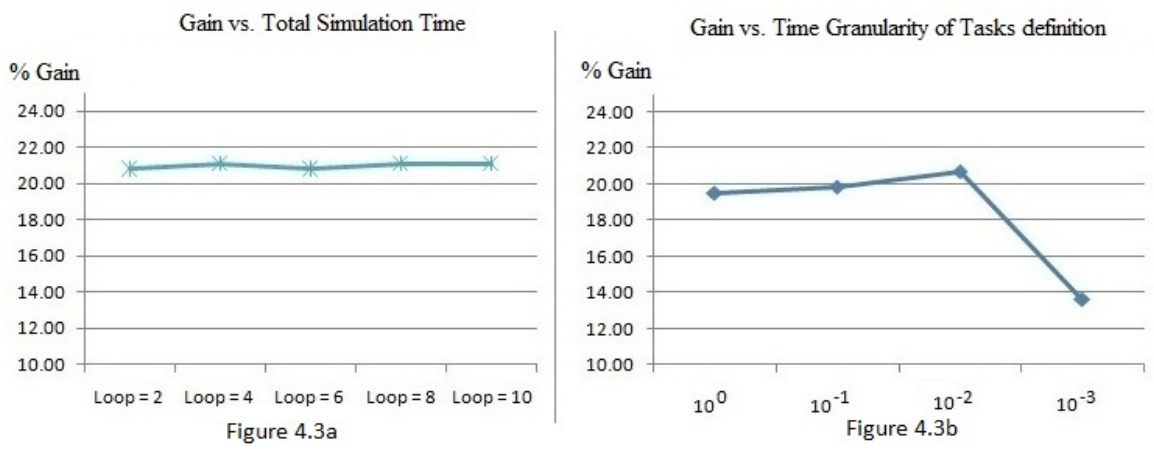

Figure 4.3: Percentage of Energy Gain vs. Application parameters.

Results in figure 4.3a show that although the total energy consumption with and without the DSF strategy changes, however, the percentage of energy gain is not affected by the different values of the *loop* variable. We then analyze the effect of decreasing the time granularity of tasks on the efficiency of DSF. Figure 4.3b shows that when the execution time of tasks is close to a certain limit (10^{-3}s), the percentage of energy gains decreases significantly. In other words, the strategy becomes inefficient and this is clearly because the delays for changing processor frequency, which is typically in the order of magnitude of a few hundred of microseconds. Task's definitions below a certain time limit (i.e. 10^{-3}s) provide abnormal scheduling behavior, as well as false energy consumption. However, the behavior of the DSF strategy can be considered as correct for values of time granularity higher or equal to 10^{-2}s.

In the following DSF experiments, we choose different values for simulation time (different values for the *loop* variable) and consider task granularities higher than 10^{-2}s in order to neglect the delays of changing frequency.

4.3 Results and Analysis

4.3.1 Multiprocessor Energy Savings

We experiment with four other application examples to analyze the energy savings and to verify the correctness of DSF execution on the multicore platforms. The results of energy gains for the different applications (i.e. *Example 2, Example 3, Example 4 and H.264Encoder*) using DSF strategy are shown in figure 4.4. We use distinct configurations of the platforms for each application as stated in section 4.1.2. We implement the examples using only the QEMU based platforms as the ARM1176JZF-S is not a multiprocessor platform.

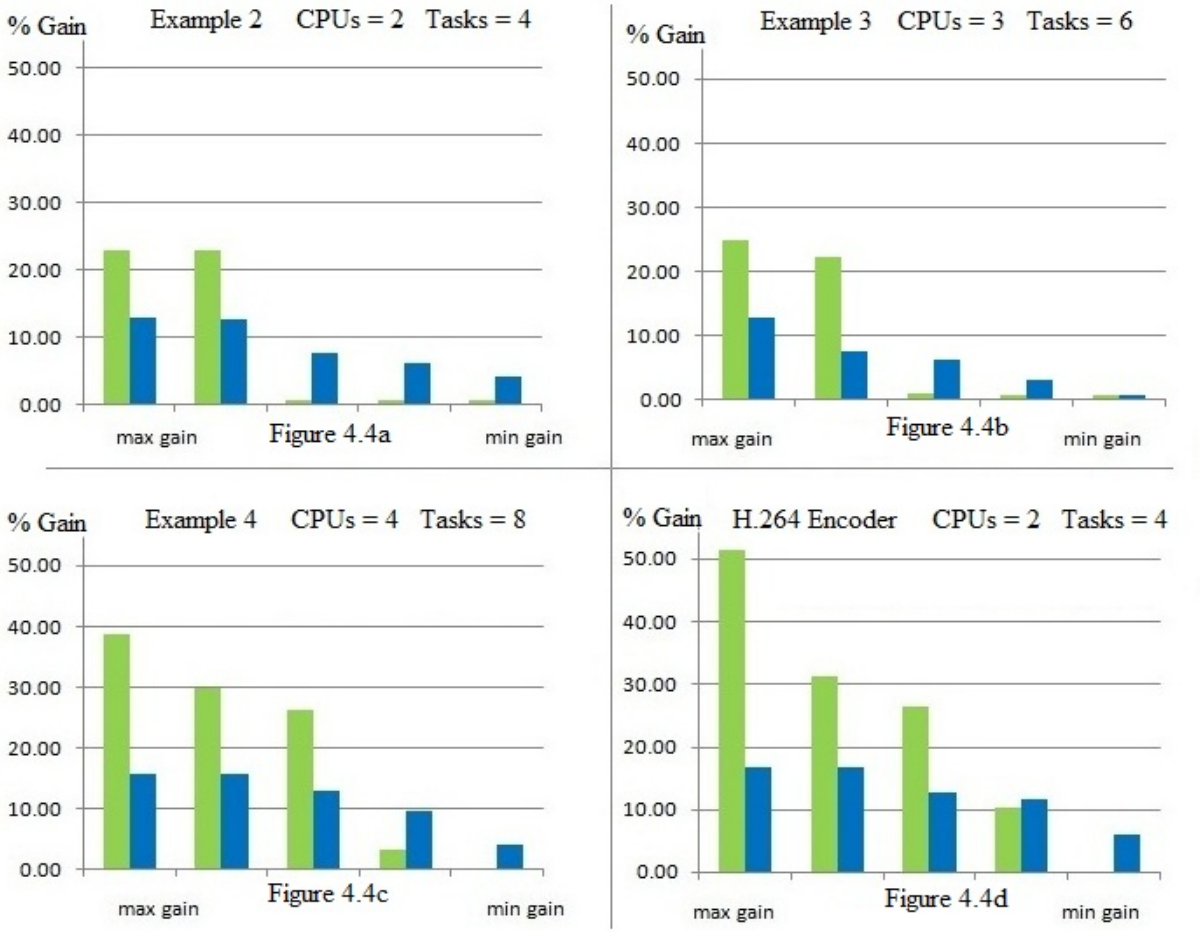

Figure 4.4: Energy gains on QEMU_ARM1176 (blue) and QEMU_CortexA9 (green) platforms for different applications and platform configurations.

Figure 4.4a shows energy gains of *Example 2* varying between 3.99% and 12.89% for QEMU_ARM1176 (blue) and between 1.96% and 22.93% for QEMU_CortexA9 platform (green). Figure 4.4b shows the energy gains of *Example 3* ranging from 3.08% to 12.69% for QEMU_ARM1176 and 0.65% to 24.92% for QEMU_CortexA9 platform. Energy gains for *Example 4* is between 4.16% and 15.76% for QEMU_ARM176 and between 3.12% and 38.59% for QEMU_CortexA9 as shown in figure 4.4c. Figure 4.4d shows the results of the DSF strategy on the H.264 video encoder model with energy savings ranging between 5.93% and 16.70% on QEMU_ARM1176 and between 10.21% and 51.46% on QEMU_CortexA9 platform.

The scheduling and performance of the applications, with and without the DSF strategy are correct on both platforms. However, it should be noted that we have bigger percentages of energy gains for QEMU_CortexA9 compared to QEMU_ARM1176. This behavior is present for all five examples. The next section explores this effect in detail for these platforms.

4.3.2 Analysis of Results

The different energy gains for QEMU_ARM1176 and QEMU_CortexA9 are strongly related to the operating points of these platforms. The power levels of each frequency with and without load are given in table 4.3 for each platform. The differences in power levels between frequencies are quite different on these platforms and this impacts the energy efficiency of the power strategy. DSF strategy allows dynamically changing between maximum and minimum frequency values depending upon the slack produced by a previous task and the WCET of the next task (see section 4.1.1). The different power consumptions of distinct platforms have power level gaps between consecutive operating points that are not the same. This gap plays an important role in energy consumption for these platforms. For instance, when the frequency is downscaled on QEMU_CortexA9 from 1000 MHz to 300 MHz (i.e. maximum to minimum), we can clearly see that the power gap is 125 mW (Load = 177, Idle = 52). However, the same power gap for downscaling the frequencies from 265 MHz to 160 MHz (maximum to minimum) on QEMU_ARM1176 is 51 mW (Load = 80, Idle = 29). Hence, a larger power gap provides more energy savings and that is why QEMU_CortexA9 to have significantly bigger gains as compared to QEMU_ARM1176 platform.

Table 4.3: Power profile of QEMU_ARM1176 and QEMU_CortexA9 platform.

Platforms	Frequency MHz	Power (Idle) mW	Power (Load) mW
QEMU_ARM1176	160	223	250
	215	238	290
	240	245	310
	265	252	330
QEMU_CortexA9	300	38	143
	600	60	215
	1000	90	320

Another effect seen in figure 4.4 is higher energy gains for *H.264Encoder*. The reason of different energy gains is due to the downscaled frequency by an application on these platforms. The required frequency is chosen from the available frequencies of the platform. If the calculated frequency is a value between the two available frequencies, the one with the higher value is chosen to provide deadline guarantees and this can cause different energy consumption for different applications. As an illustration, if we precisely analyze the scheduling and frequency switches of *Example 2*, we observe that the processor frequency downscales to a minimum value of 160 MHz in case of QEMU_ARM1176. However, the same *Example 2* downscale the frequency to 600 MHz for QEMU_CortexA9. Therefore, we conclude that we can have a further margin of energy gains for some applications for QEMU_CortexA9 that can downscale the frequency to 300 MHz. This is the case of the *H.264Encoder* example, where we have larger gains in comparison to *Example 2* (Figure 4.4*)* for the same number of processors.

4.4 Conclusion

We have presented and analyzed the effectiveness of a DVFS based low power scheduling strategy on various applications (including video encoding) on different platforms. The results of the DSF strategy on real ARM1176JZF-S and the virtual prototype of ARM11 (QEMU_ARM1176) provided correct scheduling and execution with a negligible deviation of 0.3% of energy consumption and provided energy gains around 18%. We have also validated the efficiency of DSF strategy on multicore platforms (i.e. QEMU_ARM1176 and QEMU_CortexA9) with significant energy gains ranging between 0.65% and 51.46% under different operating conditions. In addition,

the experimentations revealed the limits of the DSF scheduler that it is efficient for execution time granularity of greater than 10^{-2} seconds due to switching delays.

Furthermore, we have lightened the important effect of operating points that influence greatly the amount of energy gains. The experimentation pointed out that the CortexA9 model due to efficient power consumption (low Idle vs. load power) provided more energy savings in comparison to the ARM11 model. Therefore, we can conclude that efficiency of any DVFS based strategy is highly dependent upon the Idle vs. load power levels of a processor. Besides, our experimentations also indicated the dependence of energy gains on application parameters, as larger dynamic slack provided higher energy savings. In addition, we also pointed out that the energy gains of a particular application on two different platforms can be different due to distinct values of available frequencies.

In the next chapter, we target a DPS based strategy in order to check the conditions of its applicability on real systems, and eventually compare the efficiency of DVFS vs. DPS based strategies.

