Dynamic voltage and frequency scaling framework on digital audio player

Dahyanto, Harliono

2007

Dahyanto, H. (2007). Dynamic voltage and frequency scaling framework on digital audio player. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3523>

<https://doi.org/10.32657/10356/3523>

Nanyang Technological University

Downloaded on 26 Apr 2023 22:50:00 SGT

Dynamic Voltage and Frequency Scaling Framework on Digital Audio Player

Dahyanto Harliono

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2007

Acknowledgments

First of all, I am deeply indebted to my supervisor, Associate Professor Gan Woon Seng, for his enthusiastic guidance and encouragement throughout the course of this work. I am truly thankful to him for the invaluable opportunities and extraordinary experiences.

I would like to express my gratitude to Mr Furi Andi Karnapi, Mr Lee Kong Aik, and Mr Tan Ee Leng. Their exceptional knowledge and insights of DSP field have greatly contributed in many ways to the completion of this thesis. I would also like to thank my good friends, Mr Tandi Wijaya and Mr Ku Wei Chee. I will not forget those good times we spent together. I would also like to acknowledge my friends from DSP lab, Mr Wang Liang, Mr Zhang Hua, and Mr Wen Yuan. They have made the place a great working environment. Special thanks to both technicians, Mr Yeo Sung Kheng and Mr Ong Say Cheng, for their help and interesting discussions.

Last but not least, I am extremely grateful to my parents. Their continuing support and belief has propelled me toward the end of my study. I would like to dedicate this thesis to them.

Table of Contents

Acknowledgments.....	ii
Summary	vi
List of Figures	viii
List of Tables	xi
Chapter 1 Introduction	1
1.1. Scope of Thesis Work	4
1.2. Objectives.....	5
1.3. Major Contributions of Thesis	5
1.4. Organization of Thesis	7
Chapter 2 Fundamentals of Energy Efficient Embedded Processor	9
2.1. Derivation of Power and Energy Consumption of CMOS Circuit.....	10
2.2. Low Power Design Techniques.....	17
2.2.1. Dynamic Power Management (DPM)	17
2.2.2. Dynamic Voltage Frequency Scaling (DVFS)	19
2.2.3. Dynamic Power Management versus Dynamic Voltage Scaling	21
2.3. Power Management Features of Blackfin Processor.....	23
2.3.1. Dynamic Frequency Scaling.....	24
2.3.2. Operating Modes	25
2.3.3. Dynamic Supply Voltage Scaling.....	26
2.4. Conclusions	27
Chapter 3 Implementation of Digital Audio Player on the Embedded Media Processor	28
3.1. Hardware Development Platform.....	28
3.1.1. BF537 Processor Architecture.....	29
3.1.2. ADSP-BF537 EZ-KIT Lite	34
3.2. Digital Audio Decoding and Processing	35
3.2.1. Ogg Vorbis Decoder	35
3.2.2. Graphic Audio Equalizer	40
3.2.3. 3D Audio Effects	42

3.2.4. Reverberation Effects	43
3.3. Conclusions	46
Chapter 4 Dynamic Voltage and Frequency Scaling on Fixed Length Processing	47
4.1. Related Works	48
4.2. Real-Time Deadline	50
4.3. Execution Time and Execution Cycle	54
4.4. Optimum Voltage and Frequency Setting Determination	56
4.5. Application Programming Interfaces (APIs)	60
4.5.1. Deadline Extraction	61
4.5.2. Execution Cycle Measurement	61
4.5.3. Voltage and Frequency Determination	62
4.6. Experiment of proposed DVFS algorithm	64
4.7. Conclusion	66
Chapter 5 Dynamic Voltage and Frequency Scaling on Variable Length Processing	67
5.1. Related Works	68
5.2. Sources of Variable Length Processing	70
5.2.1. Huffman Decoding	70
5.2.2. Variable Bit Rate	74
5.3. Proposed Decoupling Buffer	77
5.3.1. Advantages of Buffer Insertion	78
5.3.2. Voltage and Frequency Setting	80
5.4. Buffer Monitoring	81
5.5. Minimum Buffer Size Determination	83
5.6. Conclusions	85
Chapter 6 Experimental Setup and Power Measurement	86
6.1. Benchmark	87
6.1.1. FLP Benchmark	87
6.1.2. VLP Benchmark	90
6.2. Buffer Level	91
6.3. Physical Current and Voltage Measurement Setup	95
6.3.1. Dedicated current and voltage measurement module on BF537 EZ-KIT Lite	96

6.3.2. Software Setup.....	99
6.4. Power Measurement Results	100
6.4.1. Ogg Vorbis Decoder with Graphic Equalizer.....	102
6.4.2. Ogg Vorbis Decoder with 3D Audio Effect	107
6.4.3. Ogg Vorbis Decoder with Reverberation Audio Effect	110
6.4.4. Ogg Vorbis Decoder with Mixed Audio Effect.....	112
6.5. Conclusions	114
Chapter 7 Conclusions and Future Works	115
7.1. Conclusions	115
7.2. Future Works.....	117
Author's Publications.....	120
Bibliography	121

Summary

Current embedded computing systems demand not only a high performance but also lower power consumption. For this reason, this research aims to build a software framework that enables a rapid design of energy-efficient embedded systems. Particularly, this research focuses on a dynamic voltage and frequency scaling (DVFS) algorithm, which has been found effective in reducing power consumption.

The rationale behind DVFS is to avoid the processor being idle in high operating clock frequency and voltage. Instead, the processor can save power by running the task at a lower clock frequency and voltage, and completing it just before the real-time deadline. The latter action is necessary to ensure a constant performance quality of the application.

This thesis proposes and describes a novel and effective DVFS algorithm that is suitable mostly for embedded signal processing applications. The proposed method does not require a costly estimation process that is being used in many conventional DVFS algorithms. Instead, our algorithm successfully tunes the clock frequency and voltage by exploiting the existence of fixed-length processing and variable-length processing in a typical digital signal processing application.

This thesis also explains the implementation of the proposed DVFS algorithm in a real-world digital audio player application. The experimental results show that the

proposed DVFS algorithm is effective in saving power consumption without compromising the performance quality of the original application.

In addition, the proposed DVFS algorithm can be easily inserted into any of the existing applications. To support this, the DVFS algorithm is translated into a set of library functions that serves as a tool to allow any designer to develop a power-conscious application quickly.

List of Figures

Figure 1.1. Worldwide handheld sales extracted from eTForecasts market research...	2
Figure 1.2. Typical percentage power consumption of portable media player [4].	3
Figure 2.1. Leakage power is dependent on the technology advancement of CMOS circuit [10].	11
Figure 2.2. CMOS logic circuit.	12
Figure 2.3. Circuit delay is affected the by level of supply voltage [14].	15
Figure 2.4. Increasing delay as V_{dd} approaches V_t .	16
Figure 2.5. Block diagram of a typical DVFS-enabled processor.	20
Figure 2.6. Comparison of execution time between DPM and DVFS on the Blackfin Processor.	21
Figure 2.7. Energy consumption analysis using DPM and DVFS techniques.	23
Figure 2.8. PLL block diagram [21].	25
Figure 3.1. Block diagram of ADSP-BF537 [21].	29
Figure 3.2. Blackfin BF537 core architecture and internal memory [21].	30
Figure 3.3. Configurable memory system of Blackfin.	32
Figure 3.4. BF537 memory map [21].	32
Figure 3.5. Various DMA data transfers [21].	33
Figure 3.6. ADSP-BF537 EZ-KIT Lite.	35
Figure 3.7. Block diagram of the Ogg Vorbis decoder.	36
Figure 3.8. An example of audio spectrum reconstruction [27].	38
Figure 3.9. Computational cost of the Ogg Vorbis decoder [29].	39
Figure 3.10. Equalizer implementation using B FIR filters with M taps each.	42
Figure 3.11. 3D audio effect generation.	43
Figure 3.12. Moorer's reverberation algorithm [32].	44
Figure 4.1. An example program and its control flow graph [33].	49
Figure 4.2. Examples of task deadline and execution time.	51
Figure 4.3. Dynamic voltage and frequency scaling of fixed-length processing task.	56
Figure 4.4. Realizable scaling values of Blackfin processor	59
Figure 4.5. C code of API <code>adi_dvfs_GetExecution()</code>	62
Figure 4.6. Flowchart of API <code>adi_dvfs_SetVF()</code>	63

Figure 5.1. Huffman decoding as a part of audio decoder.	71
Figure 5.2. Execution cycle variation of Huffman decoding for every 1-second duration of compressed songs.	73
Figure 5.3. File size comparison of VBR and CBR encoding.	75
Figure 5.4. Data flow of buffer processing method in VLP.	78
Figure 5.5. Voltage and frequency setting per frame.	81
Figure 6.1. Buffer level during Ogg Vorbis decoding with graphic equalizer for (a) the entire 14-second clip and (b) 1-second clip.	92
Figure 6.2. Buffer level during Ogg Vorbis decoding with 3D audio effect.	93
Figure 6.3. Buffer level during Ogg Vorbis decoding with reverberation audio effect.	94
Figure 6.4. Buffer level during Ogg Vorbis decoding with mixed audio effect.	94
Figure 6.5. DAQ connection to BF537 EZ-KIT Lite [54].	95
Figure 6.6. Dedicated circuit to measure core voltage and current on the BF537 EZ-KIT Lite [22].	97
Figure 6.7. Dedicated circuit to measure I/O current on the BF537 EZ-KIT Lite [22].	98
Figure 6.8. LabVIEW graphical program to log data from DAQ.	100
Figure 6.9. DAQ measurement results for Ogg Vorbis decoder with graphic equalizer (inactive DVFS mode).	102
Figure 6.10. Statistical profiling tool to monitor processor's utilization.	103
Figure 6.11. DAQ measurement results for Ogg Vorbis decoder with graphic equalizer (active DVFS mode).	104
Figure 6.12. Detailed graph of DAQ measurement results for Ogg Vorbis decoder with graphic equalizer (active DVFS mode).	105
Figure 6.13. Statistical profiling tool window.	106
Figure 6.14. DAQ measurement results for Ogg Vorbis decoder with 3D audio effect (inactive DVFS mode).	108
Figure 6.15. DAQ measurement results for Ogg Vorbis decoder with 3D audio effect (active DVFS mode).	109
Figure 6.16. DAQ measurement results for Ogg Vorbis decoder with reverberation audio effect (inactive DVFS mode).	110

Figure 6.17. DAQ measurement results for Ogg Vorbis decoder with reverberation audio effect (active DVFS mode).	111
Figure 6.18. DAQ measurement results for Ogg Vorbis decoder with mixed audio effect (inactive DVFS mode).	112
Figure 6.19. DAQ measurement results for Ogg Vorbis decoder with mixed audio effect (active DVFS mode).	113
Figure 7.1. Intermediate thread acts as a replacement of idle thread.	118

List of Tables

Table 2.1. Operational characteristics [21]	26
Table 2.2. Blackfin power domains [21]	27
Table 3.1. FIR delays and coefficients (with 17 non-zero values)	45
Table 3.2. Delay length and feedback gain of comb filters	46
Table 4.1. Deadline cycles for different operating frequencies	54
Table 4.2. Minimum step size for different ranges of frequency scaling values ..	59
Table 4.3. Core voltage and clock requirement for Blackfin ADSP-BF537 [6] ...	60
Table 4.4. Execution cycle for different filter lengths and corresponding frequency and voltage	64
Table 4.5. Power reduction for different filter lengths based on calculation	65
Table 5.1. An example of Huffman look-up table	72
Table 5.2. Variation in execution cycle and output size due to bit rate	76
Table 6.1. Frequency range of the 15-band graphic equalizer	88
Table 6.2. Benchmark results of FLP tasks	89
Table 6.3. Benchmark results of VLP task	90
Table 6.4. Voltage and frequency settings	103
Table 6.5. Comparison of power consumption for Ogg Vorbis decoder with graphic equalizer	107
Table 6.6. Voltage and frequency setting for Ogg Vorbis decoder with 3D audio effect (active DVFS mode)	109
Table 6.7. Comparison of power consumption and processor's utilization for Ogg Vorbis decoder with 3D audio effect	109
Table 6.8. Operating frequency and voltage settings for Ogg Vorbis decoder with reverberation audio effect (active DVFS mode)	111
Table 6.9. Comparison of power consumption and processor's utilization for Ogg Vorbis decoder with reverberation audio effect	112
Table 6.10. Operating frequency and voltage settings for Ogg Vorbis decoder with mixed audio effect (active DVFS mode)	113
Table 6.11. Comparison of power consumption and processor's utilization for Ogg Vorbis decoder with mixed audio effect	114

Chapter 1

Introduction

Handheld devices, such as PDA, laptop computers, portable MP3 players, and digital cameras are becoming popular in recent years. According to the review and forecast report of handheld and mobile devices sales from eTForecasts marketing research and consulting company [1], the worldwide sales of handheld devices, especially PDA with Smartphone capability has increased from 19.95 million units in 2003 to 23.5 million units in 2004, or an increase about 29.3% annually. Sales are expected to grow significantly throughout these few years, as shown in Figure 1.1. Another marketing research company, Research and Markets [2] has also forecast the sale of other handheld devices, such as portable digital audio players. In their report, they indicated that the sales of portable digital audio players will reach 104 million units in 2009, up from 27.8 million units in 2004, representing an annual growth rate of 30.2% between 2004 and 2009.