Chapter 5 : ASDPM POWER STRATEGY

5.1 Introduction

This chapter focuses on the analysis of a dynamic low power strategy called Assertive Dynamic Power Management (AsDPM) on ARM platforms. The AsDPM strategy considers mainly the processors for power and energy consumption optimization during the execution of a certain application. It works on the principle of admission control for ready tasks, by delaying the execution of ready tasks as much as possible. This controls the maximum number of active/running processors in the system at any time instant. The availability of ready tasks during the execution of a program is random. The choice to when exactly a ready task is executed on certain processor and how many processors are required for the remaining tasks can save a significant amount of energy consumption. This chapter focuses on the energy efficiency of the AsDPM strategy for real-time tasks, which decides when exactly a ready task shall execute thereby reducing the number of active processors, which eventually reduces energy consumption.

AsDPM technique is shown to be able to bring significant energy savings, while satisfying real time constraints for different applications in embedded systems. The strategy is particularly designed to be used in a multiprocessor environment, therefore we experiment it using the QEMU based multicore platforms. Several example applications are experimented on both QEMU_ARM1176 and QEMU_CortexA9 platforms to analyze the energy savings and to explore the efficiency of the AsDPM strategy. Moreover, we analyze the results of energy gains from the experimentation of the AsDPM power strategy in different platform and application configurations. In addition, we compare the AsDPM and DSF strategies for the same set of examples to provide an effective evaluation. For this reason, we initially used same examples defined previously in chapter 4, unfortunately the examples do not provided enough results. Therefore, we define new set of examples so that we have a better evaluation and detailed energy analysis of these strategies for each distinct platform configurations. By this way, we are also able to provide several conditions for the applicability of the DVFS and DPS strategies.

We experiment and analyze the AsDPM multiprocessor strategy in the following manner. Section 5.1.1 describes the AsDPM strategy in detail. Section 5.1.2 provides information about the examples used for the experimentation. Section 5.2 is divided into three parts: (a) section 5.2.1 explains the implementation of the AsDPM strategy, (b) section 5.2.2 focuses on the energy savings provided by the AsDPM strategy and (c) section 5.2.3 presents an analysis of results and energy saving conditions. In section 5.3, we compare the results of energy gains of the AsDPM strategy with the DSF strategy. Section 5.3.1 presents the results of energy gains by the DSF strategy for the new set of examples. In section 5.3.2, a comparison of results for DSF and AsDPM strategies for both QEMU_ARM1176 and QEMU_CortexA9 are given. At the end, we present our conclusions in section 5.4.

5.1.1 AsDPM Strategy

AsDPM strategy is a DPS based low power strategy in which the required number of processors depends upon the amount of remaining tasks and their deadlines. AsDPM technique exploits the idle time intervals within an application to shutdown or idle the processors. The strategy is based on the algorithm defined in [58]. At the start, all tasks within an application are sorted according to their priority. The strategy then performs a Laxity Bottom Test (LBT) at every scheduling event, starting with the assumption that at most one processor is running to accommodate most of the workload (tasks) and gradually increases the computational resources. Here, laxity within context of AsDPM is the anticipative laxity of a task's job and is the measure of its urgency to execute relative to its deadline, in the presence of all higher-priority released job(s) running and deferred on a particular processor. The highest priority tasks are assigned to the required number of processors. For the rest of the ready tasks, LBT is performed considering the first target processor. If a task passes LBT, it is deferred from execution at current scheduling event. Otherwise, if a task does not pass LBT then it implies that currently available running processors are not sufficient to satisfy the concurrent resource requirement of ready tasks and some tasks may miss their deadline in future. In this case, all tasks which are deferred or running are put into ready task queues again and more processors are activated. This procedure is repeated until all tasks are moved to the running task queue. Upon the arrival of the next scheduled event, the same process repeats itself and as a result the number of active processors may change.

Finally, if an application requires only one processor, the second highest priority task is executed on the same processor after finishing the first one. If an application requires two processors, the first higher priority task is executed on the first processor, the second priority task on second and similarly the process goes on until the completion of the remaining tasks. By this way we execute the application on the least number of processors required, hence minimizing the total energy utilization. At the end of application's execution the higher priority tasks finish earlier; this means that an executing task on the first processor completes earlier than the remaining tasks. Therefore when a scheduling event occurs, the task on second processor is moved to the first processor for its completion. Hence we minimize the total number of processors needed for the execution of an application after each scheduling event as well as at the end.

5.1.2 Application Examples

Based on its definition, the multiprocessor AsDPM strategy needs a platform with at least two processors to be applicable. We therefore need at least three different examples for the detailed experimentation and analysis to use three possible distinct configurations for the QEMU platforms. Beside these examples, we also experiment the AsDPM strategy with an application to H.264 video encoder model to evaluate its energy consumption. The main parameters used in these examples are listed in table 5.1.

Table 5.1: Examples used for the experimentation of AsDPM strategy.

	Example 5			*Example 6*			*Example 7*		
Tasks / Period	4 / 20			6 / 22			8 / 34		
	WCET	BCET	D	WCET	BCET	D	WCET	BCET	D
T1	6	3	8	7	2	9	5	2	9
T2	7	4	8	6	3	9	6	3	9
T3	4	2	14	5	4	9	7	4	9
T4	5	3	20	2	2	18	8	5	9
T5				3	3	20	3	2	13
T6				4	4	22	4	3	20
T7							5	3	26
T8							6	4	34

Note: Here time is in order of 10^{-1} seconds; D = Deadline.

Example 5 is used to experiment with the AsDPM strategy using a maximum of two processors. It consists of four tasks T1, T2, T3 and T4 with the parameters specified in table 5.1. These parameters are defined in a way that the first tasks T1 and T2 need two processors for their execution. This is controlled by defining deadlines that are different from their periods and where the total sum of their BCET is greater than their period. As an illustration, the tasks T1 and T2 are defined with a deadline for T1 = $8x10^{-1}$s and T2 = $8x10^{-1}$ s. This forces their completion time to be 8 x 10^{-1} seconds before completion of their period of $20x10^{-1}$s. Hence *Example 5* requires at least two processors before the start of the next period for its execution. To have a wide range of energy savings, we use the same procedure as we used in the previous experimentation of the DSF strategy. The AET of tasks T1 and T2 are set to different values between their BCET and WCET, while fixing the AET of T3 and T4 to an average between their BCET and WCET. By this way, we obtain a wide range of the energy gain spectrum for distinct values of AET. It should be noted that, we can obtain maximum and minimum energy gains just by using the maximum and minimum values of AET. However, the different values of AET are later used to compare the energy gains obtained by DSF and AsDPM strategies for a common set of examples.

Example 6 contains six tasks and is used with a maximum of three processors, and *Example 7* contains eight tasks and is used with a platform configuration of four processors. In *Example 6* we allow the tasks T1, T2 and T3 to change their AET, while in *Example 7* we allow tasks T1, T2, T3 and T4 to have variable AET. We use the same *H.264Encoder* model (see section 4.2.1) which consists of four tasks T1, T2, T3 and T4, where T1 and T2 are tasks used for motion estimation. T3 is used for intra prediction, texture encoding and syntax writing. T4 is a post processing filter. The first two tasks should be completed in parallel, however tasks T3 and T4 can be completed afterwards. The tasks T1 and T2 have deadline of 21 ms, T3 has a deadline of 31 ms and T4 has a deadline of 40 ms. The period is fixed to 40 ms for all tasks.

In the next section, we present the implementation and experimentation of the AsDPM strategy using the above-mentioned examples.

5.2 AsDPM Implementation and Experimentation

Here, we describe an actual implementation and experimentation of the AsDPM strategy using different multiprocessor platforms. Section 5.2.1 presents the implementation and execution correctness of the AsDPM strategy using *Example 5*. Energy savings (using all previously described application examples) for distinct platforms configurations are shown in section 5.2.2 and in section 5.2.3, we present a comparative analysis of energy savings for ARM1176 and CortexA9 platform.

5.2.1 AsDPM Implementation

The implementation of AsDPM is based on a similar principle as DSF, using a Linux userspace scheduler [53]. This scheduler is used to actually execute the strategy on any multiprocessor platform supporting Linux and C-States. For our experimentation, we use *AsDPM_scheduler* containing the code for the AsDPM strategy. The experimental procedure consists in running the scheduler on the previously described examples in different configurations, using the platform specific power monitoring drivers (*PM_Driver*) already developed in order to trace power consumption and derive energy values. The AsDPM strategy uses the C-states (instead of P-states), therefore we modify the *PM_Driver* function to allow the use of C-states. *PM_Driver* is therefore used here to change the required processor state to one of the following C-states: *Idle, Sleep, Running*. The *AsDPM_Scheduler, PM_Driver* and the test application are placed in the shared *Arm* folder of QEMU platforms.

We start experimenting with the AsDPM strategy using *Example 5* to verify its execution correctness on QEMU_ARM1176 platform. The AET of tasks T1 and T2 of *Example 5* are set to different values as stated in section 5.1.2, to verify the scheduling behavior and obtain a wide range of the energy gain. The results of energy consumption with and without the AsDPM strategy and the resulting percentage of energy gains are shown in table 5.2. Similarly, we test the correctness of the AsDPM using *Example 5* on QEMU_CortexA9 platform. Table 5.2 shows the results of this implementation for both platforms using maximum of two processors.

Table 5.2: Energy consumption of *Example 5* on QEMU platforms.

	QEMU_ARM1176			QEMU_CortexA9		
AET (10^{-1} sec)	E1 (mJ)	E2 (mJ)	% Gain	E1 (mJ)	E2 (mJ)	% Gain
3, 4, 3, 2	7482.23	4888.07	34.67	4236.11	3045.4	28.11
4, 5, 3, 4	7596.38	5212.34	31.38	4692.85	3734.38	20.42
4.5, 5.5, 3 4	7650.37	5332.44	30.30	4867.97	3990.64	18.02
5, 6, 3, 4	7655.37	5440.56	28.93	4965.08	4146.9	16.48
5.5, 6.5, 3, 4	7666.35	5560.63	27.47	5140.01	4327.21	15.81
6, 7, 4, 5	7674.39	5764.80	24.88	5440.53	4755.96	12.58

Note: E1 = Energy without AsDPM strategy, E2 = Energy with AsDPM strategy

The correctness of scheduling and coherence of energy consumption results have been checked for both platforms. The scheduling behavior and the energy consumption with and without the AsDPM strategy proved to be correct. In addition, the experimentation allow the energy savings for *Example 5* on both platforms. The percentage of energy gains lie between 24.88% and 34.67% on QEMU_ARM1176 platform, and between 12.58% and 28.11% on QEMU_CortexA9 platform. In the next section, we provide detailed results of energy gains by applying AsDPM strategy on the other application examples.

5.2.2 Energy Savings

The results of energy savings for *Example 6* using a maximum of three processors and *Example 7* using four processors are shown in figure 5.1 and figure 5.2 respectively. Figure 5.1 shows the percentage of energy gains for *Example 6* ranging between 38.65% and 49.88% for QEMU_ARM1176 and between 20.51% and 40.55% for QEMU_CortexA9 platform. The energy savings are maximum where AET of tasks is taken equal to their BCET and minimum when AET of tasks is taken equals to their WCET. This point out that AsDPM strategy provides higher gains for a larger slack produced by the different of their AET and WCET. Moreover, the AsDPM strategy also

provides energy gains for AET of the tasks equal to WCET due to the least number of processors used for application execution.