Following this increasing trend in the usage of portable, mobile, and hand-held electronic devices, manufacturers compete among themselves to produce devices which are the smallest, lightest, and feature rich. However, along with this advancement of the processing capability, such portable devices suffer from huge amount of power intake. Numerous researches [15], [33]-[37], [40], [41] on hardware

and software power management have been conducted to increase the efficiency of energy consumption in portable devices.

Figure 1.1. Worldwide handheld sales extracted from eTForecasts market research.

Since handheld devices are battery-powered, the lifetime of the battery is one of the critical issues. Ironically, FCW Media Group [3] states that although most of handheld components have changed, the batteries have not evolved. Realizing this fact, many active researches are looking for alternative ways of saving power consumption. Much of the focus is given on reducing power consumption of the processors at the core of portable devices. Processors are still the major power consumer in typical portable media players, as shown by Mobile Handset Design Line [4] in Figure 1.2. For power efficiency reason, the latest embedded processor is no longer relying on increasing the clock speed to improve its performance. Other techniques such as single instruction multiple data (SIMD), very long instruction word (VLIW), parallel

instruction sets, and multiple cores are becoming more useful, while maintaining the clock speed. In fact, some of the latest embedded processors [6]-[8] have even reduced the clock speed to reduce the power consumption.

Figure 1.2. Typical percentage of power consumption of a portable media player [4].

An alternative method of power saving is to manage power consumption dynamically based on the performance demand of the running task. Recently, the dynamic power management has become an important area of research and handheld-device makers highlight this feature as the key selling point for their wares [5]. Latest processors are enhanced with features that enable users to have broader control of the application's power profile.

The challenge that designers have to face is to find an effective strategy of minimizing the power consumption without compromising on the real-time performance. This means the designer has to make sure that the system would implement efficient power

management and at the same time, meet the deadline requirement of the application. Furthermore, there is a need to have a power management framework for rapid deployment in any application.

1.1. Scope of Thesis Work

This thesis proposes a dynamic voltage frequency scaling (DVFS) mechanism which is implemented using the latest embedded processor from Analog Devices, the Blackfin processor [6]. The software approach is emphasized so that the power-saving algorithm is portable to other embedded processors that have DVFS power management feature, such as the Texas Instruments C55x [7] and the Intel PXA255 [8]. The DVFS algorithm proposed in this thesis exploits the existence of fixed-length processing (FLP) and variable-length processing (VLP) in a typical digital audio player.

The algorithm is tested in a real-world application (digital audio player) and its effectiveness is investigated in this thesis. The application chosen is a new generation of audio decoder, Ogg Vorbis [9], which is a fully open, non-proprietary, patent-and-royalty-free, general-purpose compressed audio format for mid to high quality audio and music at fixed and variable bit rates. Furthermore, other audio processing effects are included into the audio decoder, such as the graphic audio equalizer, reverberation audio effect, and 3D audio effect.

In this thesis, a set of library functions is created to assist programmers when developing their energy-aware application, without lengthening the product's time to market. To complete the research, physical measurement process is carried out to obtain the actual power consumption figures with and without the proposed DVFS activated.

1.2. Objectives

The objectives of this thesis are:

- (i) To discuss components that contribute to power consumption and to investigate features that allow the latest processors to combat power waste.
- (ii) To study and compare existing power reduction techniques, including dynamic power management (DPM) and DVFS. The latter is a newer technique that still requires research breakthroughs to be optimally deployed in real-world applications.
- (iii) To propose and describe a novel DVFS algorithm that is targeted to real-time multimedia applications.
- (iv) To illustrate the measurement setup and present the measurement results of power consumption of a digital audio player prototype that also runs the proposed DVFS algorithm.

1.3. Major Contributions of Thesis

Major contributions of this thesis are summarized as follows.

- (i) A novel DVFS algorithm is proposed that can guarantee real-time

requirement to be met throughout the application's runtime. The proposed algorithm differs significantly from most of the existing techniques that use prediction methods to find the correct settings of processor's voltage and frequency. The risk of missed deadline has been eliminated in the proposed algorithm, and the ease of implementation is maintained.

- (ii) The proposed DVFS algorithm is translated into a set of library functions that can be inserted quickly into an application. Although the application programming interfaces (APIs) are written for the Blackfin processor, the codes are provided and can be promptly ported to any processor that features DVFS mechanism. The APIs are written to shorten the development time of a power-conscious application significantly.
- (iii) The overhead of implementing the proposed DVFS algorithm is discussed. The memory overhead is incurred mainly due to the buffering mechanism used in the algorithm. The minimum buffer size is derived mathematically and found to be within the available internal memory space of the Blackfin processor.
- (iv) The proposed DVFS algorithm is implemented in real-world applications, such as the Ogg Vorbis decoder, audio graphic equalizer, reverberation audio effect, and 3D audio effect. These applications serve as a guideline as to apply the provided APIs in fine tuning the power consumption of processor.
- (v) The measurement setup consists of hardware and software that is specifically configured to derive the current consumption of the

processor. Analysis of the results and calculation for corresponding power consumptions are also presented in this thesis.

- (vi) The proposed DVFS algorithm is applicable to both audio and video decoding applications since these applications contain variable length processes. The proposed buffering mechanism absorbs the variation and maintains the output rate. Furthermore, the buffer separates the variable length process and fixed length process, allowing each of the processes to be operated independently at different voltage and clock frequency settings.

1.4. Organization of Thesis

The thesis is organized into seven chapters.

- Chapter 1 (current chapter) presents the background and motivation for the work done in this thesis. The scope, objectives, and contributions of this thesis are highlighted as well.
- Chapter 2 discusses how power and energy dissipation are derived based on the supply voltage and operating frequency of the processor. In addition, existing power management techniques are reviewed and the power management features of the Blackfin processor are elaborated.
- Chapter 3 covers both hardware and software components that constitute the implementation of the proposed DVFS algorithm. The Blackfin processor architecture is presented, and is followed by an explanation on the implementation of audio decoding and audio post-processing tasks.

- Chapter 4 describes the proposed DVFS algorithm on fixed-length processing (FLP) tasks. This chapter shows that the deadline and execution time of the task can be derived during run-time. Moreover, the API functions to obtain deadline, execution time, and optimum frequency-voltage settings are included. An experiment is carried out in this chapter to show the effectiveness of the proposed algorithm.
- Chapter 5 explains the proposed DVFS algorithm on variable-length processing (VLP) tasks. Some issues inherited in the VLP task make it difficult to predict its workload. To overcome this, the proposed DVFS algorithm utilizes buffer to absorb the variation of workload of VLP task. API functions to control and monitor the amount of samples in the buffer are detailed, and the derivation of minimum buffer size is presented.
- Chapter 6 shows the measurement setup and analyses of the measurement results, focusing on the impact of the proposed DVFS algorithm in different circumstances. In addition, the power savings for each case are tabulated and the significance of the algorithm is analyzed.
- Chapter 7 concludes this thesis with some recommendations for future research topics.

Chapter 2

Fundamentals of Energy Efficient

Embedded Processor

Several differences between energy consumption and power consumption are emphasized in this chapter. Though closely related and often used interchangeably, these two terms are fundamentally dissimilar. Portable devices normally depend on batteries as their source of energy. Therefore, a low-power design in embedded system is only meaningful if the energy consumption is reduced. Energy is defined as the time integral of power. If power consumption is low for a very short period of time and high for the remaining time, only a little energy, if any, is saved. Similarly, there is not much energy reduction if power consumption is kept low but the processor needs a longer time to complete the task. Thus, to maximize energy efficiency and prolong battery duration, an embedded system must have low overall power consumption over the application runtime.

In the first section of this chapter, several sources of processor's energy dissipation are presented. Analysis of a circuit at the transistor-level is used to derive equations that relate voltage, frequency, and energy. Based on the circuit analysis result, the later

sections of this chapter describe energy minimization techniques focusing primarily on voltage and frequency scaling.

2.1. Derivation of Power and Energy Consumption of CMOS Circuit

Modern processors comprise of complementary metal-oxide-semiconductor (CMOS) circuits. Total power dissipated by CMOS circuit can be grouped into two components, namely static and dynamic power as shown in the following equation,

$$P_{\text{total}} = P_{\text{static}} + P_{\text{dynamic}} \quad . \quad (2.1)$$

Conventionally, dynamic power consumption dominates the CMOS power consumption. However, CMOS technology advancement has lowered the dynamic power consumption to a level comparable to the static power consumption. Therefore, it is relevant to discuss the significance of static power consumption.

The static power consumption is contributed mainly by the leakage current of the logic circuits. The leakage current was not an issue in the earlier design of CMOS circuits, so its effect was neglected and an “off” transistor could be safely treated as an open circuit. As the current trend of CMOS circuits is to compact more transistors into a single chip, the transistor channel length has to be shrunk lower than 130 nm where the leakage current is no longer omissible. Furthermore, for the purpose of low-power designs, the supply voltage of CMOS circuits has been lowered significantly as compared to the earlier designs. As the threshold voltage, which determines the “off” or “on” states of the transistors, has to be lowered too, the sub-threshold leakage

current inevitably becomes a major component of overall power dissipation. Figure 2.1 shows the trend of leakage current which increases along the technology advancement (indicated by reduction of channel length) [10].

Figure 2.1. Leakage power is dependent on the technology advancement of CMOS circuit [10].

Static power consumption is inherent in the CMOS integrated circuit once the fabrication is completed. State-of-the-art techniques to reduce static power consumption include multiple threshold voltages, gate length variation, and oxide tunneling [11]. On the contrary, dynamic power consumption is dependent on the operation of the processor, and consequently, can be tuned to the application being run on the processor. Our research is focused on the reduction of dynamic power consumption for two specific reasons. Firstly, the proposed method can be implemented on any embedded processor. Secondly, dynamic power consumption is still of considerable amount, if not dominant, of the total power consumption of CMOS circuit.

Further expansion [12] of the dynamic power consumption is shown in the following equation.

$$P_{\text{dynamic}} = P_{\text{switching}} + P_{\text{short-circuit}} \quad (2.2)$$

Both switching power consumption, $P_{\text{switching}}$, and short-circuit power consumption, $P_{\text{short-circuit}}$, can be illustrated in the below figure.

Figure 2.2. CMOS logic circuit.

The most basic CMOS logic circuit is shown in Figure 2.2 that also functions as an inverter. The circuit utilizes two types of metal-oxide-semiconductor field-effect transistor (MOSFET), namely p-type (MOSFET A) and n-type (MOSFET B). Ideally, only one of the MOSFET A or B can become active at any given time in circuit operation. However, during the transition, there is always a short period when both MOSFETs are still active, and thus, form a conducting path or a short circuit between V_{dd} and ground. This is the main cause of $P_{\text{short-circuit}}$.

The more prominent component of dynamic power consumption is $P_{\text{switching}}$. $P_{\text{switching}}$ takes place when the circuit changes its logic state [13]. Figure 2.2 also depicts the total output load capacitance, C_L , at the V_{out} node. A high input voltage at V_{in} (logic '1') turns MOSFET A off and MOSFET B on. Effectively, MOSFET A behaves like an open circuit, while MOSFET B provides a conducting path from V_{out} to ground, for C_L to discharge. A transition from high to low voltage level at V_{in} causes MOSFET A to connect the voltage supply, V_{dd} , to V_{out} node, while MOSFET B becomes an open circuit. Clearly, during this transition, some power is drawn from the supply to charge C_L , as given by the following equation.

$$P_{\text{switching}} = V_{dd} \times i_L \quad , \quad (2.3)$$

where i_L is the current flowing into the load and can be calculated as,

$$i_L = C_L \cdot \frac{\partial V_{\text{out}}}{\partial t} \quad . \quad (2.4)$$

During the other transition of input from low to high, C_L discharges through MOSFET B. As this discharging process does not draw current from the voltage supply, V_{dd} , it does not have any effect on the power consumption. In other words, although there are two transitions in every clock cycle, only one of them causes the switching power consumption. So, energy dissipation incurred from switching power consumption can be derived as,

$$\begin{aligned}
 E_{\text{switching}} &= \int_0^T P_{\text{switching}} \cdot dt \\
 &= V_{dd} \int_0^T i_L \cdot dt \quad ,
 \end{aligned}
 \tag{2.5}$$

where T is the period of the clock. Combining (2.4) and (2.5), and assuming that the high voltage level of V_{out} is at the same level as V_{dd} , switching energy dissipation can be further simplified to

$$E_{\text{switching}} = C_L \cdot V_{dd}^2 \quad . \tag{2.6}$$

Equation (2.6) is derived for a single CMOS circuit. For more complicated integrated circuits, such as processors [14], (2.6) is compensated with a few additional parameters. In a processor, the duration of a clock cycle is not very meaningful. So, the calculation of switching energy dissipation is extended to a duration in which a task executes to completion. This duration is represented in multiples of clock cycle and is denoted as N_{task} . Another parameter is the switching activity, α , of the processor while running the task. Therefore, (2.6) can be rewritten as follows.

$$E_{\text{switching}} = N_{\text{task}} \cdot \alpha \cdot C_L \cdot V_{dd}^2 \quad . \tag{2.7}$$

To calculate the average switching power consumption during task execution, total switching energy consumption is averaged over the task execution time.

$$\begin{aligned}
 \overline{P_{\text{switching}}} &= \frac{E_{\text{switching}}}{(T \cdot N_{\text{task}})} \\
 &= \alpha \cdot C_L \cdot V_{dd}^2 \cdot f \quad ,
 \end{aligned}
 \tag{2.8}$$

where $(T \cdot N_{\text{task}})$ is the task execution time and f is the clock frequency ($f = 1/T$).