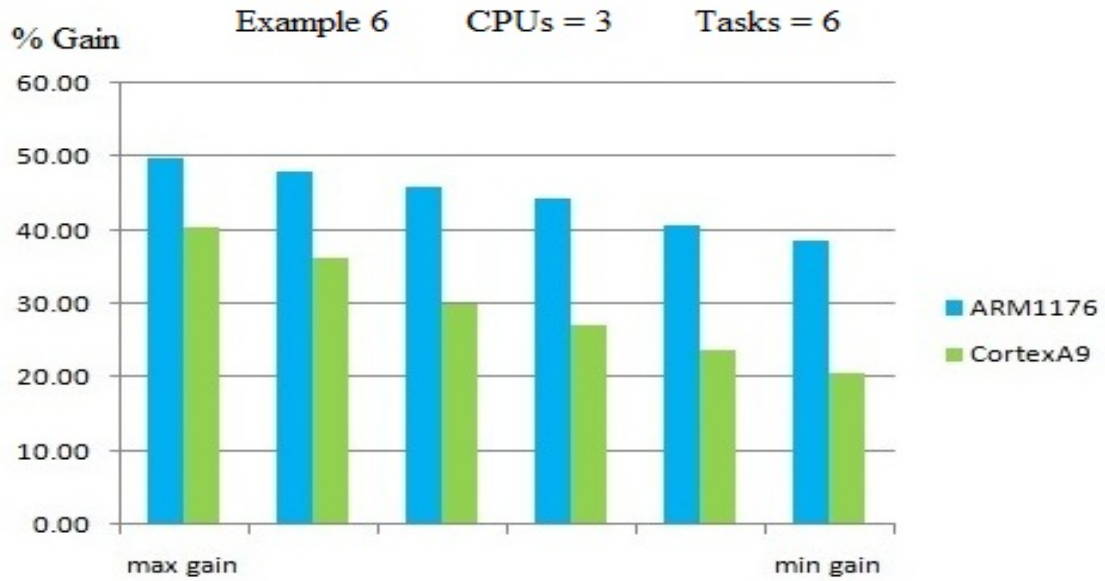

Figure 5.1: Energy gains of Example 6 on QEMU platforms.

Figure 5.2 shows the results of *Example 7* having energy gains ranging between 50.64% and 59.18% on QEMU_ARM1176 and between 30.95% and 47.69% for QEMU_CortexA9 platform. Again, we have maximum energy savings for larger slack. These examples show the ability of AsDPM strategy to actually save energy in configurations up to 4 processors for both platforms. In each case, we have also verified the correctness of application scheduling with and without the AsDPM strategy.

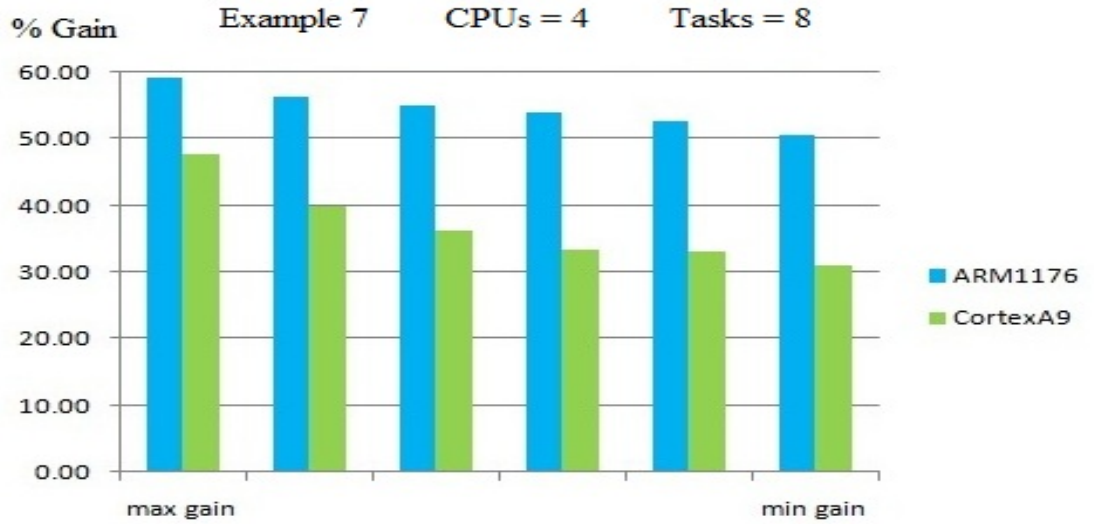

Figure 5.2: Energy gains of Example 7 on QEMU platforms.

The AsDPM strategy is then applied to the video *H.264Encoder* example on both platforms. We set six different values of AET for the Tasks T1 and T2 (presented as $Config_n$ in table 5.3) to obtain different energy consumptions for the *H.264Encoder* example. At start, the *H.264Encoder* requires two processors for the execution of tasks T1 and T2, while the remaining tasks T3 and T4 need one processor for their completion when scheduled by the *AsDPM_scheduler*. The second processor is idled depending upon the value of slack produced by the tasks T1 and T2. Figure 5.3 shows the result of energy gains obtained from this implementation where energy gain for *H.264Encoder* ranges between 24.05% and 46.73% on QEMU_ARM1176 and between 15.32% and 42.72% on QEMU_CortexA9 platform.

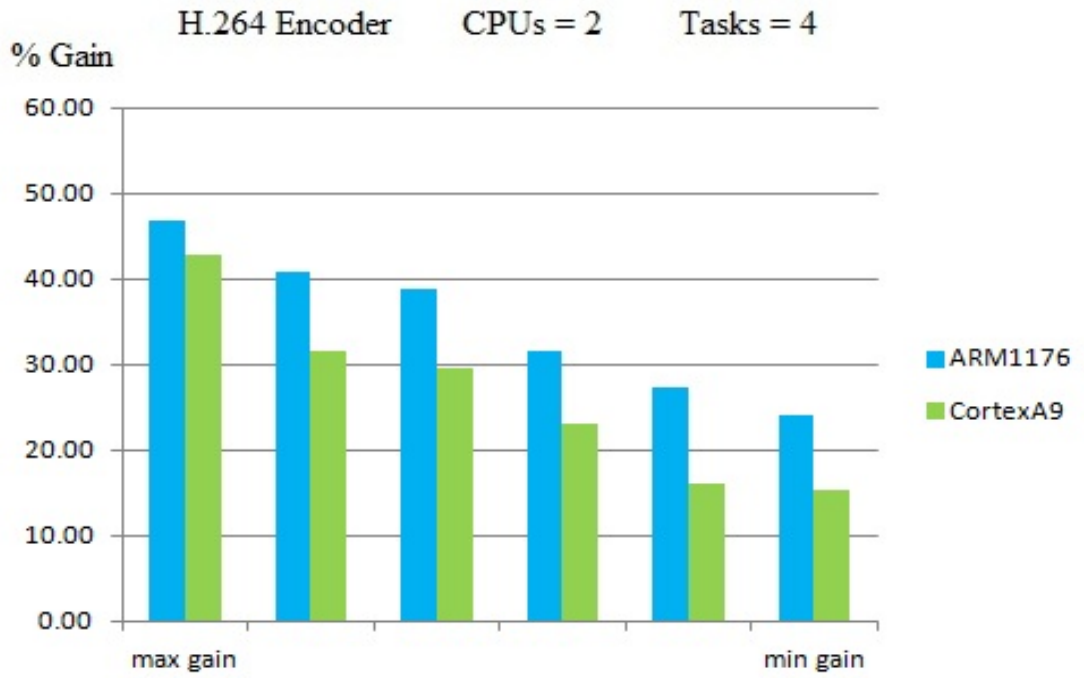

Figure 5.3: Energy gains of *H.264Encoder* on QEMU platforms.

From the above results, we can assume that ARM1176 consumes less energy as it provides more energy savings, however, it is not the case. In fact what is expected is that the CortexA9 consumes less energy, and actually, it does consume less as pointed out by the results of *Example 5* in table 5.2 and *H.264Encoder* in table 5.3. As an illustration, the energy consumption of *H.264Encoder* example (*Config 1,* Table 5.3*)* with and without AsDPM is 3915 mJ and 7350 mJ respectively on QEMU_ARM1176 and the same energy consumption is 2909 mJ and 5080 mJ on QEMU_CortexA9. As a matter of

fact, this is true in case of all examples used for experimentation that CortexA9 consumes less energy than ARM1176 (which is what we expect from smaller scale technology integration). What we additionally remark on the other hand, is that the energy gains resulting from using AsDPM are more important on the ARM11 than on CortexA9. This is further analyzed in the next section.

Table 5.3: Energy Consumption and Gain for *H.264Encoder.*

	QEMU_ARM1176			QEMU_CortexA9		
AET (10^{-1} sec)	E1 (mJ)	E2 (mJ)	% Gain	E1 (mJ)	E2 (mJ)	% Gain
Config 1	7350.12	3915.76	46.73	5080.23	2909.96	42.71
Config 2	7866.55	4660.13	40.76	5746.56	3936.97	31.49
Config 3	7922.68	4851.23	38.77	5956.56	4202.35	29.45
Config 4	7974.64	5458.08	31.56	6185.15	4754.58	23.13
Config 5	7996.6	5816.76	27.26	6291.68	5291.3	15.9
Config 6	8022.99	6092.99	24.05	6461.38	5471.49	15.32

5.2.3 Further Analysis of Results

The higher percentage of energy gains in case of QEMU_ARM1176 in comparison to QEMU_CortexA9 platform is related to the operating points of these platforms. We implement the AsDPM strategy using the maximum frequency of the platforms. Table 5.4 shows the idle vs. load power levels for both platforms. As the Cortex based platform uses a smaller recent integration technology (45 nm), we should expect a higher proportion of idle power consumption (related to static power) for the CortexA9. But it can be noted here that surprisingly, the idle power of the ARM1176 is much more important (252 mW instead of 90 mW for the CortexA9). This is due to the same reasons explained in chapter 3 concerning the inefficiency of operating points. The ARM1176JZF-S Versatile Baseboard is a development platform used for early prototyping that has important performance limitations. This is why the maximum frequency is limited only to 265 MHz while a production device would be able to run at 800 MHz. Given this, the results of table 5.3 show a higher share of energy gains for QEMU_ARM1176 which is not necessarily representative.

Table 5.4: Idle vs. load power levels for QEMU platforms at maximum frequency.

Platforms	Frequency MHz	Power (Idle) mW	Power (Load) mW
QEMU_ARM1176	265	252	330
QEMU_CortexA9	1000	90	320

In fact, AsDPM, and DPS based techniques in general, should be more efficient for small factor integration technologies as they address static power by switching off some processors. This shows again the importance of platform characteristics on the efficiency of power strategies. In the following, we also check the efficiency of DPS to address static power consumption by comparing the energy savings to a DVFS strategy.

5.3 Energy Gain Comparison of DSF and AsDPM

5.3.1 Energy Gains for DSF

We have set four application examples (*Example 5, Example 6, Example 7 and H.264Encoder*) to be common with AsDPM to compare the two strategies. The examples exhibit different workload (number of tasks) vs. processing power (number of processors) in order to span different use cases. We therefore start by implementing the DSF strategy on these set of examples with platform configuration having two, three and four processors. The AET of the tasks are varied between their BCET and WCET to observe energy savings for these examples.

Figure 5.4 a), b), c), and d) show the results of energy savings using respectively application (*Example 5*, *Example 6*, *Example 7* and *H.264Encoder)*, where figure 5.4a reports energy gains ranging from 3.34% to 9.50% for QEMU_ARM1176 and from 2.82% to 12.73% for QEMU_CortexA9 platform. Figure 5.4b shows percentage of energy gains for *Example 6* ranging between 1.58% and 13.59% for QEMU_ARM1176 and between 1.30% and 22.24% for QEMU_CortexA9. In figure 5.4c, the percentage of energy gains for *Example 7* lie between 2.44% and 10.44% for QEMU_ARM1176 and between 0.89% and 14.24% for QEMU_CortexA9. Figure 5.4d shows the result of energy gains for *H.264Encoder* example which lie between 6.01% and 16.72% for QEMU_ARM1176 platform and between 1.34% and 40.14% for QEMU_CortexA9 platform.

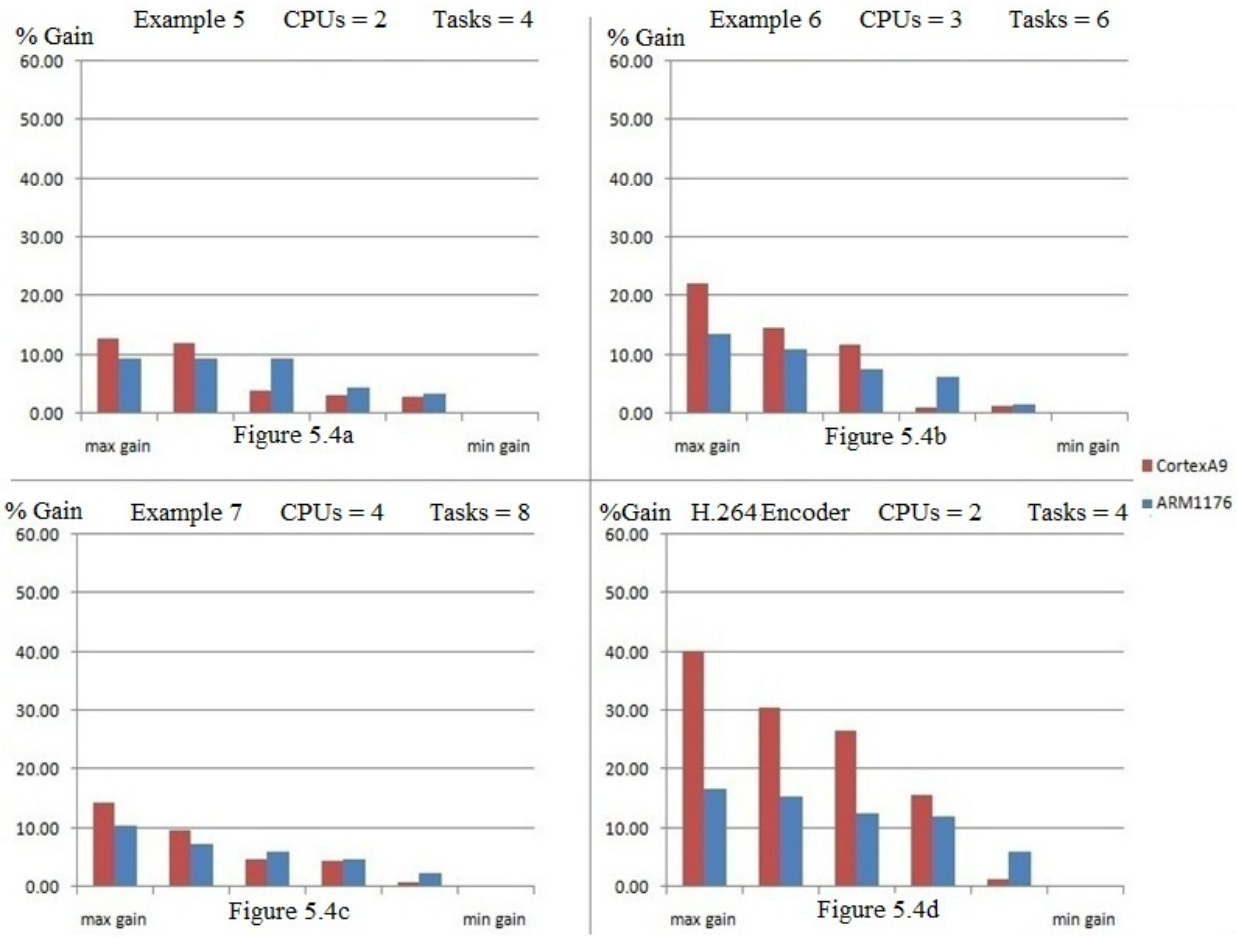

Figure 5.4: Energy Gains for DSF Strategy on QEMU platforms.

The energy gains are present in all cases, and are more important in case of the *H.264Encoder*. This is obvious as the slack produced by the tasks of *H.264Encoder* are much larger than other examples and allows the processors to downscale frequency to the minimum allowed value (as already observed in chapter 4). In addition, the energy gains for DSF strategy are higher on CortexA9 platform for all examples due to efficient operating points (as discussed previously in chapter 4). Furthermore, the amount of energy savings depends upon the dynamic slack and we can see from these results that the DSF strategy does not provide any energy gain when AET is set equal to WCET (min gain is zero).

In the next section we focus on the comparison and analysis of these examples using AsDPM results.

5.3.2 Comparison and Analysis of Results

Here, we provide the comparative results of AsDPM and DSF using the same set of examples considered previously in section 5.3.1. The respective results of *Example 5* (4 tasks, 2 CPUs), *Example 6* (6 tasks, 3 CPUs), *Example 7* (8 tasks, 4 CPUs) and *H.264Encoder* (4 tasks, 2 CPUs) are provided in figure 5.5 a), b), c), and d) for QEMU_ARM1176 platform and in figure 5.6 a), b), c), and d) for QEMU_CortexA9 platform.

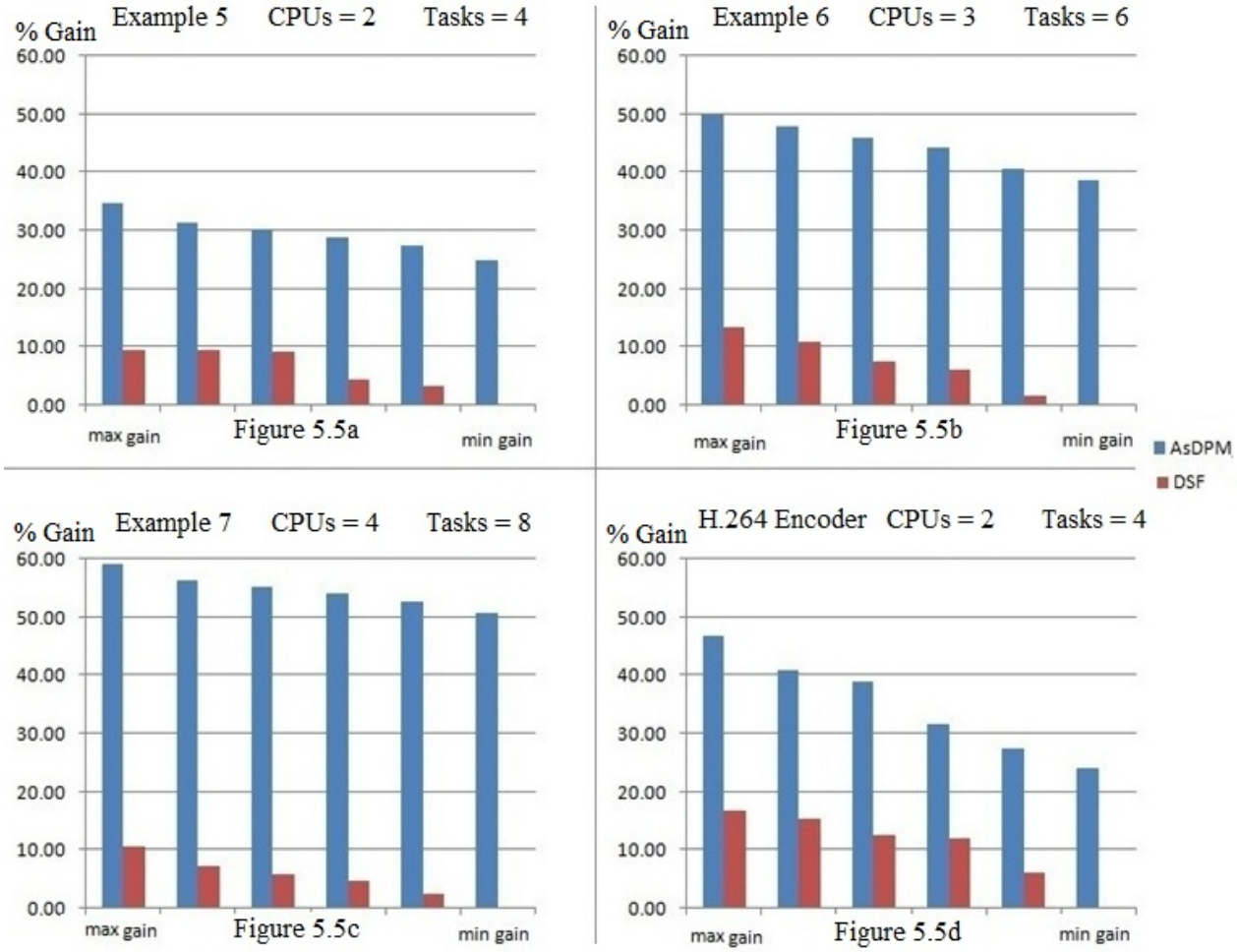

Figure 5.5: AsDPM vs. DSF energy gains for different examples on QEMU_ARM1176.

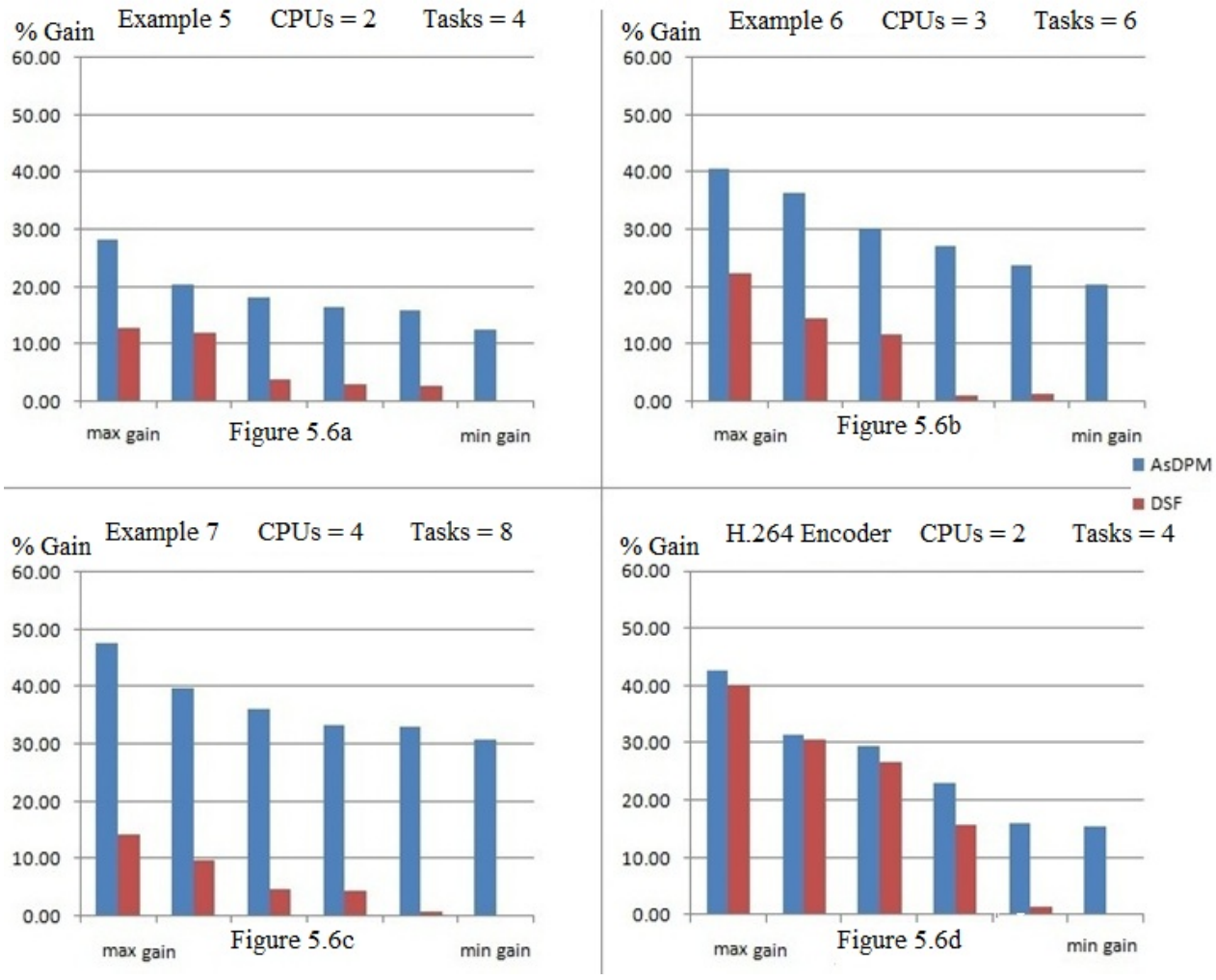

Figure 5.6: AsDPM vs. DSF energy gains for different examples on QEMU_CortexA9.