Equation (2.8) clearly proves that switching power consumption is proportional to the processor's operating frequency, f , and to the square of supply voltage, V_{dd} .

Nevertheless, the operating frequency and supply voltage are not fully independent of each other. In other words, a rule of thumb relating these two parameters has to be met to ensure the correct or predictable operation of the integrated circuit. Inherently, a reduction in supply voltage must be followed by a reduction in the processor's operating frequency as will be described in the following paragraphs.

Reducing supply voltage of the CMOS circuits comes with a penalty of increased delay. This delay worsens as the supply voltage, V_{dd} , approaches the threshold voltage, V_t , of the MOSFET. In other words, the circuit does not respond as quickly as when a high V_{dd} is deployed. Figure 2.3 illustrates this phenomenon.

Figure 2.3. Circuit delay is affected by the level of supply voltage [14].

The relationship between V_{dd} and the circuit delay, T_d is approximated in the following first order equation [15],

$$T_d = \frac{C_L \times V_{dd}}{\frac{\mu C_{ox}}{2} (W/L) (V_{dd} - V_t)^2} \quad , \quad (2.9)$$

where the technology dependent parameters are μ (charge-carrier mobility), C_{ox} (gate oxide capacitance), W (gate width), and L (gate length).

Figure 2.4. Increasing delay as V_{dd} approaches V_t .

The dependency of V_{dd} , V_t , and T_d is reflected in Figure 2.4. There is a trade-off that has to be considered in the design stage of CMOS logic circuits. To lower the

power consumption, it is preferable to have a low V_{dd} . Subsequently, (2.9) implies that a low V_{dd} has to be accompanied with a low V_t , in order to keep T_d unchanged. However, a MOSFET that has a low V_t suffers more from the sub-threshold leakage current. Therefore, numerous design techniques [10], [15], [16] have been proposed to minimize the leakage current, so that the overall power saving gain from reducing V_{dd} is maintained.

The delay calculated in (2.9) should impose the restriction on maximum operating frequency for a given operating voltage. Thus, in order to reduce power consumption, the processor's operating frequency has to be lowered prior to the operating voltage. Accordingly, to increase the computing performance of the processor, the operating voltage has to be increased prior to the operating frequency.

2.2. Low Power Design Techniques

There are two low power design techniques that have been widely used in industry, namely, dynamic power management and dynamic voltage frequency scaling.

2.2.1. Dynamic Power Management (DPM)

This technique involves the division of the processor into different power domains, each of which has independent power modes. Whenever a particular domain is idle, that domain can enter a low power mode. The low power mode can be achieved in several methods.

The first method is clock gating [17], [18]. This method deactivates clock in any idle power domain, thus effectively freezing the propagation of signal in that domain. Consequently, the dynamic power consumption in the domain (and overall system) is greatly reduced.

The second method is by completely shutting down all the components in a power domain. Unlike clock gating method, this method is able to eliminate the leakage current. A maximum power reduction can be achieved with this method; however, it comes with a cost of longer wake-up time.

The third method is multiple power modes. Instead of only two modes (shutdown and active), the power domain can have more power modes. This method increases the flexibility of dynamic power management and its usage can be more finely tuned to the application. Typical power modes are full performance mode, normal mode, sleep mode, and hibernation mode. Examples of these power modes can be found in Section 2.3, which discusses the power management features of the Blackfin processors.

The main limitation of the DPM technique is the time and power overhead incurred when waking up the components and restoring their original state. A strategy that immediately switches off idling resources may even increase the power dissipation [19]. In view of the real-time application, this method may not be always desirable, because the necessary wake-up time slows down the response of the system and may cause real-time deadline misses. Thus, effectively this method is only useful when a

power domain undergoes a long idle time.

2.2.2. Dynamic Voltage Frequency Scaling (DVFS)

In contrast to DPM, the DVFS technique is constantly optimizing power usage even when the processor is actively executing a task. This technique is a direct implementation of (2.8), where the power consumption is reduced by lowering voltage and frequency. With the advancements in the design of a modern embedded processor, the processor is now fast enough to perform multiple tasks within a very short time and there is room to exploit variable voltage and frequency during processing to best optimize the power consumption.

The main advantage of a good DVFS technique is the ability to maintain the quality of performance in the application. This is possible since the task is always guaranteed to complete before the deadline. In other words, the exact task execution time is transparent and irrelevant to the users, as long as the deadline is always met. Based on this real-time requirement, the DVFS algorithms always revolve around the determination of minimum frequency for a given task. Only after the optimum frequency is obtained, the corresponding operating voltage can be derived from the circuit characteristics of the processor according to (2.9).

Figure 2.5. Block diagram of a typical DVFS-enabled processor.

Figure 2.5 shows the block diagram of a typical DVFS-enabled processor. The processor core performs data transfers with the external memory, I/O, and other peripherals. By isolating the processor core, the voltage and clock, in which the processor core operates, can be adjusted without affecting the other subsystems (external memory, I/O, peripherals). Both voltage and clock are governed by the DVFS circuit consisting of voltage regulator and phase lock loop (PLL). To alter the voltage and clock, the processor core writes the desired values into the voltage regulator control and PLL control registers. Accordingly, the voltage regulator produces two levels of voltage, namely internal voltage ($V_{dd,INT}$) and external voltage ($V_{dd,EXT}$). Similarly, the PLL control generates two clocks, namely core clock (CCLK) and system clock (SCLK). These adjustments of voltage and clock take a finite time to stabilize, around tenths of microseconds. Nevertheless, the disruption to the execution of instructions will be negligible during this transition. This “on-the-fly”

adjustment of the DVFS technique results in fewer overhead cycles as compared to the DPM technique.

2.2.3. Dynamic Power Management versus Dynamic Voltage Scaling

When both DVFS and DPM features are available in embedded processor architecture, it is preferable to exploit DVFS first [20]. The following example further illustrates that DVFS results in better energy saving as compared to the DPM. For easier illustration purpose, we will provide an example of DPM and DVFS which is applicable to the Blackfin embedded processor. As shown in the top diagram of Figure 2.6, the Blackfin processor is given a periodic task at a sampling interval of 666.67 μ s. Initially, the Blackfin processor runs at 600 MHz speed and at a supply voltage of 1.2 V. The power dissipated is estimated as 260 mW. With the specified operating condition of the processor, the task can be completed within 400 μ s. This implies that there is a slack time of 266.67 μ s. Assuming the processor is shut down during this slack time, energy consumption can be calculated as $260 \text{ mW} \times 400 \mu\text{s} = 0.104 \text{ mJ}$.

Figure 2.6. Comparison of execution time between DPM and DVFS on the Blackfin Processor.

If the strategy to save energy is changed from DPM to DVFS, the following result is achieved. The processor can run at 360 MHz to complete the task execution at its deadline of 666.67 μ s. Therefore, the slack time has been exploited to give lighter workload to the processor. With this speed, the processor's supply voltage can be further reduced to 0.85 V. Thus, the power consumed is 78.3 mW. Finally, energy consumption is computed as $78.3 \text{ mW} \times 666.67 \mu\text{s} = 0.052 \text{ mJ}$. Comparing this value with that obtained by the DPM technique, it is shown that the DVFS technique is more effective in this case.

The above example is generalized by ranging the number of execution cycles from zero to number of deadline cycles. As shown in Figure 2.7, DVFS performs better as compared to the DPM method for less than 350,000 execution cycles. For execution cycles that are greater than 350,000 in number, there is not much slack time for DVFS to exploit.

Figure 2.7. Energy consumption analysis using DPM and DVFS techniques.

Realizing the fact that DVFS is generally better than DPM in periodic tasks, we focus our research on implementing power reduction by scaling voltage and frequency dynamically for audio streaming applications. The word “dynamic” means that the method should be able to adapt to changes or variations during execution of the applications, in order to achieve maximum efficiency of energy usage.

2.3. Power Management Features of Blackfin Processor

Blackfin processor [6] supports different techniques to develop power-efficient applications. The first technique is to change the processor’s clock frequency. The

Blackfin has two internal clock domains, which can be changed dynamically using the on-chip PLL. The second technique is to change the processor's operating mode. Blackfin supports different operating modes, which have different performance characteristics and power dissipation profiles. The third technique is to divide the processor into different power domains; voltage level of each domain can be changed independently. The Blackfin processor supports these three power domains.

2.3.1. Dynamic Frequency Scaling

The Blackfin processor's clock frequency scaling is achieved by changing the on-chip PLL module. There are two main clock frequency domains in the Blackfin processor, namely, (i) core clock (CCLK) which controls the processor's core frequency and (ii) system clock (SCLK) which clocks the peripheral and external devices. In order to change these parameters, the Blackfin processor should operate in certain operating modes (will be discussed in Section 2.3.2), in which the PLL, CCLK, and SCLK are enabled.

The CCLK and SCLK of the Blackfin processor are derived from the input clock (CLKIN) signal of the processor. An on-chip PLL multiplies the CLKIN signal by a user programmable $1\times$ to $63\times$ multiplication factor to produce an intermediate clock, VCO. PLL divide register (PLL_DIV) gives further flexibility to vary CCLK and SCLK by dividing VCO with specified divider ratios [21]. The PLL block diagram is shown in Figure 2.8.

Figure 2.8. PLL block diagram [21].

For the purpose of debugging and benchmarking, the Blackfin processor includes **CYCLES** and **CYCLES2** registers that count core clock cycles while the program is executing [21]. As this 64-bit counter increases every core clock cycle, we can utilize these cycle registers from time to time to determine the number of cycles a task takes to complete. Without these registers, counting of cycles introduces significant overhead in the application.

2.3.2. Operating Modes

The Blackfin processor works in four operating modes, each with different performance characteristics and power dissipation profiles. Table 2.1 summarizes the operating characteristic of each mode.

Table 2.1. Operational characteristics [21]

Operating Mode	Power Savings	PLL Bypassed		CCLK	SCLK	Allowed DMA Access
		Status				
Full On	None	Enabled	No	Enabled	Enabled	L1
Active	Medium	Enabled ¹	Yes	Enabled	Enabled	L1
Sleep	High	Enabled	No	Disabled	Enabled	–
Deep Sleep	Maximum	Disabled	–	Disabled	Disabled	–

¹ PLL can also be disabled in this mode.

The “Full On” mode is the maximum performance mode. In this mode, the PLL is enabled and not bypassed. In addition, the “Full On” mode is the normal execution state of the processor, with the processor and all enabled peripherals running at full speed. Direct memory access (DMA) is also available to internal L1 memories in this mode. In “Active” mode, the PLL is enabled but bypassed. Because the PLL is bypassed, the processor’s CCLK and SCLK run at the input clock (CLKIN) frequency. DMA access is available to appropriately configure internal memories. The “Sleep” mode significantly reduces power dissipation by idling the core processor. The CCLK is turned off while the SCLK is still enabled in this mode. Lastly, the “Deep Sleep” mode maximizes power savings by disabling the PLL, CCLK, and SCLK. In this “Deep Sleep” mode, the processor core and all peripherals, except the Real-Time Clock (RTC) are disabled. In addition, DMA is not supported in the last two modes.

2.3.3. Dynamic Supply Voltage Scaling

Besides different operating modes, the Blackfin processor uses three power domains. These power domains are shown in Table 2.2. Each power domain has a separate V_{DD}

supply that allows the domain's supply voltage to be changed independently from one domain to another. Note that the internal logic of the processor and many of the processors input-output (I/O) can be run over a range of voltages.

Table 2.2. Blackfin power domains [21]

Power Domain	V _{DD} Range
All internal logic except RTC	Variable
Real-Time Clock I/O and internal logic	Variable
All other I/O	Variable

The on-chip voltage regulator of the Blackfin BF53x processor generates internal voltage levels (0.8 V to 1.2 V) from an external 2.25 V to 3.6 V supply. Using the voltage regulator control register (VR_CTL), we can vary the internal logic voltage levels in 50 mV increments.

2.4. Conclusions

The equations that relate power, energy, voltage, and frequency have been explained in this chapter. This chapter also presents two energy reduction techniques that have been widely used in the industry. There are advantages and disadvantages of each technique. As DVFS is a newer technology, our research is focused on DVFS as it allows the power to be saved even in the processor's active mode.

Chapter 3

Implementation of Digital Audio Player on the Embedded Media Processor

A typical digital audio player decodes compressed bitstream, performs post-processing and outputs pulse-code modulation (PCM) samples to loudspeakers or headsets. Modern digital audio players are realized on embedded processors, such as the Analog Devices Blackfin processor. Section 3.1 of this chapter discusses the architecture of the Blackfin processor. Subsequently, the introduction of the processing tasks that are used in our power management experiments is provided in Section 3.2. Section 3.3 summarizes the main points in this chapter.