We can clearly remark that AsDPM provides higher energy gains than DSF strategy and this difference is significant for all applications. In addition, AsDPM also provides energy gains in cases where DSF does not provide any energy gain at all. Indeed, when AET of a task is set equal to its WCET, DSF strategy cannot save energy because there is no frequency switch in this case, however in same conditions, AsDPM provides significant energy gains because of fewer number of processors used for execution. Another reason of higher gains for AsDPM is due to the operating points of the platforms. QEMU_CortexA9 is representative of recent technologies (45 nm) as the

model is derived from load vs. idle power consumption measured on a real platform (ST-Ericsson-based Snowball [55]). QEMU_ARM1176 is not representative of a smaller factor technology, but the actual power levels of this platform (obtained from actual measures on a Versatile Baseboard ARM1176JZF-S) exposes a very high share of idle power, in the same way as a recent technology. This is due to different limitations inherent to the use of an evaluation platform (operating points, performances, and power levels), which is not a final production device. Both platform results are thus coherent with the assumption that DPS should be more efficient than DVFS to address recent technologies with an important share of static power consumption.

Nevertheless, DPS applicability is subject to conditions related to the application, in particular concerning tasks execution times that should permit to neglect delays for switching to processor sleeping states. This is why beside the above observations, we can notice from figure 5.6 that the difference of energy savings of *H.264Encoder* on CortexA9 platform is not too much for the DSF and AsDPM strategies.

5.4 Conclusion

We have presented and analyzed the effectiveness of AsDPM, a DPS based power strategy, on various applications (including video encoding) for different ARM based platforms. To provide a relevant characterization of energy gains, we have considered the execution of the different test applications on different multiprocessor configurations (up to 4 processors). The AsDPM strategy provides significant energy gains in all cases ranging between 12.58% and 60%. The comparisons of results on QEMU_ARM1176 and QEMU_CoretxA9 have also highlighted the effect of idle vs. active power on the level of AsDPM energy savings. These results further show the importance of platform characteristics (here, idle vs. load power levels) on the efficiency of a DPS strategy.

Finally, we have also compared the energy gains of DSF vs. AsPDM considering a common set of applications. The results have shown AsDPM outperforms DSF in terms of energy gains for all applications on both QEMU_ARM1176 and QEMU_CortexA9. As discussed in the analysis of results, this leads to expect AsDPM to be more efficient for systems based on recent integration technologies where static power is a significant part of total power. Nevertheless, as will be discussed in next chapter, DPS based

strategies are not always applicable in practice because of the latencies related to switching of idle states, especially when an application cannot afford large waiting time.

Chapter 6 : GLOBAL ANALYSIS AND CONCLUSION

This work explored power management issues and opportunities to manage efficiently the overall energy consumption of battery powered embedded systems. We addressed actual multicore platforms and analyzed in depth their energy consumption by providing a detailed experimentation and analysis of different domain or application specific power strategies. We have chosen an experimental approach based on representative multiprocessor platforms (real or virtual) to consider the real impact of different parameters such as the effect of supported frequencies, voltages, scheduling and transition delays, idle and load power, application execution time and workload. The following section presents global analysis and conclusion based on the rich results from this work. We can categorize these conclusions into two main areas. First, the actual implementation and experimentation approach enabled us to identify the important conditions from the hardware and application viewpoints that highly determine the efficiency of power management strategies. Second, we propose some high level models of the different strategies that have been used, in order to help designers as well as application developers to evaluate the benefits of power management at early stages of a system development, in the particular context of multiprocessor platforms.

6.1 Power Management Effectiveness

The following section describes platform and application conditions that have been identified to influence greatly the effectiveness of power strategies. Actually, three important conditions have been identified in our experiments: (a) the characteristics of operating points, (b) latencies of changing states, and (c) application level conditions. Then we present two additional considerations of power management effectiveness related to the use of domain specific strategies and the effectiveness of DVFS and DPS in saving energy. We discuss each of these points separately in the following sections.

6.1.1 Characteristics of Operating Points

The characteristics of operating points play an important role on energy consumption. The amount of energy savings of a given platform is highly dependent upon its supported frequencies and their corresponding voltage levels. In our experimentation, we observed two distinct effects of platform characteristics that are discussed below.

6.1.1.1 Operating Points Inefficiency

The majority of power strategies are based on the hypothesis that decreasing the frequency of processors actually results in saving energy. Although this fundamental assumption is verified on most platforms, in fact there are some cases where it is not. Decreasing frequency decreases power at which a processor operates but at the same time results in an increase of execution time. As energy is the product of power by time, energy savings are dependent on the condition that the proportion of power decrease is greater than the proportion of time reduction, which depends on the characteristics of operating points in terms of frequency values and associated power levels.

The experimentations of the DVFS based video strategy in chapter 3 have shown that using this strategy on the ARM1176JZF-S platform actually results in an increase of energy consumption, despite decreasing the frequency. Indeed the operating points of the ARM1176JZF-S platform do not allow to compensate the increase of execution time due to frequency downscaling, and therefore results in an increase of energy consumption. With the help of a virtual prototype, we have been able to modify the original power levels associated with each operating point, and we have verified that increasing the power level gaps results in effective positive energy gains. The experimentation has been conducted on a set of four configuration of operating points (Figure 3.5), where energy savings (Figure 3.7) are negative in case C*onfig1* and *ARM1176* whereas the energy savings are positive in case of *Config2* and *Config3*. This shows that some platforms are not relevant for actual efficient DVFS. The decrease of power levels between two consecutive decreased frequencies must be sufficient to compensate the increase of execution time and energy. In case of the ARM1176JZF-S platform, such inefficiency can be explained by the fact it is an early evaluation platform that does not operate at full performance potential. Typically, this platform is limited to 265 MHz while a production

platform runs at 800 MHz. At this frequency, operating points are more suited and can provide effective energy savings.

Although it is rather uncommon, the example of the ARM1176JZF-S platform is not an isolated example of suboptimal operating point characteristics. Actual platforms might present non-performing DVFS implementation that can be due to different reasons like poor voltage regulators efficiency, too important switching delay latencies, same level of voltage for different frequencies or, like previously, performance limitations due an early evaluation platform. Because of this, the actual efficiency of operating points in terms of power / frequency characteristics must be addressed before any power strategy is envisaged.

6.1.1.2 Operating Points Impact on Energy Savings

As discussed in previous section, operating points and more generally platform characteristics impose essential prerequisites that can prevent the actual effectiveness of any well-defined power strategy. These characteristics also naturally affect to a different degree the level of achievable energy savings. Power management policies usually account for supported frequencies but do not consider the related differences in power consumption. The power levels associated to the frequencies of a processor also play an important role in energy consumption. The power consumptions with and without load for a frequency vary from one platform to another due to different voltages and leakage power. We have shown two examples of this which result in different energy consumption for a same application.

The experimentations of the first power strategy for video have shown that the load power levels between two frequencies contribute directly to the amount of energy savings. Large differences in load power between two frequencies provide more energy savings, while smaller differences result in complete inefficiency of the strategy. This is shown by the results of table 3.2 where *Config2* and *Config3* have large load power gaps between different frequencies permitting energy reductions that exceed noticeably the increment due to execution time. However *Config1* and *ARM1176* having smaller power differences both result in an actual increase of energy consumption while the power strategy operates correctly by decreasing the frequency.

This impact of idle vs. load power levels of operating points on energy savings is also present in the experimentation of the DSF strategy in chapter 4. For the two platforms used, the differences of power level (load vs. load, idle vs. idle) for different frequencies are not the same. Hence, these differences affect the overall energy savings and experimentations show that the platform with the largest power gaps provides the highest energy gains. Table 4.3 shows for example that when the frequency (f_i) is downscaled from 1000 MHz to 300 MHz (maximum to minimum) on QEMU_CortexA9, the Load Power (LP) gap is 177 mW (LP_{f1} = 320, LP_{f2} = 143), and Idle Power gap (IP) is 52 mW (IP_{f1} = 90, IP_{f2} = 38). However, the same power gaps for the frequencies downscaled from 265 MHz to 160 MHz (maximum to minimum) on QEMU_ARM1176 are 80 mW (PL_{f1} = 330, PL_{f2} = 250) and 29 mW (NPL_{f1} = 252, NPL_{f2} = 223). Hence, larger power gaps on QEMU_CortexA9 provide more energy gains than QEMU_ARM1176 (i.e. maximum gain of 51.46% on CortexA9 and 16.70% on ARM1176).

The influence of operating points is also visible in the energy saving results of the AsDPM strategy. The determining factor here is the load power consumption associated to an operating frequency when processors are not idle. On both experimented platforms, these powers are not the same, therefore the resulting energy consumption differs and the amount of energy savings obtained by idling a processor also varies. For example in case of QEMU_CortexA9 platform, we have less overall energy consumption due to lower load power levels associated to the maximum frequency (320 mW at 1000 MHz) in comparison to QEMU_ARM1176 platform which consumes much more energy due to higher power levels (330 mW at 265 MHz). However, the resulting energy gains are higher for QEMU_ARM1176 than for QEMU_CortexA9 due to a wider margin of power saving when a processor is idle (see Table 5.4).

6.1.2 Latencies of Changing States

Ideal DVFS based policies are often based on the assumption that voltage/frequency values can be changed instantaneously. However in reality, it takes time to change the CPU frequency/voltage due to factors such as the internal PLL (phase lock loop) locking time and capacitances that exist in the voltage path. A frequency transition results in the processor core and shared cache being unavailable for a small period during the transition. A real time application may be sensitive to this period of unavailability,

especially if the processor is switching between frequency transitions at a high rate. The experimentation with DSF strategy have shown that decreasing the execution time of tasks below the order of a millisecond alters the effectiveness of the strategy very quickly. In other words, the strategy becomes inefficient when the execution time of tasks becomes close to the delays for changing a processor frequency, which is typically in the order of magnitude of a few hundred of microseconds. Therefore the latency of frequency switching of a platform should also be considered before implementing any DVFS based strategy. Frequency switching of more than several milliseconds may not always be used in case of demanding applications like real time systems and video decoding (typically 40 ms frame processing), for example. In addition, the experimentation with the DVFS video strategy has pointed out that increasing the number of frequency switching degrades the performance of the video strategy. Therefore, we must keep in view these latency constraints that might prevent the actual effectiveness of a strategy, and which effect is amplified with the number of frequency switching.

In case of DPS based strategies, the delays of CPU mode transitions are typically higher than those of changing frequency. This may prevent any DPS based strategy to be efficient in advanced and high performance applications, which is typical in real time processing.

6.1.3 Application Level Conditions

An important condition for the efficiency of a power strategy is its ability to exploit application variability. All the strategies we have considered depend on different application knowledge for energy savings. The amount of energy saved depends strongly on the value of different application parameters at run time, frame rate for the video strategy and slack time for DSF and AsDPM. The maximum energy gains will thus often rely upon the ability of a strategy to use the lowest processor frequencies. For the DVFS based video strategy, this will highly depend on the performance of the application implementation, as high video decoding frame rate permit lowest CPU frequency processing. Similarly in case of AsDPM and DSF strategies, the amount of energy savings depend upon the value of slack produced by the application actual execution time. Both strategies provide more energy saving when the application's tasks generate large dynamic slacks (i.e. when AET values are close or equal to BCETs).

Another condition affecting the energy and scheduling behavior of an application is the time definition of its tasks. Application time parameters are defined using different units of time according to the application domain or system requirements and two different applications can have different timing definition for their tasks. As stated in previous section, the effect of switching delays, either from changing a CPU frequency or idle state, can lead to ineffective results of power strategies. For example, if we consider the example of a real time sensor providing temperature readings of a chemical every 100 ms. Accordingly, a decision is taken by a boiler unit to control the input heat of the system to avoid overheating and damages. The system can allow a delay of few milliseconds, however a larger delay in decision may cause immediate damages. As deeper sleep state provides more delay in waking up from its state, therefore DPS based policies will be inefficient and do not provide the needed power management solutions. Hence, a DPS solution could better suit scenarios when, either there is a low delay for entering and exiting deeper sleep states, or an application can afford large waiting time with respect to wakeup time latencies. In general, if there is lower power consumption for an idle mode then there is less benefit in using DVFS, however spending longer time to idle may increase power consumption due to wakeup overhead. To operate properly, WCET of application's tasks must be large enough to be able to consider switching delays as negligible. Regarding our results, WCET higher than 10% of frequency switching delays are satisfying for an efficient implementation of DVFS and DPS based strategies.

6.1.4 Domain Specific Strategies

General purpose policies cannot always bring the maximum potential savings compared to domain or application specific strategies. A typical example is the one of video. In chapters 3, 4 and 5, the three power strategies used provide each significant energy gains: up to 52% for the DVFS video strategy, and up to 60% for DSF and AsDPM on a H.264 encoder model (CortexA9 based platforms). Using a typical workload based policy (such as Linux *OnDemand* for example), the power consumption would have been set to the maximum since video processing always represent a high workload demanding maximum processor frequency, for all the video processing duration.

This illustrates the usefulness of using domain specific low power strategies. General purpose strategies have the advantage of being applicable in all cases, but the

counterpart is that they do not always exploit all the energy saving potential. Sometimes they are even completely inefficient at saving power like in the example of video processing. We have shown on concrete examples that we can actually save energy up to 17% on the ARM1176 platform and 52% on the CortexA9 platform using DSF, up to 60% on ARM1176 and 48% on CortexA9 platform using AsDPM, which are two domain specific (real time scheduling) power strategies.

6.1.5 Efficiency of DVFS vs. DPS

Many works investigate the field of low power scheduling techniques but very few (none to our knowledge) have been faced with real implementation results, probably because of the very high complexity inherent to multiprocessor scheduler development. In this work, we have shown very realistic results based on a real scheduler prototyping under Linux based on the work described in [53], also developed by LEAT in the scope of the COMCAS project.

Under these conditions, the implementation and effectiveness of deadline scheduling has been shown in the real world to be possible and to provide actual energy savings. In addition, we have used the same set of application examples for DSF and AsDPM as much as possible which let us make an efficiency comparison of DSF vs. AsDPM. In the following, we present an analysis of both techniques in order to derive which of two techniques, and by extension which of DVFS and DPS, has the best ability to save energy.

6.1.5.1 DSF vs. AsDPM

In our experimentation context of application and multiprocessor platforms, AsDPM outperforms DSF in all configurations of measures. DSF provides maximum energy gains up to 17% for QEMU_ARM1176 and 40% on QEMU_CortexA9. On the other hand, AsDPM provides maximum energy gains of 60% on QEMU_ARM1176 and 48% on QEMU_CortexA9. This is a notable result, but the difference of energy gains between both techniques is not always very large. For instance, the DSF strategy provides energy savings of 40% for the *H.264Encoder* model on QEMU_CortexA9 platform, while AsDPM strategy provides energy savings of 42% for the *H.264Encoder* model on same platform. This shows that the difference of energy gains is not always significant and depends upon a lot of parameters such as platform, application and

strategy characteristics. Because of the variety of influential factors, it is difficult to generalize on the efficiency of DPS vs. DVFS techniques. Nevertheless, these results indicate that DPS based strategies are likely to save more energy than DVFS based strategies. We further develop this effectiveness analysis in the next section.

6.1.5.2 DVFS vs. DPS

Despite above conclusions, an important applicability criterion to consider relates to the latencies of switching states and operating points. The latencies of entering and leaving DPS states are usually higher than those of changing processor frequency with DVFS. This implies that DVFS strategies are usable in a larger number of applications than DPS based strategies. DPS usage is more constrained by the latencies of switching states as explained in section 6.1.3, and will not always be applicable depending on the time granularity of application's tasks. In addition, DVFS strategies are also limited by the latencies of switching frequencies, so there are some cases where both DPS and DVFS are likely to be inefficient. As stated in section 6.1.3, it is thus much recommended to investigate switching states latencies before any DVFS or DPS implementation decision.

A second consideration about DVFS vs. DPS effectiveness is related to the share of static power in recent integration technologies. DVFS addresses only dynamic power of a CPU, while DPS impacts also the static power with more or less deep sleeping states. DPS is thus a privileged solution for recent integration technologies where transistor leakage is significant. This is however confirmed by our results on two platforms that have power levels characteristics of a high share of static power, and where AsDPM has been more effective in all application cases.

6.2 Power Models

From the results of chapters 3, 4 and 5, we can derive some simple models that can help in estimating the benefits of these strategies on application energy consumption. The principle is to derive from the various measurement results energy saving curves depending on the strategy's respective leading application parameter (i.e. frame rate, slack time). In each case, the model is standardized in a common representation of percentage of energy gains versus the driving application parameter (varying from maximum to minimum).

6.2.1 DVFS Video Strategy

The power model of the DVFS video strategy is derived from the achieved energy gains reported in chapter 3 which depend on the frame rate constraint that is set by the user.

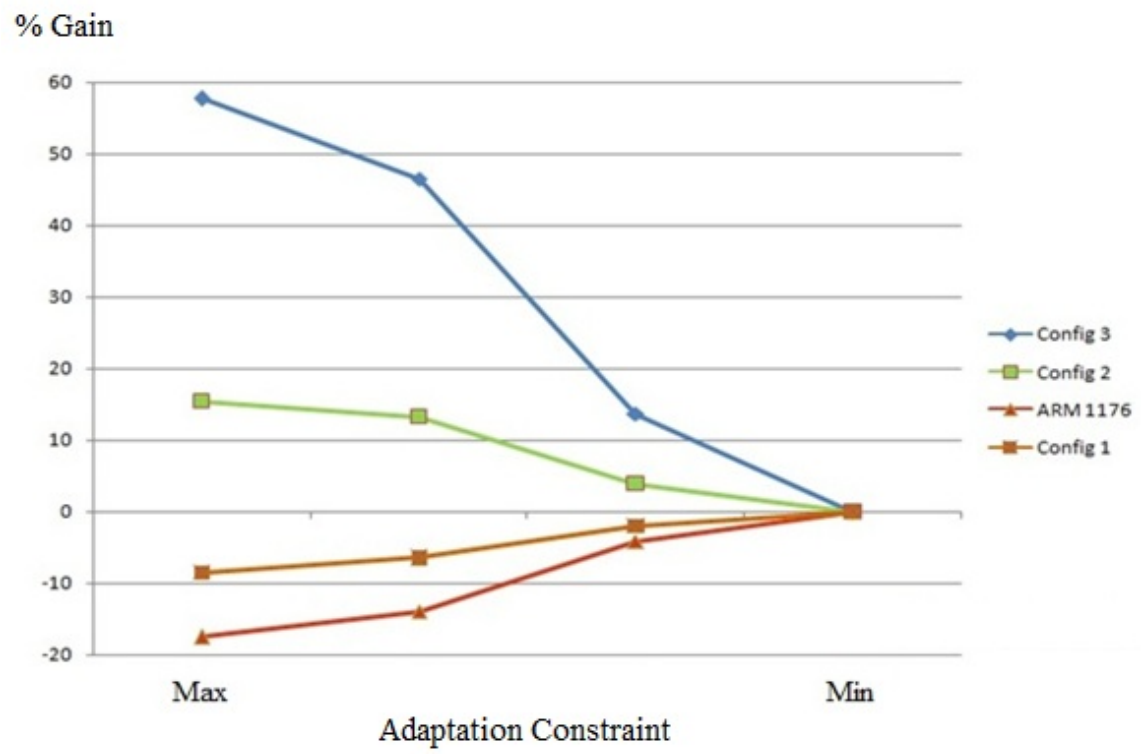

Figure 6.1: Energy Model for DVFS based Video Power strategy.

Figure 6.1, obtained from the measures of chapter 3, represents the range of energy gains from maximum (processor always operating at minimum frequency) to minimum (frame rate set to the average decoder speed). Since different configurations of operating points were evaluated, this representation can be used to quickly derive an estimation of energy savings for different characteristics of operating points, using a simple interpolation. It is thus also easy to know the benefits of using the DVFS video strategy and the amount of energy gain to expect given a level of performance of the application, here a H.264 decoder.

It should be noted that the energy savings reaches a certain threshold (i.e. 4 fps in this case), as the DVFS strategy can only downscale the frequency to a certain allowed limit. In addition, if user needs maximum performance then the adaptation constraint is set equal to the average video decoding speed at nominal conditions (i.e. 11 fps in this case) but there will be no energy saving in such case.

6.2.2 DSF

The energy model for the DSF strategy is derived from the results of chapter 4. It is based on averaging the different results of energy savings on all application examples and for both experimentation platforms. The two corresponding energy models are thus shown in figure 6.2 reflecting the level of energy saving according to the main driving application parameter, here the slack time.

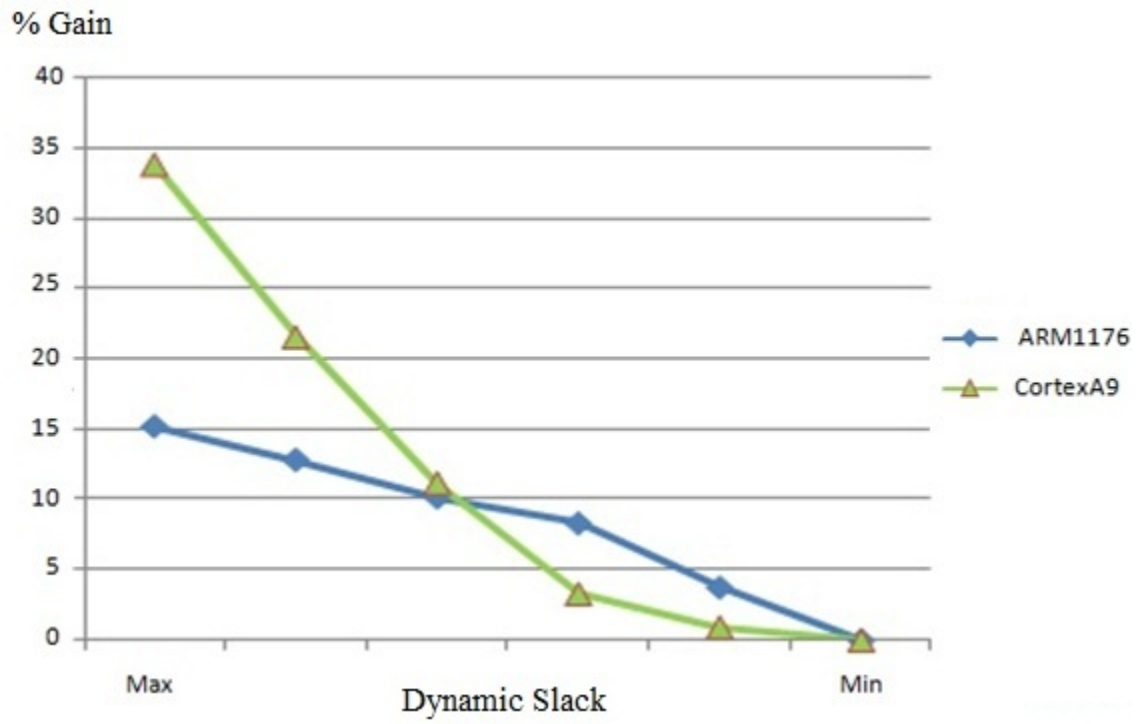

Figure 6.2: Energy Model for DSF power strategy.