3.1. Hardware Development Platform

In this research, the BF537 EZ-KIT Lite is selected as the development platform for the digital audio player implementation. The BF537 EZ-KIT Lite is a standalone printed circuit board (PCB) that contains the BF537 Blackfin processor and other peripherals such as audio CODEC, flash memory, SDRAM, switches, LEDs, and so on. The relevant details of this hardware development platform are presented in the following sub-sections.

3.1.1. BF537 Processor Architecture

The ADSP-BF537 [6] is a member of the Blackfin processor family from Analog Devices, Inc. (ADI). The Blackfin processor was designed based on the Micro Signal Architecture (MSA). MSA, jointly developed by ADI and Intel, integrates the digital signal processor architecture with microcontroller functionality that significantly improves the performance, power consumption, and programmability of the processor. This combination of processing enables the Blackfin processor to perform equally well in both signal processing and control processing applications, and eliminates the requirement for separate heterogeneous processors as in many conventional embedded media systems. Consequently, this unique feature greatly simplifies both the hardware and software design implementation tasks. In addition, the Blackfin processor features a rich set of peripherals. Figure 3.1 shows the block diagram of the BF537 processor core and peripherals.

Figure 3.1. Block diagram of ADSP-BF537 [21].

Several major components of the BF537 processor are described below.

(i) Core Architecture

Figure 3.2 depicts the BF537 processor core, which consists of three main units: data arithmetic unit, address arithmetic unit, and control unit. The data arithmetic unit is used to perform computation. This unit consists of two 16-bit multipliers, two 40-bit accumulators, two 40-bit arithmetic logic units (ALUs), four 8-bit video ALUs, and a 40-bit shifter [21]. The computational units process 8-, 16-, or 32-bit data from the register file, which comprises of eight 32-bit general-purpose registers (R0-R7).

Figure 3.2. Blackfin BF537 core architecture and internal memory [21].

The second unit within the core architecture is the address arithmetic unit. The important addressing mechanism that is used in our implementation is the circular buffer addressing, which is operated by modifying Index, Modify, Base, and Length

registers. The Index registers (I0-I3) contain the target memory location which would be used as circular addressing. The Modify registers (M0-M3) contain the post-modify counter that updates the corresponding Index register. The Base registers (B0-B3) contain the base address of circular addressing, and the Length registers (L0-L3) contain the length of the circular buffer addressing. In order to use the circular buffer mechanism, one set of Index, Base, and Length registers has to be initialized. For example, in order to use the circular buffer, one may initialize I0, B0, and L0 to perform one circular buffer addressing. Since there are four sets of these registers, the maximum number of four circular buffer addressing can be used at the same time in the program.

The last unit within the core architecture is the control unit or program sequencer. A program sequencer controls the instruction execution flow, including instruction alignment and decoding. Furthermore, a loop buffer is specially provided to cater for zero-overhead looping.

(ii) Memory Architecture

Blackfin processor supports a modified Harvard architecture in combination with a hierarchical memory structure [21]. Figure 3.3 illustrates a two-level memory configuration of Level 1 (L1) and Level 2 (L2) memories. The L1 memory typically operates at the full processor speed with little or no latency. Certain L1 memory blocks may be configured as either SRAM or cache. L1 memory blocks consist of:

- L1 instruction memory that holds instructions only.
- L1 data memory that holds data.
- L1 scratchpad data memory that stores stack and local variable information.

Figure 3.3. Configurable memory system of Blackfin.

On the other hand, the L2 memory serves as a memory expansion that can be accessed through External Bus Interface Unit (EBIU). The L2 memory runs at a slower speed compared to the L1 memory. The memory map of the BF537 processor is shown in Figure 3.4.

ADSP-BF537 MEMORY MAP		
0xFFFF FFFF	CORE MMR REGISTERS (2M BYTE)	INTERNAL MEMORY
0xFFE0 0000	SYSTEM MMR REGISTERS (2M BYTE)	
0xFFC0 0000	RESERVED	
0xFFB0 1000	SCRATCHPAD SRAM (4K BYTE)	
0xFFB0 0000	RESERVED	
0xFFA1 4000	INSTRUCTION SRAM/CACHE (16K BYTE)	
0xFFA1 0000	RESERVED	
0xFFA0 C000	INSTRUCTION BANK B SRAM (16K BYTE)	
0xFFA0 8000	INSTRUCTION BANK A SRAM (32K BYTE)	
0xFFA 00000	RESERVED	
0xFF90 8000	DATA BANK B SRAM/CACHE (16K BYTE)	
0xFF90 4000	DATA BANK B SRAM (16K BYTE)	
0xFF90 0000	RESERVED	
0xFF80 8000	DATA BANK A SRAM/CACHE (16K BYTE)	
0xFF80 4000	DATA BANK A SRAM (16K BYTE)	
0xFF80 0000	RESERVED	
0xEF00 0800	BOOT ROM (2K BYTE)	EXTERNAL MEMORY
0xEF00 0000	RESERVED	
0x2040 0000	ASYNC MEMORY BANK 3 (1M BYTE)	
0x2030 0000	ASYNC MEMORY BANK 2 (1M BYTE)	
0x2020 0000	ASYNC MEMORY BANK 1 (1M BYTE)	
0x2010 0000	ASYNC MEMORY BANK 0 (1M BYTE)	
0x2000 0000	SDRAM MEMORY (16M BYTE - 512M BYTE)	
0x0000 0000		

Figure 3.4. BF537 memory map [21].

The Blackfin processor views memory as a single, unified 4 Gbyte address space,

which is translated to 32-bit addresses. It does not define a separate I/O space. Generally, like other microcontrollers or digital signal processors, the Blackfin processor has a set of control registers to configure and control the overall system and peripherals. These registers are mapped to the same address space and are termed as memory mapped registers (MMR).

(iii) Direct Memory Access

The Blackfin processor has multiple and independent DMA controllers [21], which support data transfers without core intervention. DMA transfers can occur between the internal memories and any of its DMA-capable peripherals. DMA transfers can also be accomplished between any of the DMA-capable peripherals and external devices connected to the external memory interfaces, including the SDRAM controller and the asynchronous memory controller. The peripherals that are DMA-capable include the serial ports (SPORTs), serial peripheral interface (SPI), universal asynchronous receiver transmitter (UART), and parallel peripheral interface (PPI). Each individual DMA-capable peripheral has at least one dedicated DMA channel. These data transfers are illustrated in Figure 3.5.

Figure 3.5. Various DMA data transfers [21].

The DMA controller supports both one-dimensional (1D) and two-dimensional (2D) data transfers. The 2D DMA capability supports arbitrary row and column sizes up to 64K elements by 64K elements, and arbitrary row and column step sizes up to +/- 32K elements. In our experiments, the 2D DMA capability is used to implement the ping-pong buffer for audio processing.

3.1.2. ADSP-BF537 EZ-KIT Lite

In this research, the ADSP-BF537 EZ-KIT Lite [22], as shown in Figure 3.6, is selected as the development board. This standalone printed circuit board (PCB) consists of the ADSP-BF537 Blackfin processor, 64 Mbytes of SDRAM, 4 Mbytes of flash memory, audio CODECs (AD1871 and AD1854), six LED indicators, four push buttons, and other supporting peripherals.

The data transfer between the audio CODEC, which is interfaced to the SPORT, and the processor's memory is controlled by DMA. Hence, for most audio processing tasks, the DMA controller for SPORT determines the real-time deadline of the tasks.

Figure 3.6. ADSP-BF537 EZ-KIT Lite.

3.2. Digital Audio Decoding and Processing

This section discusses the audio decoding and processing tasks used in this thesis. The audio decoding is focused on the Ogg Vorbis decoder. The selected audio processing tasks which are mainly used to enhance the original audio are graphic equalizer, reverberation audio effect, and 3D audio effect.

3.2.1. Ogg Vorbis Decoder

Ogg Vorbis [9] is a relatively new perceptual audio compression format. Its performance is comparable to other second generation audio coding techniques, such as MPEG4-AAC [23], and significantly better than MP3 [24]. The main strength of the Ogg Vorbis coding technique lies in the fact that it is a completely free, open, and unpatented audio coding standard. Ogg Vorbis has found many applications, such as video games [25], radio, and music sites [26].

The Ogg Vorbis decoding process starts from the initialization of the decoder using three header packets with the following sequence: identification header packet, comment header packet, and setup header packet. This initialization stage gives the necessary information to the decoder about the incoming audio packets. In addition, there is virtually no restriction or limit on the packet size. Packets are formed that can be truncated and still remain decodable. This feature is very useful in bit peeling mechanism, where the bitrate can be directly reduced in the compressed domain. The decoding and synthesis process of audio packets can be summarized in Figure 3.7.

Figure 3.7. Block diagram of the Ogg Vorbis decoder.

Each block is described briefly as follows.

- Floor decoding and floor curve generation are two decoding blocks that reconstruct the spectral floor vector, which is a low-resolution representation of audio spectrum. Floor decoding involves Huffman entropy decoding, which is a variable-length decoding.
- Residue decoding is done to reproduce the detailed structure of audio spectrum. Similar to floor decoding, residue decoding includes entropy decoding.
- Although the number of residue vectors equals the number of channels, a decoded residue vector may not directly represent a channel. Due to channel coupling, a decoded vector may be a combination of magnitude (or angle) of two channels. Thus, channel decoupling is used to recover each individual residue vector, which represents detailed spectral structure of a channel.
- Audio spectrum reconstruction is obtained by element-by-element multiplication of the floor curve and residue vectors. Figure 3.8 shows an example of audio spectrum reconstruction from the floor curve and residue vectors.

Figure 3.8. An example of audio spectrum reconstruction [27].

- The full audio spectrum is transformed back to time-domain signal by inverse modified discrete cosine transform (IMDCT) [28]. The modified discrete cosine transform (MDCT) of N real numbers, $x(n)$, $n = 0, 1, \dots, N-1$, is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) \cos \left[\left(n + \frac{N+2}{4} \right) \left(k + \frac{1}{2} \right) \frac{2\pi}{N} \right], \quad (3.1)$$

for $k = 0, 1, \dots, N/2-1$.

Thus, the IMDCT algorithm is expressed as

$$x(n) = \frac{2}{N} \sum_{k=0}^{N/2-1} X(k) \cos \left[\left(n + \frac{N+2}{4} \right) \left(k + \frac{1}{2} \right) \frac{2\pi}{N} \right], \quad (3.2)$$

for $n = 0, 1, \dots, N-1$.

- To obtain the actual PCM samples, the first half of the current frame is overlapped with the second half of the previous frame, using the appropriate Vorbis window as defined in the following formula.

$$w(n) = \sin\left(\frac{\pi}{2} \sin^2\left(\frac{\pi}{2N}(n+0.5)\right)\right) \quad (3.3)$$

Kosaka et al. [29] compare the computational cost of each block of the Ogg Vorbis decoding, as shown in Figure 3.9.

Figure 3.9. Computational cost of the Ogg Vorbis decoder [29].

It is shown that the IMDCT block has the highest workload. As expected, the workload of residue decoding increases with increasing bit rate. The reason is simply that the finer spectral details are represented at a higher bit rate. Note that floor and residue decoding is actually a variable length process, whose workload varies from one time frame to another. It implies that for some time frames, the workload of floor and residue decoders may be different from the average execution cycle, as shown in Figure 3.9. The discussion of variable length process will be postponed to Chapter 5.

3.2.2. *Graphic Audio Equalizer*

Audio equalizer is often used in audio processing to reshape the frequency envelope of the audio. The main benefit is to compensate the unbalanced frequency response of an audio processing system or circuit. For example, the low frequency components of an audio clip need to be amplified if played through a typical low-cost speaker with inherent poor low frequency response. In addition, an equalizer can be utilized to create the feeling of different music genres, such as pop, rock, classical, and many others. This can be done by presetting the gain in each frequency band of the equalizer, as each music genre emphasizes different frequency bands.

An audio equalizer is basically a set of filters, each of which operates in a different frequency band. For example, a 3-band equalizer has a low-pass filter, a band-pass filter, and a high-pass filter. Similarly, 8-band equalizer comprises of a low-pass filter, six band-pass filters, and a high-pass filter. These filters are usually characterized by three parameters, namely center frequency, bandwidth, and gain. The three main types of audio equalizers are parametric equalizer, notch filter equalizer, and graphic equalizer. Parametric equalizer allows all the three filter parameters to be varied, while notch filter equalizer only allows the adjustment of two filter parameters (bandwidth and gain).

Graphic equalizer only changes the gain among filter parameters. Graphic equalizer employs a set of filters with predetermined bandwidths and center frequencies. Typically, the center frequencies of the filters are placed regularly according to the musical interval (octave). Therefore, common graphic equalizer's configurations are 8-band (an octave), 15-band (two third of an octave), and 31-band (one third of an

octave).

In our experiments, the graphic equalizer is constructed from finite impulse response (FIR) filters. All the filters are designed with constant filter order to simplify the implementation. The entire implementation of graphic equalizer can be summarized in the following equation.

$$y(n) = \sum_{j=0}^{B-1} G_j \cdot \sum_{k=0}^{M-1} h_j(k) x(n-k) \quad , \quad (3.4)$$

where $y(n)$ is the output of equalizer at time n ,

M is the number of filter taps,

B is the number of bands,

G_j is the gain of j -th band,

$h_j(k)$ is the k -th filter coefficient of j -th band, and

$x(n)$ is the input sample at time n .

Figure 3.10 shows the block diagram of the graphic equalizer implementation using B M -tap FIR filters. The equalization process begins by updating filters' impulse responses with their corresponding gain settings. Subsequently, the impulse responses from the various bands are summed together to form a new combined impulse response. This combined impulse response is then convolved with the input samples to form the equalized output.

Figure 3.10. Equalizer implementation using B FIR filters with M taps each.

3.2.3. 3D Audio Effects

With 3D audio effect processing, stereo loudspeakers or headsets are able to reproduce sound that is perceived as coming from any desired location. In other words, the sound is spatialized in 3D environment. This spatialization is possible due to the understanding of how humans determine spatial location of a sound. It is widely accepted that there are three important cues containing spatial information; they are inter-aural time difference (ITD), inter-aural intensity difference (IID), and spectral cues [30]. ITD and IID are related to the difference in time and amplitude of the sound when arriving at left and right ears, respectively. Spectral cues are present due to the sound that interacts with the human head, external ear, and body, according to the direction of source.

The 3D audio effect can be achieved by convolving the original sound with a set of impulse responses. Each impulse response, obtained through experiments, is called head related impulse response (HRIR). A commonly used set of HRIR was measured

by the MIT Media Lab [31], which consists of 710 different locations. A location can be defined by its azimuth and elevation angles. To spatialize a sound, at least two sets of HRIR are required, namely left HRIR and right HRIR. Figure 3.11 illustrates the 3D audio effect generation, in which the original sound is convolved with left and right HRIR, and the convolution results are output accordingly to the left and right side of the loudspeaker or headset.

Figure 3.11. 3D audio effect generation.

The implementation of 3D audio effect processing can be carried out by simply using two FIR filters. The HRIRs (left and right) are pre-stored in the memory during compilation of the program. Thus, during runtime, if a new position is selected, the program just needs to update the filter coefficient pointers to point to the selected left and right HRIRs.

3.2.4. Reverberation Effects

Reverberation happens when a sound wave travels and reflects in a confined space and the effects can still be heard even after the sound source has stopped. Some musical tracks can sound richer if played with reverberation effect. Reverberation

effect may be simulated using a set of infinite impulse response (IIR) filters and FIR filters. Reverberation algorithm proposed by Moorer [32], shown in Figure 3.12, realistically simulate the three components of reverberation, namely direct sound, early reflections, and blended reverberations.

Figure 3.12. Moorer's reverberation algorithm [32].

Direct sound is the sound wave which travels directly from the sound source to the listener without any reflection. Early reflections happen when the sound wave reflects once before reaching the listener, usually within 100 msec after the original sound. Blended reverberation, taking place after 100 msec and models the sound wave that reflects more than once. The early reflections are generated using tapped-delay FIR filters, while blended reverberation is implemented using two types of IIR filters: comb filters and allpass filters.

A sparse tapped-delay FIR filter is used to mimic the early reflections and is expressed in the following equation.

$$y(n) = \sum_{k=0}^{17} h(D_k) x(n - D_k) \quad , \quad (3.5)$$

where D_k is the k -th delay and $h(D_k)$ is the corresponding coefficient, as defined in Table 3.1. Note that 17 coefficients are selected to sufficiently emulate the early reflections.

Table 3.1. FIR delays and coefficients (with 17 non-zero values)

Delay (number of samples)	Coefficient
190	0.841
949	0.504
993	0.49
1183	0.379
1192	0.38
1315	0.346
2021	0.289
2140	0.272
2524	0.192
2590	0.193
2625	0.217
2700	0.181
3119	0.18
3123	0.181
3202	0.176
3268	0.142
3321	0.167
3515	0.134

Comb filters are implemented using the following equation.

$$y(n) = x(n) + G_{\text{comb}} \cdot y(n - D_{\text{comb}}) \quad , \quad (3.6)$$

where G_{comb} is the feedback gain and D_{comb} is the delay line. The values of G_{comb}

and D_{comb} for each comb filter are given in Table 3.2.

Table 3.2. Delay length and feedback gain of comb filters

	Delay	Feedback Gain
Comb 1	1579	0.628
Comb 2	1949	0.624
Comb 3	2113	0.619
Comb 4	2293	0.614
Comb 5	2467	0.61
Comb 6	2647	0.605

Allpass filter is applied as in the following equation.

$$y(n) = G_{\text{allpass}} \cdot y(n - D_{\text{allpass}}) - G_{\text{allpass}} \cdot x(n) + x(n - D_{\text{allpass}}) \quad , \quad (3.7)$$

where G_{allpass} and D_{allpass} are the gain and delay of allpass filter, defined as 0.7 and 307 samples, respectively.

3.3. Conclusions

This chapter discussed the hardware and software implementation of a digital audio player, which will be used in our experiments. The hardware discussion focused on the power management features of the Blackfin processor, which enables designers to optimize their power-efficient application in various ways. The Ogg Vorbis decoding and several audio processing tasks were examined in the second part of this chapter. This overview of these algorithms presented the processing complexity of each processing task.

Chapter 4

Dynamic Voltage and Frequency Scaling on Fixed Length Processing

Digital signal processing algorithms usually handle a large amount of data in continuous data stream. Thus, it is not possible to manipulate these streams of data at once for two reasons. Firstly, it incurs a very long waiting time before the first output sample can be obtained. This latency is not acceptable for any real-time application. Secondly, large amount of memory is required to buffer this data for processing. Most of the systems do not have the luxury of large memory; this is particularly true in an embedded system which typically has a small memory size, in the range of hundreds of kilobytes.

Therefore, data is normally divided into frames. Latency can be greatly reduced by processing a frame of data and outputting it immediately. In addition, smaller memory is required, as only a single frame is buffered. Every frame of data is processed continuously with a set of instructions. If the instructions take identical time to complete in every constant-size frame, the processing is referred as fixed-length processing (FLP).

This chapter discusses our proposed DVFS method on FLP that can be shown to be effective in saving power consumption. Although our method is applicable to any processor, we particularly use the Blackfin processor (described in Chapter 3) to show a real-time implementation of our method. This chapter is divided into seven sections. The first section elaborates on the previous related works proposed by other researchers. The next four sections describe the proposed method in detail. Finally, experimental results of the method and a summary of the chapter are presented in the last two sections.

4.1. Related Works

Three conventional DVFS methods [33]-[35] are described in this section. These methods are significantly different from one another, and represent three important schemes, namely intra-task scheduling, prediction based on derived statistical data, and performance tuning.

Shin, et al. [33] proposed an intra-task voltage scheduling algorithm that predicts the worst case execution time (WCET) of the task in advance. Figure 4.1 illustrates the control flow graph of a program, in which the execution path of $\{b_1 \rightarrow b_{wh} \rightarrow b_3 \rightarrow b_4 \rightarrow b_5 \rightarrow b_{wh} \rightarrow b_3 \rightarrow b_4 \rightarrow b_5 \rightarrow b_{wh} \rightarrow b_3 \rightarrow b_4 \rightarrow b_5 \rightarrow b_{wh} \rightarrow b_{if} \rightarrow b_6 \rightarrow b_7\}$ produces WCET. During runtime, when the program flows out of this longest execution path, there is a need to apply lower operating frequency and voltage. The limitation of this approach is that it can be sometimes tedious for the programmer to analyze their application thoroughly. This is especially true in real-world application where there are many branching operations and asynchronous interrupt events.

Figure 4.1. An example program and its control flow graph [33].

Vaidya, et al. [34] developed a system-level voltage and frequency scaling scheme based on the statistical data of processor utilization. This approach creates sampling window, which collects statistic data of the processor utilization. Based on the prediction model, which has been derived previously, the power scaling can be executed. Thus, in addition to the accuracy of the derived prediction model, this technique is heavily dependent on the processor of interest. Porting code from one processor to another may take much effort that can potentially lengthen the development time significantly.

Chakraborty, et al. [35] proposed a low-power audio decoder but at the cost of playback quality degradation. In this method, the voltage and frequency setting is not adapted to the workload requirement of the real-time task. Instead, the workload of the task is varied based on a given voltage and frequency setting. This method

definitely ensures meeting the real-time requirement, as there should not be any deadline misses. However, the performance degradation is not always desirable, though it is claimed that difference in perceived output quality is small.

4.2. Real-Time Deadline

A system is said to be real-time if its quality is determined not only by the correctness of the output, but also by the completion timing of the output. Two major timing properties of a real-time task are deadline and execution time. Deadline can be defined as the amount of time in which a task has to be completed; while execution time is the amount of time a task takes to complete [36]. Figure 4.2 shows three examples of task deadline and execution time. In these examples, the deadline is d seconds. In the first example shown in Figure 4.2(a), the execution time, e , is shorter than the deadline. The power consumption is not optimized due to idling processor (after task completion and before deadline). In the second example shown in Figure 4.2(b), the execution time, e' , is longer than the deadline. The performance of the task is degraded due to the missed deadlines. In the third example shown in Figure 4.2(c), the execution time, e'' , is the same or slightly less than the deadline. This condition is desirable since power consumption is optimized without degrading the performance of the application. This section discusses real-time deadline in detail, while the discussion for execution time is carried out in the next section.

Figure 4.2. Examples of task deadline and execution time.

Real-time systems can be split into two categories, namely hard real-time systems and soft real-time systems. A failure to meet deadline in a hard real-time system causes the whole system to fail. This failure can be very costly, and is very hard to recover from, if not impossible. An example of a hard real-time system is a braking mechanism of a car. Conversely, a soft real-time system can tolerate infrequent failures to meet the deadline. Despite the quality degradation of the application, the system still functions in general. Thus, a failure in a soft real-time system can easily be remedied without affecting the overall execution of the application. An example of soft real-time system is the data transferring in a computer network. Besides, a soft real-time system with very stringent performance quality can also be classified as a hard real-time system.

Therefore, a video decoder that does not tolerate any frame skipping is certainly a hard real-time system.

The quality of an audio decoder can easily be noticed by the user. From the application's point of view, quality of the application is guaranteed by providing enough output samples when deadline arrives. If the audio decoder is linked with an analog-to-digital converter (ADC) operating at 48 kHz, there should be 48,000 decoded output samples in every second. When the application fails to achieve this requirement, the users will judge the entire application as having a poor performance. A choppy song will not sound good even though each of the samples is logically correct.

In an event-triggered system, a real-time processing task is activated when there is a job request from the peripheral. In most digital signal processors, there is another layer of hardware that links between the processor and peripheral. Direct memory access (DMA) handles data transfer between peripherals and the processor. DMA can run independently without disrupting processor activities, and notifies the processor by triggering an interrupt only after it finishes transferring certain number of data samples. In our experiment, DMA controls the data transfer between an audio CODEC and an internal memory space of the processor. Therefore, the real-time deadline is the period when the task must be completed before the next DMA request arrives [37].

As has been described previously, digital signal processing usually handles data in frames. The frame size can be of any value greater than or equal to one. A processing

with a frame size of one is also referred as sample processing. However, a frame with a size larger than one is preferred for reasons of higher throughput and reduced processing overhead [38]. Frame size determines the deadline of a real-time task. For our audio processing task, deadline can be defined as

$$t_{\text{deadline}} = \frac{S_{\text{frame}}}{f_{\text{codec}}}, \quad (4.1)$$

where S_{frame} is the frame size (in number of samples) and f_{codec} is the sampling rate (in Hz) of an audio CODEC.

In our experiment, the frame size is set to 32 samples. As each sample occupies two bytes (short integer data type), 64 bytes of memory space is required to buffer a frame. The audio CODEC operates at a fixed rate of 48 kHz [22]. Therefore, real-time deadline can be calculated as

$$\begin{aligned} t_{\text{deadline}} &= \frac{32}{48 \times 10^3 \text{ Hz}} \\ &= 666.67 \mu\text{s}. \end{aligned} \quad (4.2)$$

In the actual implementation, deadline can be measured in terms of number of processor cycles; we term this as deadline cycle. Period of a processor cycle is the inverse of the processor clock speed. For example, a 100 MHz processor runs 100 million cycles in a second, or completes an instruction cycle in 10 nanoseconds (ns). As a DVFS-capable processor can vary its clock speed, the deadline cycle would also vary. Conversion from deadline, t_{deadline} , to deadline cycle, N_{deadline} , is given by

$$N_{\text{deadline}} = t_{\text{deadline}} \times f_{\text{core}}, \quad (4.3)$$

where f_{core} is the processor's current operating frequency.

Table 4.1 summarizes different number of deadline cycles for various operating frequencies. Note that $S_{\text{frame}} = 32$ and $f_{\text{codec}} = 48 \text{ kHz}$ are used in this experiment.