The maximum energy gains correspond to a situation where 100% of the application slack time can be exploited to run processors at their minimum frequency, while the minimum correspond to the application running at its WCET (implying maximum

processors frequency in a DSF strategy) corresponding to no energy saving. This simple characterization is both realistic (as based on actual measurements) and useful to provide early estimation of the benefits for power strategies developing efforts.

6.2.3 AsDPM

Similarly, an energy model for the AsDPM strategy is derived from the average values of energy savings in the measures of chapter 5. The corresponding model of expected energy gains is shown in figure 6.3 for the entire possible range of dynamic slack, for both platforms experimented.

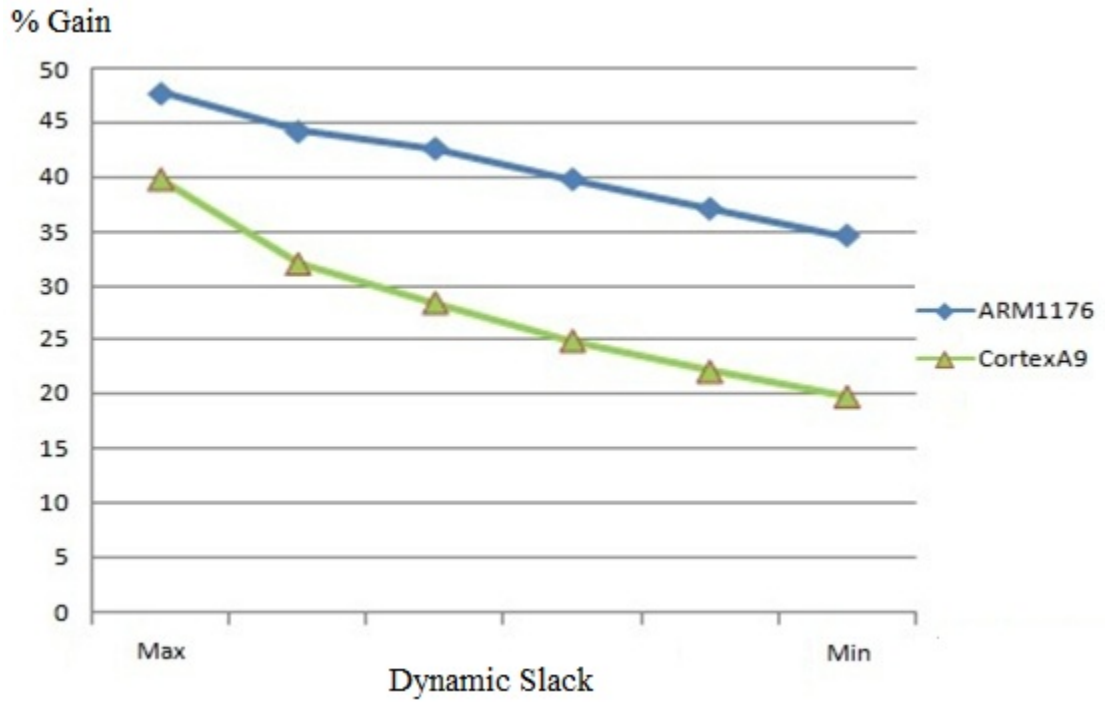

Figure 6.3: Energy Model for AsDPM strategy.

Like DSF, maximum energy gains correspond to the case where all application slack can be exploited to decrease processors execution at their minimum frequency. Conversely, the minimum energy gain is reached when no slack is produced by the application tasks, which means that their executions are close to their WCET. Here the minimum energy gain does not correspond to zero because even when there is no slack, processors are put to sleep (instead of idle) when they are unused. The energy gains at these boundaries correspond respectively to the maximum and minimum energy gains of the strategy and

for the target platform. From these curves, we can thus more easily apprehend the cost/benefit tradeoff resulting from using an AsDPM strategy for a given application/platform mapping.

6.3 Conclusion and Perspectives

A great part of our everyday life is somehow linked with electronic devices today. Efficient embedded systems, especially low power wireless systems, have become an important challenge for engineering design processes. The designs of these systems are under increasing pressure to extend battery time and at the same time offer more features and performance. The upcoming generation of embedded systems is going to need lower active and sleep power consumption while simultaneously increasing the ease of power management development needed to meet time-to-market requirements. In the near future, most sophisticated high performance applications will be deployed on complex platforms based on heterogeneous, multicore and many core architectures, and this perspective will pose new challenges at the power management level. This work already addressed realistic constraints of multicore systems at near-term perspective, based on Dual Cortex A9 in the context of the COMCAS project. Inevitably, the evolutions in platform architectures and application complexity especially given the very fast trends of mobile technology will stress the need for effective and fast development approaches.

This thesis has brought a contribution to the characterization of real constraints encountered in advance power management solutions on actual platforms. The realistic energy measurement approach adopted has led to define helpful guidelines for the effective use of power management. Mainly, existing power management solutions can be categorized on the basis of two broad techniques which are DPS and DVFS, and our state of the art also pointed out that the majority of available operational solutions were general purpose and workload based. Therefore, we have further investigated domain or application specific power management solutions in search of the greatest net energy gains possible. The results obtained indicate consistent energy savings accompanied by certain platform and application conditions, which can greatly affect the efficiency of any power strategy. The most relevant achievements in addressing the effectiveness of power management are summarized below.

- Reducing power consumption using DVFS based policies does not always provide satisfying energy savings. In practice, the efficiency of DVFS is highly dependent upon the characteristics of operating points of the target platform. These characteristics must be analyzed before deciding to use a DVFS based strategy.

- To further help the evaluation of power policy relevance for an application and target platform, early estimations can serve a very useful purpose. This study has also proposed some high level energy models to let the estimation of power management benefits that can be applied at very early stages of a system development.

- Energy savings for different DPS and DVFS power strategies pointed out the effectiveness of using application and domain specific power management solutions, which provide further room for energy improvements in contrast to general purpose power strategies that are sometimes in efficient in case of specific, advanced or demanding applications.

- Different results highlighted the important effects of state transition latencies inherent to the platforms, which could at some point limit (and sometimes prevent) the applicability of a power management strategy.

- Findings in respect of actual energy savings for DPS and DVFS based strategies helped in finding out the effects of platform/application parameters on the achievable gains. By this way, we also categorized the conditions for the effectiveness of a specific strategy and better instruct which solution is suited for a particular platform and technology.

- Another contribution relates to the relative lack of real world experimentation based research in the field of power management, especially concerning multiprocessor systems. This work also demonstrates the feasibility of advanced power management approaches such as those based on real multiprocessor scheduling, thanks to a prototyping method developed at LEAT.

- Finally, this work has brought a successful contribution within the COMCAS project in a close cooperation with some project partners (Thales Communications, TIMA, CEA LETI), which results have been presented for

demonstrations at the project reviews for the CATRENE office held at Nijmegen and Grenoble, with very positive feedback from the project reviewers.

6.3.1 Perspectives

The work was carried out in the context of the COMCAS project, to address the challenge of finding a breakthrough in ultra-low-power design for data communication-centered, heterogeneous, multicore architectures, targeting 45 nm and 32 nm CMOS technologies. In future works, this contribution can serve a basis for further investigations of power management strategies in the scope of the next project proposal which is under submission.

This will focus however on heterogeneous system for upcoming technologies beyond 32 nm. Therefore, an axis of research will be to address for instance many core architectures and low power scheduling techniques based on the use of dynamically reconfigurable accelerators.

Another interesting and necessary area to explore would be to consider application development standards for power management, which could be a subpart or super part of ACPI. Such a standard would add information regarding various factors (platform parameters, application limitations, operating points, switching latencies etc.) that should be accounted before developing new applications. By this way, the developer would be able to consider minimizing the energy consumption at development stages by integrating the power management solutions within an application.

Finally, as emerging solutions such as Energy Harvesting or Wireless Sensor Networks tend to suggest, future systems will certainly have to reach higher orders of magnitude in energy efficiency. A promising solution lies in the investigation of hybrid power management that combines and adapts different techniques to fit dynamically the environment of context of execution with the most suited power strategies.

VITA

Name: Jabran Khan Jadoon

Address: Laboratory of Electronics, Antennas and Telecommunications
University of Nice Sophia Antipolis
Campus Sophi@tech-Batiment forum
BP 145 - 930 Route des Colles
06903 sophia antipolis cedex

Email Address: jkhan@unice.fr

Education: B.Sc Computer Engineering
COMSATs University of Information Technology, PAKISTAN

M.Sc Telecommunications and System Microelectronics
University of Nice Sophia Antipolis, France

BIBLIOGRAPHY

[1] G. E. Moore, “Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114,” *Solid-State Circuits Newsletter, IEEE,* vol. 11, no. 5, pp. 33-35, 2006.

[2] I. Microsoft, *Advanced Power Management (APM) : BIOS Interface Specification*, Revision 1.2 ed., February 1996.

[3] D. M. Brooks, P. Bose, S. E. Schuster *et al.*, “Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors,” *Micro, IEEE,* vol. 20, no. 6, pp. 26-44, 2000.

[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *In proceedings of 27th international symposium on Computer Architecture,* pp. 83-94, 2000.

[5] A. Sinha, and A. P. Chandrakasan, "JouleTrack-a Web based tool for software energy profiling," *In proceedings of Design Automation Conference* pp. 220-225, 2001.

[6] T. Simunic, L. Benini, and G. De Micheli, “Energy-efficient design of battery-powered embedded systems,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 9, no. 1, pp. 15-28, 2001.

[7] W. Ye, N. Vijaykrishnan, M. Kandemir *et al.*, “The design and use of simplepower: a cycle-accurate energy estimation tool,” *in Proceedings of the 37th Annual Design Automation Conference*, Los Angeles, California, USA, 2000, pp. 340-345.

[8] TIMA. "Techniques de l'informatique et de la Microélectronique pour l'Architecture des systems intégrés," *http://tima.imag.fr/tima/fr/timalaboratory/overview.html.*

[9] Intel, *ACPI Component architecture User Guide and Programmer Reference* 5.10 16, Oct 2012.

[10] G. Bruno, and N. Nicolas, "Dynamic voltage scaling under EDF revisited," *Real-Time Syst.,* vol. 37, no. 1, pp. 77-97, 2007.

[11] ARM, *ARM1176JZF Development Chip - Technical Reference Manual*, 2007-2012. *http://www.arm.com/product/processor/classic/arm11/arm1176.php*

[12] ARM, *Cortex™- A9 MPCore® Technical Reference Manual*, 2008-2012. http://www.arm.com/product/processor/classic/arm11/arm11-mpcore.php

[13] AMD, *Revision Guide for AMD Family 10h Processors*, 3.92 ed., March 2012.

[14] Intel, *Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor*, March 2004.

[15] Transmeta, "Transmeta™ Crusoe™ TM5800 Processor for Embedded Applications," http://datasheets.chipdb.org/Transmeta/pdfs/brochures/crusoe_for_embedded_applications.pdf.

[16] S. Saowanee, and R. Rajkumar, "Practical voltage-scaling for fixed-priority RT-systems," *In proceedings of the 9th IEEE symposium on Real time embedded Technology and applications,* pp. 106-114, 2003.

[17] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 8, no. 3, pp. 299-316, 2000.

[18] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE,* vol. 91, no. 2, pp. 305-327, 2003.

[19] AMD, "AMD Cool'n'Quiet™ Technology," http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx.

[20] AMD, *AMD PowerNow! Technology*, November 2000.