Table 4.1. Deadline cycles for different operating frequencies

Core Frequency	Deadline Cycle
31.25 MHz	20,833
62.5 MHz	41,667
125 MHz	83,333
250 MHz	166,670
550 MHz	366,670

4.3. Execution Time and Execution Cycle

Another timing property of a real-time system is the task execution time. The amount of time a real-time task takes to complete is critical, as it has to be less than the real-time deadline. As long as this real-time condition is met, the performance quality of the real-time system is maintained. Since this quality is the only thing noticeable by the end-users, users will not be able to differentiate between short and long execution time of the task. As today's digital signal processors (DSPs) are becoming more powerful, it is very feasible to finish the task execution much earlier than the deadline. Ironically, the performance quality is not increased by completing the task early. In contrast, it consumes more energy if compared to a task that completes just before its deadline. Thus, it is always preferred to complete the task execution as late as possible but just before the deadline. In other words, the task is completed just in time. For this purpose, many modern processors feature DVFS to allow performance and power consumption tuning to suit the current task in an application.

Unlike deadline, the execution time is not easily determined before the task actually

runs. This is especially true for a large application which consists of many parallel tasks or subtasks occurring at different instance of time. Different types of tasks definitely have different execution times. Even for the same algorithm, different ways of implementation would result in different execution times. Methods to estimate or predict the task execution time have been proposed to overcome these issues. However, prediction methods bear a greater risk of missing deadlines. The accuracy of the prediction can be improved, but at a cost of increased complexity. The portability of the application code can also suffer, as this method requires detailed information of the program and the processor used in the compilation stage.

Due to the above facts, a feasible way to compute execution time of a real-time task is by benchmarking it during the actual execution. This is primarily possible in a FLP that implements digital signal processing algorithms. The first run of the frame is used to learn about the execution time of the task. Subsequently, based on the obtained execution time, the processor adapts its performance and power consumption to optimize the following frames. We will discuss this method in more details in the next section.

Similar to deadline, execution time can be represented as number of clock cycles; or execution cycle. In benchmarking a section of code, execution cycle is measured instead of its execution time. A conversion from execution cycle, N_{exe} , to execution time, t_{exe} , can be done as follows

$$t_{\text{exe}} = \frac{N_{\text{exe}}}{f_{\text{core}}}, \quad (4.4)$$

where f_{core} the processor's current operating frequency.

For a given FLP task, the execution cycle is constant even though the processor clock speed varies. The reason is that the same number of cycles is taken to run an instruction regardless of the processor's speed. In conclusion, there is a contrast between execution time with deadline. If we scale the processor's clock speed, we will see a change in deadline cycle and execution time, but deadline and execution cycles stay constant. In other words, deadline and execution cycles are insensitive to the processor's operating frequency.

4.4. Optimum Voltage and Frequency Setting Determination

Once the deadline and execution time are obtained, the optimum voltage and frequency can be determined. For FLP tasks, the voltage-frequency estimation process is illustrated in the following figure.

Figure 4.3. Dynamic voltage and frequency scaling of fixed-length processing task.

Figure 4.3 shows the block diagram of our proposed DVFS method on a FLP task. The layers of shaded boxes indicate FLP tasks that are operated on different frames of the data. As discussed, the execution time of FLP task is constant from one frame to another. Hence, our method intentionally lets the first run of FLP task work in non-optimum (or highest) voltage and frequency. The purpose of doing this is to obtain the execution time of the FLP task, with the assumption that it will not change for the next frame. In addition to execution time, it is also shown in Figure 4.3 that the information on deadline can be extracted from the DMA that handles data transfer. Every time a DMA finishes transferring a frame of data, an interrupt will be triggered. So, deadline is simply the interval between these two interrupts. As deadline is usually constant throughout the application lifetime, it needs to be measured only once.

Based on the two timing information of the real-time task, we can immediately determine the optimum frequency setting for a given task. Ideally, the new frequency, f'_{core} , can be calculated as follows:

$$f'_{\text{core}} = \frac{(1+a)N_{\text{exe}}}{t_{\text{deadline}}}, \quad (4.5)$$

where N_{exe} is the number of task execution cycles and a is an extra number of cycles chosen to cater for overhead, such as interrupt latency and DMA setup overhead. The value of a must be a positive number but much smaller than one. The selection of number of extra cycles is open for experiment, but in general, a can be fixed at 0.1.

Equation (4.5) gives an optimum frequency setting to minimize power consumption. However, the computed value may not be achievable as it does not coincide with the

combination of multiplication and division of the external oscillator frequency. As shown in Chapter 3, there are three controls to adjust the processor frequency, namely the input divider (DF), the multiplier ratio ($MSEL$), and the core divider ratio ($CSEL$). Thus, the scaled processor frequency, f_{core} , can be calculated as

$$\begin{aligned} f_{\text{core}} &= \phi \cdot f_{\text{in}} \\ &= \frac{MSEL}{DF \times CSEL} f_{\text{in}} \end{aligned} \quad (4.6)$$

where ϕ is the frequency scaling value,

$$MSEL = 1, 2, 3, \dots, 64 \text{ ,}$$

$$DF = 1, 2 \text{ , and}$$

$$CSEL = 1, 2, 4, 8 \text{ .}$$

Hence, it becomes clear that not any clock frequency is realizable due to discrete values of these multiplier and divider ratios. Figure 4.4 shows the 192 realizable frequency scaling values, ϕ , from the combination of multiplier and divider ratios. Table 4.2 tabulates the steps between two consecutive scaling values. The size of these steps varies for different ranges of clock frequency scaling values, primarily due to the maximum limit of multiplier ratio, $MSEL$, at 64. For example, let us take a case of frequency scaling value of 4. If we would like to achieve a scaling value at 4.0625, we need to set $\{DF, CSEL, MSEL\}$ as $\{2, 8, 65\}$. However, this frequency scaling value is not realizable since the maximum limit of $MSEL$ is only 64. Thus, the nearest scaling value to 4.0625 is 4.125, that would be achieved by setting $\{DF, CSEL, MSEL\}$ as $\{2, 4, 33\}$ or $\{1, 8, 33\}$. The transitions of step size happen at scaling values of 4, 8, 16, and 32 (indicated by triangle markers in Figure 4.4).

Figure 4.4. Realizable scaling values of Blackfin processor

Table 4.2. Minimum step size for different ranges of frequency scaling values

Scaling Value	Minimum Step Size
0.0625 ~ 4	1/16
4.125 ~ 8	1/8
8.25 ~ 16	1/4
16.5 ~ 32	1/2
32 ~ 64	1

Another way to obtain optimum frequency is by gradually reducing the clock frequency from the maximum value in every frame. If deadline cycle is found to be larger than execution cycle in a frame, the processor clock frequency will be scaled one step lower in the following frame. This process may be repeated in a few frames until deadline cycle is as close as possible to, but still larger than the execution cycle. Although this method takes a longer time, a few frames, to settle to optimum frequency, it removes the issue of unachievable calculated frequency and provides a gradual change of core frequency.

Once the optimum operating frequency is registered for each real-time task, the next

call to the task does not require the optimum frequency search anymore. The registered optimum frequency can be directly applied.

As the performance of a real-time system largely depends on the timing of task completion, we must first of all ensure that task execution cycle is smaller than deadline cycle. Therefore, our method begins with the determination of minimum safe operating frequency that can guarantee task completion before deadline. After implementing the frequency adjustment, the voltage level can be altered accordingly to save power consumption even further. The core voltage adjustment can be applied depending on the current core frequency of the processor. The minimum core voltage for a set of core frequencies can be found in the Blackfin processor's datasheet [6], as shown in Table 4.3.

Table 4.3. Core voltage and clock requirement for Blackfin ADSP-BF537 [6]

Parameter		Min	Max	Unit
f_{CLK}	Core Clock Frequency ($V_{\text{DDINT}} = 1.2 \text{ V}$ minimum)		600	MHz
f_{CLK}	Core Clock Frequency ($V_{\text{DDINT}} = 1.045 \text{ V}$ minimum)		475	MHz
f_{CLK}	Core Clock Frequency ($V_{\text{DDINT}} = 0.95 \text{ V}$ minimum)		425	MHz
f_{CLK}	Core Clock Frequency ($V_{\text{DDINT}} = 0.85 \text{ V}$ minimum)		375	MHz
f_{CLK}	Core Clock Frequency ($V_{\text{DDINT}} = 0.8 \text{ V}$)		250	MHz

4.5. Application Programming Interfaces (APIs)

In order to assist programmers to rapidly develop their power-conscious applications, a software library has been written to provide the DVFS functionality. Users can simply insert the function calls to this library in the code and pass necessary parameters to activate power saving feature in the Blackfin processor. A few

important application programming interfaces (API) have been created and they are listed below.

4.5.1. Deadline Extraction

This API is to extract the deadline cycle which depends on the frame size, sampling rate (of the peripheral), and processor's clock speed. The full prototype of this API is

```
int adi_dvfs_GetDeadline (int framesize, int freqperipheral);
```

The above function uses (4.1) and (4.3) to compute the deadline cycle.

A variant API of `adi_dvfs_GetDeadline()` is `adi_dvfs_GetDeadline_Measure()` which measures the deadline cycle. This API requires programmer to insert `adi_dvfs_GetDeadline_Measure()` at the beginning of a program section, which is immediately activated when there is a task request. Usually, the program section is an interrupt service routine (ISR) that handles peripheral. `adi_dvfs_GetDeadline_Measure()` uses CYCLE registers to measure number of cycles spent between two consecutive routine calls.

4.5.2. Execution Cycle Measurement

Figure 4.5 shows the C code of `adi_dvfs_GetExecution()` API function to obtain the execution cycle of a real-time task.

```

int adi_dvfs_GetExecution(int* exeCycle, void* RealTimeTask)
{
    int cycleBegin, cycleCount, cycleEnd;
    cycleBegin = sysreg_read(reg_CYCLES);
    RealTimeTask( );
    cycleEnd = sysreg_read(reg_CYCLES);
    cycleCount = cycleEnd - cycleBegin;
    return cycleCount;
}

```

Figure 4.5. C code of API `adi_dvfs_GetExecution()`.

The usage of this API is quite straightforward. Programmers just need to replace the line where they normally call the real-time task with this API. Notice that the task function pointer is inserted as one of the parameters for this `adi_dvfs_GetExecution()` API. Before the actual task function call is made inside the library, the values of CYCLE registers are stored to variable `cycleBegin`. Similarly, after the task has been completed, the values of CYCLE registers are stored into variable `cycleEnd`. The difference between these two variables is the execution cycle for the task and is registered in the `cycleCount` variable.

4.5.3. Voltage and Frequency Determination

This API library is used to compare execution cycle of the task and the deadline cycle, and subsequently, determine the optimized voltage and frequency settings. Figure 4.6 shows the flowchart of the `adi_dvfs_SetVF()` API. When there is a request to execute real-time task, the voltage and frequency of the processor are first scaled to the maximum. Subsequently, if the deadline cycle is much larger than the execution cycle, the processor's clock frequency is scaled down in steps. This downward scaling

is continued until the deadline cycle is closest (but smaller) to the execution cycle. The obtained optimum frequency and voltage settings are stored for future execution of the real-time task. Finally, the optimum voltage and frequency are applied to the processor during the execution of the real-time task to effectively save power consumption.

Figure 4.6. Flowchart of API `adi_dvfs_SetVF()`.

4.6. Experiment of proposed DVFS algorithm

To implement the proposed DVFS algorithm, we create a simple FIR filtering task with variable filter lengths [39]. By pressing an external switch in the EZ-KIT board, the filter length can be selected as 32 taps, 128 taps, 512 taps, or 1024 taps. This example is chosen to show that changes in performance demand of task must be accompanied with changes in operating frequency and voltage to reduce power consumption.

In this experiment, the number of audio samples per DMA block is 32 samples and the DMA runs at fixed rate of 48 kHz. From (4.2), deadline for this setting is 666.67us. There are 16 predetermined groups of frequency that can be selected. Deadline cycle of each group can be calculated using (4.3).

In this experiment, a in (4.5) is given as 0.1. These extra cycles are inserted to cater for other minor tasks of processor, such as interrupt services for the switch and LED display. Table 4.4 summarizes the execution cycle for different filter lengths. The table also indicates the minimum operating frequency and voltage for each filter.

Table 4.4. Execution cycle for different filter lengths and corresponding frequency and voltage

Filter Length	Execution Cycle	Frequency	Voltage
32	13,922	62.5 MHz	0.8 V
128	39,083	125 MHz	0.8 V
512	138,476	250 MHz	0.8 V
1024	269,601	550 MHz	1.14 V

Without frequency adjustment, we can only run all four filter tasks at the clock frequency of 550 MHz clock speed. This clock frequency is selected because we need to guarantee that the most demanding task (i.e. 1024-tap FIR filtering) executes correctly. Thus, if all tasks run at 550 MHz, inefficiency occurs since execution cycle of the other three tasks is much smaller than deadline cycle. To quantify how much power savings have been achieved after voltage and frequency adjustments, the following equation can be used. Assuming processor consumes constant power throughout the task lifetime, (2.8) can be further expanded as follows:

$$\begin{aligned}
 P_{\text{reduction}} &= \left(1 - \frac{P_{\text{adjust}}}{P_{\text{initial}}} \right) \times 100\% \\
 &= \left(1 - \left(\frac{V_{\text{adjust}}}{V_{\text{initial}}} \right)^2 \times \frac{f_{\text{adjust}}}{f_{\text{initial}}} \right) \times 100\% .
 \end{aligned} \tag{4.7}$$

Equation (4.7) shows the linear and quadratic effect of voltage and frequency adjustment on the power saving, respectively. Using (4.7), we can estimate the power saving of this voltage-frequency adjustment. Variables with subscript “initial” are parameters taken from the state when there is no frequency and voltage adjustment applied. In this case, the initial state is 1024-tap FIR filter task, and hence, the initial voltage (V_{initial}) and frequency (f_{initial}) are 1.14 V and 550 MHz, respectively. The estimated percentage of power reduction with respect to 1024-tap FIR task is tabulated in Table 4.5.

Table 4.5. Power reduction for different filter lengths based on calculation

Filter Length	Power Reduction
32	94.4%
128	88.8%
512	77.6%

4.7. Conclusion

An efficient method to implement DVFS on fixed-length processing task has been presented in this chapter. This method is adaptive, instead of predictive, to the workload of the current task being run. Therefore, the method is suitable for digital signal processing applications, where there are normally large numbers of data to be processed with a fixed repetitive set of instructions. As the quality of the application is not compromised, any change in voltage and frequency settings should not degrade the real-time performance of the application. In this chapter, an experiment is carried out to test the effectiveness of the proposed DVFS algorithm. Its implementation on a real-world application will be carried out in Chapter 6.

Chapter 5

Dynamic Voltage and Frequency Scaling on Variable Length Processing

A digital audio player typically performs two main tasks, namely audio decoding and audio post-processing. To save storage space, songs are normally stored in compressed format that needs to be decoded to pulse-code modulation (PCM) data. Subsequently, the PCM data is manipulated to create various sound effects, such as 3D audio, reverberation, and graphic equalizer.

DVFS on audio post-processing can be extended from our previous proposal described in Chapter 4. Digital signal processing algorithms usually involve computations of a large amount of data that have been divided into frames. An audio post-processing task spends a fixed amount of time to operate on each frame. Exploiting this fact, we use the first frame to let our routine learn about the execution time of a given post-processing task. As the following frames of data undergo the same processing, an optimized frequency and voltage setting can be determined based on the obtained execution time. The main advantage of this method lies in its ability to respond to change of workload in runtime.

In contrast, an audio decoding task (using variable length coding) does not result in constant execution time for every frame. Consequently, a different approach of voltage and frequency adjustment is required, while its practicality must be maintained. In the remaining of this chapter, we use FLP and variable-length processing (VLP) terms to represent audio post-processing and audio decoding, respectively. Section 5.1 examines the previous DVFS works on VLP. Explanations on why execution time variation in VLP exists are detailed in Section 5.2. The proposed method is described in Section 5.3 and its implementation is presented in Sections 5.4 and 5.5. Finally, the essential points are summarized in the last section.

5.1. Related Works

The main challenge of DVFS implementation on VLP lies in the workload variation from time to time. Both predictive and adaptive methods have been proposed by other researchers [33], [40]-[48] to come up with an effective solution. However, each of these methods has its own problems. For example, an inaccurate predictive method can cause missed deadlines that consequently deteriorates the quality. Similarly, a slow adaptive method is not suitable for applications with frequent fluctuation in execution time. In addition, the computational complexity of a DVFS method should be low, in order to avoid processing overhead of processor. In other words, we look for a DVFS method which is not only effective but also efficient.

An adaptive method responds to the change in workload by scaling voltage and frequency. In [40], DVFS was implemented based on the ratio of idle and busy cycles of the processor when the application is being run. Though this adaptation can save

significant amount of power consumption, there is a clear risk of missing deadlines, especially when the fluctuation of VLP is high. As opposed to that, another proposed method, which can fully guarantee VLP task completion before deadline, always assumes the worst-case execution time (WCET) of the task [33]. As discussed previously in Chapter 4, this over-pessimistic method reduces achievable power saving, as most of the times VLP takes less time than WCET.

Various prediction methods have also been proposed to estimate the future workload. Bavier, et al. [41] derive a relationship between decoding time and the number of bits in an encoded frame. However, the actual frame data size in compressed format is not known before the decoding process, due to utilization of entropy encoding and variable bit allocation. Other prediction methods are based on the workload history of previously executed intervals [42], [43]. Furthermore, an error compensation scheme can be incorporated to the prediction DVFS method [44]. However, these prediction methods suffer with a trade-off between prediction accuracy and computational complexity. Normally, to increase accuracy of the prediction, a given predictive DVFS scheme involves more complex algorithms that certainly burden the processor.

To avoid this predictive scheme, a few similar DVFS methods have also been proposed. Instead of adapting to the workload of the application, the processor is fed with more information on the application's average execution time [45], [46] or its optimum voltage and frequency settings [47]. These schemes require modification to the bit stream of the encoded audio or video data, in order to embed this extra information. Therefore, it is not practical to implement these methods on any random decoders, as they may not be aware of these extra bits. In other words, the output of

these methods is a bit stream that is not compliant to the standard audio or video coding.

There are also DVFS methods that use buffers to effectively shift a hard real-time system into soft real-time system. In [48], an offline algorithm is used to schedule the decoding process that has been divided into stages. The decoding process, as a major task in the application, can be executed in both high and low frequencies depending on the buffer level. Because of the variation of decoding process during runtime, this method can only guarantee either optimum power consumption or the fulfillment of real-time requirement, but not both. If the offline calculation assumes WCET for the decoding process, power can still be saved as most of the time decoding processes takes less time than WCET. If the offline calculation assumes execution time less than WCET, deadline can be missed due to decoding process that is over-optimistically run at lower frequency.

5.2. Sources of Variable Length Processing

In general, VLP exists because of the needs to compress data that can be fit into smaller storage space or transmitted over a limited bandwidth channel. This data-dependent compression requires a VLP to recover the source data. This section is dedicated to illustrate the sources of VLP in an audio coding application.

5.2.1. Huffman Decoding

In most audio coding techniques [9], [23], [24], Huffman coding [49] is applied after the perceptual audio coding to further remove redundancies. Unlike perceptual audio

coding, Huffman coding is a lossless data compression, which allows source data to be fully recovered from the compressed data. Huffman coding assigns different codeword for different input samples based on the statistical data of input. More common input characters are coded with shorter codeword, while less common input characters are coded with longer codeword. Consequently, bit stream of the compressed data varies from one input sample to another.

Figure 5.1 illustrates the decoding process of standard audio coding. Huffman decoding is a variable length processing due to the varying bit stream of compressed data. Each constant-size extraction of compressed data does not always represent the same number of output samples. If several shorter codewords are found in a data extraction, more output samples can be produced. Conversely, if a longer codeword is found, another data extraction may be required to keep the number of output samples constant for next decoding stages after Huffman decoding. In addition, the processing time in Huffman decoding also varies among codewords.

Figure 5.1. Huffman decoding as a part of audio decoder.

A conventional method to implement Huffman decoding is by searching the codeword in a look-up table. The look-up table is usually composed of codeword length, codeword, and output character. Table 5.1 shows an example of look-up table. If we encounter a codeword which is located at the beginning of the table, time spent to

search for the output character is short. But, if the codeword is located near the last entries of the look-up table, the search time can be significantly longer. Many researches have been carried out to speed up Huffman decoding process [50], [51]. Nevertheless, due to its inherent characteristic, Huffman decoding is still a variable-length processing.

Table 5.1. An example of Huffman look-up table

Codeword Length	Codeword	Output Character
1	1	A
3	011	B
3	010	C
4	0011	D
4	0010	E
5	0001 1	F
5	0001 0	G
7	0000 111	H
7	0000 110	I
⋮	⋮	⋮

Figure 5.2 shows the varying execution cycle of Huffman decoding in the Ogg Vorbis decoder that runs on the BF537 Blackfin processor. There are two tracks selected for this example; namely *Boom Boom Boom* (by *Vengaboys*) and *Drum* (by unknown artist). These two tracks are coded into Ogg Vorbis file with nominal bit rate of 112 kbps. The whole Ogg Vorbis decoding application is run to completion, but only the Huffman decoding module is benchmarked. To show execution time variation, the benchmarking is carried out for every 1-second duration of the song. Benchmark results for the first seven seconds of each song are shown in Figure 5.2.

Figure 5.2. Execution cycle variation of Huffman decoding for every 1-second duration of compressed songs.

The execution cycle for Huffman decoding varies over the different 1-second intervals. The difference between the longest and the shortest execution times can be quite significant as shown in Figure 5.2; 165,000 cycles or 9.5% for *Boom Boom Boom*, and 180,000 cycles or 11.4% for *Drum*. If we are to compare two different songs, the variation can become even greater. For example, the longest execution cycle of *Boom Boom Boom* and the shortest execution cycle of *Drum* differ by 360,000 cycles or more than 23% of the shortest execution cycle of *Drum*.

The execution time fluctuation of Huffman decoding is random. Thus, it can also be implied that there is a difficulty in predicting the time needed for next Huffman decoding process. Huffman decoding of a fast-paced song may produce large variation between two consecutive frames. Therefore, a prediction based on the

statistical data of the already-decoded output samples may not be robust enough. A DVFS scheme that uses prediction method has a risk of quality drop if the prediction fails too often. We will revisit this problem again later in this chapter when we discuss the proposed solution.

5.2.2. Variable Bit Rate

There are generally two encoding modes in audio coding schemes, namely constant bit rate (CBR) and variable bit rate (VBR). Bit rate is the amount of output data per time segment of audio input being encoded. CBR mode encodes the entire input data with fixed bit rate, disregarding the complexity of each segment. As each segment uses same number of bits, the more complex segments that essentially require more bits will have lower quality than the simpler segments. On the other hand, VBR allocates variable number of bits to each segment to maintain the quality. Thus, an encoding process of a song in VBR mode usually quantifies the desired quality with a number ranging from 0 (lowest quality) to 10 (highest quality). Another way to encode a song in VBR mode is by specifying nominal bit rate. Although it is still up to the encoder to allocate bits to each segment, the encoder tries to produce average bit rate as close as possible to the nominal bit rate. Ogg Vorbis encoding runs best in VBR mode.

In contrast, CBR is still the default mode for MP3 [24] encoding. CBR is also widely used for streaming multimedia applications, which limits the maximum transfer rate. In this case, CBR is used to occupy the available channel bandwidth, which otherwise will be wasted. Another characteristic of CBR is the predictable file size of the

encoded data. Though perfect CBR is difficult to achieve in audio coding, partly due to the Huffman encoding, the size of output data does not deviate much from expected value, which can be calculated as follows:

$$\text{Size (in bit)} = L \times \text{Bit rate} , \quad (5.1)$$

where L is time length of the audio playback. Let us take an example of a 3-minute uncompressed audio file that is being encoded to MP3 format. If CBR mode is selected with a bit rate of 128 kbps or 16 Kbytes per second, the expected output file size is 2,880 Kbytes. In contrast, the output size of VBR encoding is not easily predictable, as it is very much dependent on the complexity of input data

Figure 5.3. File size comparison of VBR and CBR encoding.

Figure 5.3 shows the comparison of output size from CBR and VBR encoding. Both CBR (bit rate of 256 kbps) and VBR (nominal bit rate 256 kbps) encodings are

applied on Ogg Vorbis files. Three song clips are selected for this experiment, *Canon in D – Pachelbel* (4 minutes 43 seconds), *Hotel California – Eagles* (7 minutes 11 seconds), and *It's My Life – Bon Jovi* (3 minutes 46 seconds). The expected size is calculated using (5.1). Thus, it can be observed that the size of Ogg Vorbis files with CBR encoding is always close to the expected value, while Ogg Vorbis files with VBR encoding are smaller than the expected size. The reason of these smaller Ogg Vorbis files is that VBR encoder can reduce more bits for simpler segments. In other words, VBR encoder increases the storage efficiency of these songs, while still maintaining the quality.

Consequently, VBR affects the execution time of decoder. The variation of bit rate in a compressed song means that the workload of the decoder will vary from time to time. Table 5.2 tabulates the variation in execution cycle, decoded output data size, and output playback duration of Ogg Vorbis decoder run on the Blackfin processor. The track, *Boom Boom Boom – Vengaboys*, is encoded in CBR with bit rate 112 kbps and VBR with nominal bit rate 112 kbps. In this example, the decoder extracts and decodes 100,000 bytes from each compressed file.

Table 5.2. Variation in execution cycle and output size due to bit rate

Mode	Execution Cycle	Decoded Data Size (bytes)	Playback Duration (seconds)
CBR	324,133,069	1,257,216	6.55
VBR	364,473,323	1,465,088	7.63

In conclusion, it is not possible to always obtain exactly the same amount of decoded data required for next processing stages in an audio decoder application. At one time, the amount of data exceeds the required amount; at another time, it is much less than

the required amount. Hence, the use of buffer is inevitable in the typical audio decoding applications. We extend the usage of such buffer to implement DVFS on the audio decoder.

5.3. Proposed Decoupling Buffer

To dynamically adjust voltage and frequency on VLP, we cannot simply rely on execution cycle number from previous frame. As the workload of processor varies from time to time, it is less likely that the next frame execution time is similar to the current frame. Realizing that an error in prediction can cause missed deadline, the proposed buffer insertion method tries to overcome this problem at the expense of small memory overhead. This method creates a buffer that collects the decoded output samples before they are transferred to the FLP module. The basic idea is to decouple FLP task from the VLP task. While FLP still needs to meet its hard real-time deadline, VLP now has a soft deadline as its role is only to fill up the buffer. In other words, VLP has been disconnected from the external real-time application, where real-time requirement constraint is imposed.