[21] M. Broyles, C. Francois, A. Geissler *et al.*, "IBM EnergyScale for POWER7 Processor-Based Systems*," In IBM journal of Research and development,* Vol 55 Issue 3 May 2001.

[22] ARM, *Intelligent Energy Manager (IEM) Hardware Control System in the ARM1176JZF-S Development Chip*, Nov 2006.

[23] ARM, *Intelligent Energy Controller - Technical Overview*, 2003-2005.

[24] Pushkar singh, and V. Chinta, *"*Survey Report on dynamic Power Management*," In survey report of University of illnois,Chicago (ECE Department), Chicago, USA 2008* .

[25] J. Haris, S. Muhammad, H. rg *et al.*, "System-level application-aware dynamic power management in adaptive pipelined MPSoCs for multimedia," *in Proceedings of the International Conference on Computer-Aided Design*, San Jose, California.

[26] C. Kihwan, D. Karthik, C. Wei-Chung *et al.*, "Frame-based dynamic voltage and frequency scaling for a MPEG decoder," *in Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, San Jose, California, 2002.

[27] Inki Hongy, Gang Quy, Miodrag Potkonjaky *et al.*, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," *In proceedings of the 9th IEEE synmposium on REAL Time Systems,* pp. 178-187, 1998.

[28] I. Tohru, and Y. Hiroto, "Voltage scheduling problem for dynamically variable voltage processors," *in Proceedings of the 1998 international symposium on Low Power Electronics and Design*, Monterey, California, United States, 1998.

[29] W. Mark, W. Brent, D. Alan *et al.*, "Scheduling for reduced CPU energy," *in Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, Monterey, California, 1994.

[30] G. Quan, and X. Hu, “Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processor,” *in Proceedings of the conference on Design, automation and test in Europe*, 2002.

[31] N. Navet, and B. Gaujal, "Ordonnancement temps réel et minimisation de la consommation d'énergie," *Systèmes temps réel 2 - Ordonnancement, réseaux et qualité de service*.

[32] F. Gruian, “Energy-Centric Scheduling for Real-Time Systems”, Doctoral dissertation 15, Department of Computer Science - Lund Institute of Technology, 2002.

[33] W. Weixun, and P. Mishra, "PreDVS: Preemptive dynamic voltage scaling for real-time systems using approximation scheme," *in proceedings of 47 ACM/IEEE Design Automation Conference*, pp. 705-710, 2010.

[34] S. Youngsoo, C. Kiyoung, and S. Takayasu, “Power optimization of real-time embedded systems on variable speed processors,” *in Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, San Jose, California, 2000.

[35] H. Aydin, R. Melhem, D. Mosse *et al.*, “Power-aware scheduling for periodic real-time tasks,” *Computers, IEEE Transactions on,* vol. 53, no. 5, pp. 584-600, 2004.

[36] P. Minkyu, H. Sangchul, K. Heeheon *et al.*, “Comparison of Deadline-Based Scheduling Algorithms for Periodic Real-Time Tasks on Multiprocessor*This work is supported in part by Brain Korea 21 project and in part by ICT,” *IEICE - Trans. Inf. Syst.,* vol. E88-D, no. 3, pp. 658-661, 2005.

[37] M.-A. Pedro, L. Eugene, and M. Daniel, “Adaptive scheduling server for power-aware real-time tasks,” *ACM Trans. Embed. Comput. Syst.,* vol. 3, no. 2, pp. 284-306, 2004.

[38] S. Lui, R. Rajkumar, and S. S. Sathaye, “Generalized rate-monotonic scheduling theory: a framework for developing real-time systems,” *Proceedings of the IEEE,* vol. 82, no. 1, pp. 68-82, 1994.

[39] C. L. Liu, and W. L. James, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” *J. ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[40] J. W. S. Liu, "Real-Time Systems," pp. - 2000.

[41] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," *In proceedings of 12th Real Time Systems Symposium*, pp. 129-139, Dec - 1991.

[42] S. Lui, A. Tarek, E. Karl *et al.*, “Real Time Scheduling Theory: A Historical Perspective,” *Real-Time Syst.,* vol. 28, no. 2-3, pp. 101-155, 2004.

[43] M. Ali, and C. Chaitali, “Variable voltage task scheduling algorithms for minimizing energy,” in Proceedings of the 2001 international symposium on Low power electronics and design, Huntington Beach, California, United States, 2001.

[44] S. Dongkun, K. Jihong, and L. Seongsoo, “Low-energy intra-task voltage scheduling using static timing analysis,” in Proceedings of the 38th annual Design Automation Conference, Las Vegas, Nevada, United States, 2001.

[45] M. K. Bhatti, C. Belleudy, and M. Auguin, "An inter-task real time DVFS scheme for multiprocessor embedded systems," *in conference on Design and Architectures for Signal and Image Processing (DASIP)*, pp. 136-143, Oct 2010.

[46] M. L. Dertouzos, and A. K. Mok, “Multiprocessor online scheduling of hard-real-time tasks,” *Software Engineering, IEEE Transactions on,* vol. 15, no. 12, pp. 1497-1506, 1989.

[47] Sudarshan K. Dhall, and C. L. Liu, “On a Real-Time Scheduling Problem,” *in Journal of Operation Research*, pp. 26:127-140, 1978.

[48] B. Sanjoy, and F. Nathan, “Component-Based Design in Multiprocessor Real-Time Systems,” *in Proceedings of the 2009 International Conference on Embedded Software and Systems*, 2009.

[49] Shelby Funk, Vincent Nelis, Joel Goossens *et al.*, “On the Design of an Optimal Multiprocessor Real-Time Scheduling Algorithm under Practical Considerations, ” *in archiveslibrary of Cornell University library ArXiv e-prints*, January 2010.

[50] Tse Lee, Albert Mo, and K. Cheng, “Multiprocessor Scheduling of Hard-Real-Time Periodic Tasks with Task Migration Constraints, ” *International workshop on RTCS and Application*, 2007.

[51] K. Shinpei, and Y. Nobuyuki, “Real-Time Scheduling with Task Splitting on Multiprocessors,” *in Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007.

[52] K. Bhatti, “Energy-aware Scheduling for Multiprocessor Real-time Systems,” *PhD dessertion - LEAT, University of Nice Sophia Antipolis*, June 2011.

[53] Sébastien Bilavarn, KhurramBhatti, and C. Belleudy, "Procédé d'ordonancement avec contraintes d'échéances, en particulier sous Linux, réalisé en espace utilisateur", *Patent pending CNRS - France*, France

[54] Thomas Ritzau, and R. Warnke," QEMU - qemu-kvm & libvirt", 2010.

[55] M. Graphics. "Code Sourcery Tool Chain," http://www.mentor.com/embedded-software/codesourcery.

[56] F. Bellard. "QEMU Open Source Processor Emulator," http://wiki.qemu.org/Main_Page.

[57] S. Bilavarn, C. Belleudy, M. Auguin *et al.*, "Embedded Multicore Implementation of a H.264 Decoder with Power Management Considerations," *in 11th EUROMICRO conference on Digital System Design Architectures, Method and Tools, DSD'08*, pp. 124-130, 2008.

[58] M. K. Bhatti, M. Farooq, *et al.*, “Assertive dynamic power management (AsDPM) strategy for globally scheduled RT multiprocessor systems,” *in Proceedings of the 19th international conference on Integrated Circuit and System Design: power and Timing Modeling, Optimization and Simulation, Delft*, The Netherlands, 2010, pp. 116-126.

[59] M. D. Santambrogio, H. Hoffmann, J. Eastep *et al.*, "Enabling technologies for self-aware adaptive systems," *in conference on Adaptive Hardware and Systems (AHS), NASA / ESA,* pp. 149-156, 2010.

[60] M. A., and G. Fri, "Linux / Unix Command: minicom," http://linux.about.com/od/commands/l/blcmdl1_minicom.htm, Feb 2011].

APPENDIX A – ACRONYM INDEX

ACPI	Advance Configuration Power Interface
AET	Actual Execution Time
AsDPM	Assertive Dynamic Power Management
APM	Advance Power Manager
API	Application Programming Interface
BCET	Best Case Execution Time
BIOS	Basic Input Output System
CMOS	Complementary Metal-Oxide Semiconductor
D	Deadline
DDR	Double Data Rate
DSR	Dynamic Slack Reclamation
DSF	Dynamic Stretch to Fit
DVFS	Dynamic Voltage and Frequency Scaling
DVI	Digital Visual Interface
DPS	Dynamic Power Switching
EDF	Earliest Deadline First
EIST	Enhanced Intel Speedstep Technology
EPM	Enhanced Power Management
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
IEC	Intelligent Energy Controller
IEM	Intelligent Energy Manager
IT	Information Technology
LAN	Local Area Network
LCD	Liquid Crystal Display
MMC	Multi Media Card
OPM	Optimized Power Management
OS	Operating System
P	Period
PCI	Peripheral Component Interconnect
PMC	Power Management Controller
PM	Power Management

POSIX	Portable Operating System Interface
PSRAM	Pseudo Random Access Memory
RM	Rate Monotonic
RAM	Random Access Memory
SDCARD	Secure Digital Card
SDRAM	Synchronous Dynamic Random Access Memory
SMP	Symmetric Multiprocessing
T	Time
UARTS	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus
VGA	Video Graphic Array
WCET	Worst Case Execution time
WFI	Wait For Interrupt

ABSTRACT

The purpose of this study is to investigate how power management strategies can be efficiently exploited in actual platforms. Primarily, the challenges in multicore based embedded systems lies in managing the energy expenditure, determining the scheduling behavior and establishing methods to monitor power and energy, so as to meet the demands of the battery life and load requirements. The work presented in this dissertation is a study of low power-aware strategies in the practical world for single and multiprocessor platforms. The approach used for this study is based on representative multiprocessor platforms (real or virtual) to identify the most influential parameters, at hardware as well as application level, unlike many existing works in which these parameters are often underestimated or sometimes even ignored. The work analyzes and compares in detail various experimentations with different power policies based on Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Switching (DPS) techniques, and investigates the conditions at which these policies are effective in terms of energy savings.

The results of these investigations reveal many interesting and notable conclusions that can serve as prerequisites for the efficient use of power management strategies. This work also shows the potential of advanced domain specific power strategies compared to real world available strategies that are general purpose based in their majority. Finally, some high level consumption models are derived from the different energy measurement results to let the estimation of power management benefits at early stages of a system development.

Résumé

L'objectif de cette thèse est d'étudier l'efficacité énergétique des stratégies basse consommation pour des plateformes représentatives. Principalement, nous nous intéresserons aux stratégies énergétiques pour des systèmes embarqués multicœur en étudiant le comportement de politiques logicielles qui permettent la réduction effective de l'énergie tout en répondant aux exigences applicatives.

Le travail présenté dans ce mémoire vise à étudier des stratégies de gestion de la consommation pour des plateformes monoprocesseur puis multiprocesseur concrètes. L'approche utilisée pour cette étude fut basée sur des plateformes représentatives afin d'identifier les paramètres significatifs, aussi bien au niveau matériel qu'au niveau applicatif, à l'inverse de nombreux travaux dans lesquels ces paramètres sont assez peu pris en compte voir ignorés. Ce travail analyse et compare diverses expérimentations menées sur des politiques énergétiques basées sur des techniques DVFS (Dynamic Voltage and Frequency Scaling) et DPS (Dynamic Power Switching) et définit les conditions sous lesquelles ces stratégies sont efficaces.

Ces expérimentations ont permis d'établir des conclusions remarquables qui peuvent servir de pré-requis lors de la définition de stratégies efficaces de gestion de la consommation. Ces résultats montrent également que pour obtenir des stratégies efficientes il est nécessaire de tenir compte du domaine applicatif. Enfin, il faut noter que les modèles de haut de niveau de consommation ont été définis sur la base des mesures effectuées et afin d'estimer les gains énergétiques dès les premières étapes d'un flot de conception.