Figure 5.4 illustrates how the buffer processing method works. The buffer separates the VLP and FLP modules. A feedback path (dotted line) from the buffer to the VLP processing block serves two purposes. Firstly, if the number of decoded data in the buffer is large, the feedback must inform the VLP module to suspend the decoding process. Thus, only the FLP can be performed. Secondly, if the amount of decoded samples in buffer is low, the feedback must request the VLP processing module to continue the decoding process so that the buffer is filled up to well exceed a certain

threshold and ready for the next frames. In conclusion, the VLP module may not run for all time; it is dependent on the state of the buffer. By contrast, the FLP module always runs as long as the audio playback mode is active.

Figure 5.4. Data flow of buffer processing method in VLP.

5.3.1. Advantages of Buffer Insertion

The advantages of buffer insertion can be summarized as follows.

(i) Modifying hard deadline to soft deadline

The most important advantage of using buffers is that no explicit frame processing time is needed, thus missed deadlines due to prediction errors are no longer relevant. By using some simple control schemes, buffer has changed the hard deadline for processing any frame into a soft deadline. In other words, a failure to meet deadline does not disrupt the performance as there are always ready data in the buffer.

(ii) Reducing overhead of DVFS

DVFS on a processor induces a small time overhead (for the PLL to stabilize), that can be still acceptable if the changes in voltage and/or frequency do not occur too often. The small overhead usually spans to tenth of micro seconds [52]. A DVFS scheme that adjusts voltage and frequency at the end of every frame can be counterproductive due to this overhead. This issue can be resolved by a buffer insertion method. A buffer collects output data from a task that runs in such a high speed that the amount of output data is more than a frame. Then, in the subsequent frames, the clock frequency can be lowered as the processor only needs to consume the available data from buffer at a constant rate. Before the buffer becomes empty, the processor speed will be increased again to cater for the buffer filling. Thus, buffer insertion clearly minimizes the overhead incurred, as DVFS takes place after several frames, instead of every frame.

(iii) Optimizing a finite number of realizable operating frequencies

A DVFS-capable processor has a finite number of operating frequencies that can be realized with a given external oscillator (as discussed in chapter 4). If a desired optimum frequency is not feasible, the processor has to run a given task at a slightly higher frequency. Accordingly, the task is completed before its deadline arrives. If this task is frequently executed, the power wastage of running at higher clock speed can be quite significant. The utilization of buffer can help to solve this matter. These extra cycles can be used for its future task (or other incoming tasks), and the corresponding output data are saved to the buffer. In subsequent frame, as some of the output data have been pre-processed, the workload of task is reduced and processor speed can be

readjusted to a lower level.

(iv) Avoiding lagging response

Buffer insertion can also improve the performance of short and random tasks without interrupting other tasks. This is especially true when a user interface is involved. A good user interface must respond to the user's action within a short, unnoticeable time. Without a buffer, this sporadic job may come at times when the processor is busy with a complex task. As the complex task may last for many frames, the processor can only process the sporadic task in much later time. Thus, the users will experience a lagging response from their user interface. However, with buffer insertion, the complex task can be pre-empted for a short time to allow processor to deal with the sporadic task.

5.3.2. Voltage and Frequency Setting

In Figure 5.5, deadlines arriving at every T ms are determined by DMA request. For block-processing implementation, a constant number of output samples must be available at the arrival of deadline. Let us assume the frame size is S samples. Hence, for every T ms, at least S output samples must be ready to be transferred out by the DMA. VLP always runs at higher speed in order to produce as many samples of output as possible. Figure 5.5 also illustrates that the available time to execute VLP task is t ms, which is obtained by subtracting deadline from the execution time of the FLP task. During this t ms period, the number of decoded output samples can exceed S samples. In fact, it is very likely that the number of decoded output samples is multiples of S , which will only be transferred out by the DMA in the subsequent frames. All the excess decoded output samples are temporarily contained in the buffer.

Therefore, as indicated in Figure 5.5, frames 2 to frame p are executed at a lower voltage and frequency as only FLP tasks are performed. Low buffer level is detected in frame $p+1$ that triggers VLP and repeats the VLP-FLP cycle.

Figure 5.5. Voltage and frequency setting per frame.

5.4. Buffer Monitoring

The amount of data in the buffer needs to be monitored continuously. The buffer control must suspend VLP as long as the buffered data is enough for the next frames. On the other hand, the buffer control must initiate the VLP when the amount of data in the buffer is low. The implementation of buffer control, though not complex, is critical for our DVFS method. A set of library, written to perform buffer monitoring, will be detailed in the remaining of this section.

Buffer control is a task that should be run in the background, as it should be kept transparent to the main application. For this reason, buffer control is implemented in deferred callback functions. A callback is a function that is passed as an argument to another function. Typically, a callback handles asynchronous events or interrupts, which happen outside the normal flow of execution. Callback can be considered as a

more efficient way to service an interrupt than a polling method. Callback can automatically respond to a hardware interrupt it is registered to. In contrast, a polling method requires the processor to periodically check the status of the hardware or peripheral.

There are two types of callbacks, namely, live callbacks and deferred callbacks [53]. Live callbacks also term as interrupt service routines (ISR), responds immediately when then interrupt is triggered. Thus, live callbacks run at hardware interrupt time, while a deferred callback executes at software interrupt time, which is the lowest priority interrupt. A live callback of a high priority hardware interrupt should be completed before the processor can service other lower priority hardware interrupts. In other words, low priority interrupts are always disabled when a live callback of a high priority hardware interrupt is being executed. On the other hand, deferred callback of a hardware interrupt does not disable the lower priority interrupts. In a deferred callback, lower priority interrupts are enabled because the callback service executes the callback at the software interrupt which is the lowest priority. When a hardware interrupt occurs, the callback is not immediately executed; instead it is queued to run later, after exiting from the high priority hardware ISR.

Buffer control is implemented in deferred callback because of the following reasons. Firstly, as buffer collects more output data of VLP than required at a given moment, VLP can be suspended for certain duration to allow processor to execute a lower priority interrupt routine. Secondly, VLP task is usually a complex process that may take a longer time to complete compared to the FLP task. Thus, there must be a mechanism to break its execution and switch the processor's focus to FLP, which has

stricter deadline.

The following steps sum up the implementation of our buffer control scheme:

1. A DMA interrupt directs the program flow to an ISR.
2. A deferred callback function is installed to monitor the buffer level.
3. FLP tasks run at low voltage and frequency.
4. ISR returns; program flow exits from the ISR and goes to callback function installed in step 2.
5. In callback function, the buffer level is being monitored. If the level is high, processor does nothing and waits until the next DMA interrupt arrives. If the level is low, step 7 will be executed.
6. When new DMA interrupt arrives, FLP tasks run at low voltage and frequency.
7. Steps 4-6 repeats for several frames until the check in step 5 indicates that buffer level is lower than threshold. The voltage and frequency are then increased while waiting for the next DMA interrupt.
8. When the next DMA interrupt arrives, FLP tasks run at high frequency and voltage. Thus, its execution time is much shorter than in step 6.
9. The remaining time is used to execute VLP tasks until the next interrupt arrives.
10. Return to step 1.

5.5. Minimum Buffer Size Determination

The approximate size of the buffer can be calculated as follows. At any one time, buffer must have at least one decoded frame ready. If frame size, S , is selected for c

number of audio channels, and each data sample is represented by l bytes, the minimum buffer size (in bytes), B , is

$$B = c * S * l \quad . \quad (5.2)$$

In our implementation, as each sample is represented in two bytes, the memory required to store a frame (512 samples) is 2,048 bytes for stereo output. If the core speed during decoding is 550 MHz and the CODEC sampling rate is 48 kHz, the deadline cycle, $N_{deadline}$, for a frame can be calculated using (4.1) and (4.3) as $5.87 * 10^6$ cycles. In our experiment, average million instructions per second (MIPS) for decoding a 256 kbps Ogg Vorbis song is 61 (the complete benchmark results will be presented in Chapter 6). Thus, execution cycle of this decoding process, N_{VLP} , for each frame is

$$\begin{aligned} N_{VLP} &= \frac{\text{MIPS} * 10^6}{f_{\text{codec}}} * S \\ &= \frac{61 * 10^6}{48000} * 512 \\ &= 6.5 * 10^5 \text{ cycles} . \end{aligned} \quad (5.3)$$

The post processing is a graphic equalizer with execution cycles, $N_{FLP} = 6.5 * 10^5$ cycles. Hence, if the first frame is fully utilized for decoding (as shown in Figure 5.5), the average number of decoded frames, n_{frame} , is

$$\begin{aligned} n_{\text{frame}} &= (N_{\text{deadline}} - N_{\text{FLP}}) / N_{\text{VLP}} \\ &= (5.87 * 10^6 - 2.8 * 10^5) / (6.5 * 10^5) \\ &= 8.6 . \end{aligned} \quad (5.4)$$

Rounding up this number to 9 and adding 1 ready frame in the buffer, we can determine that the minimum capacity of buffer is 10 frames, or 20,480 bytes, which is well within the available memory of the processor.

5.6. Conclusions

A buffer insertion method has been presented in this chapter. This buffer decouples the two types of processing that always exist in an audio decoder application, namely VLP and FLP. Unlike FLP, the workload of VLP tasks varies from time to time that makes it hard to predict. Thus, our strategy is to let the VLP task run at higher voltage and frequency and store the output data in the buffer. In the next few frames, when only FLP tasks execute, the DVFS method discussed in Chapter 4 can be applied. The major strength of our DVFS method is its simplicity and practicality that do not cause significant time and workload overheads to the processor. Furthermore, programmers can easily and quickly embed the provided DVFS library on their application as the detail algorithms used in the DVFS method are transparent to them.

Chapter 6

Experimental Setup and Power

Measurement

In this chapter, the previously proposed DVFS methods are applied to an audio decoder application using the Blackfin BF537 processor. The audio decoder application is selected because it consists of both fixed-length processing (FLP) and variable-length processing (VLP). The BF537 EZ-KIT Lite is selected from a range of available Blackfin development boards, mainly due to its power measurement capability that is connected to the National Instruments data acquisition card [54]. A detailed explanation of the power measurement is presented in this chapter.

There are four FLP tasks chosen for our experiments, namely, graphic equalizer, reverberation audio effect, 3D audio effect, and mixed audio effect (combination of equalizer, reverberation, and 3D). The VLP task is the Ogg Vorbis audio decoder. Each of the mentioned tasks is benchmarked independently, and the results are tabulated in Section 6.1. A more detail observation on the buffer is carried out in Section 6.2. Section 6.3 details the hardware and software experimental setup. Section

6.4 presents and analyzes the physical measurement results. Finally, the last section concludes this chapter.

6.1. Benchmark

To benchmark processing tasks, the API `adi_dvfs_GetExecution()` (which is described in Chapter 4) is used. The results are represented in million instructions per second (MIPS). MIPS specifies the minimum processor's speed to run the task while guaranteeing that it meets the real-time deadline. In our application, the deadline is determined by the CODEC sampling rate, which is 48 kHz. One way to derive the MIPS rating of a processing task is by benchmarking the tasks when processing 48,000 output samples. In other words, if the frame size is 256, benchmarking should take place in $48000 \div 256 = 187.5$ (rounded up to 188) consecutive frames. Note that the frame size for all the experiments stated in this chapter is set to 256.

6.1.1. FLP Benchmark

As mentioned in Chapter 4, FLP tasks have constant workloads throughout the application runtime. By running the FLP tasks in a short duration, their workloads can be obtained and used for determining optimum voltage and frequency settings.

The graphic equalizer comprises 15 FIR filters, each of which consists of 512 taps and operates on a different frequency band. These frequency bands are shown in Table 6.1. Users can adjust the gain of each band by pressing external switches in the EZ-KIT board. The active band and its current gain are also indicated by four LEDs in the

EZ-KIT board.

Table 6.1. Frequency range of the 15-band graphic equalizer

Band	Center Frequency (Hz)
0	40
1	80
2	120
3	180
4	250
5	400
6	650
7	1,000
8	1,600
9	2,500
10	4,000
11	6,400
12	10,000
13	14,000
14	18,000

Reverberation audio effect can be generated by method described in Chapter 3. Unlike the equalizer task that uses FIR filter routine of the DSP library functions provided by the integrated development environment, VisualDSP++ [56], all the filtering routines in the reverberation task are written in linear C as these filter structures need to be specifically defined. A new library for this reverberation task was created by writing these filter routines in C and building them with the compiler's optimization turned on. The three customized filters are comb filter, allpass filter, and tapped FIR filter. Subsequently, the reverberation task calls these filter routines to create the room listening environment.

The 3D audio effect is created by FIR filters with predefined sets of impulse responses, as presented in Chapter 3. At any time, only a few FIR filters are active to

create the spatial effect. For example, to perceptually place a mono sound source in a particular 3D location, two FIR filters are selected to mimic the sound path for the left and right ears, respectively. For the experiment, our 3D audio effect is configured in dynamic mode in which the sound source is perceived to move around the listener's head. The transitions of active FIR filters take place in every second.

Table 6.2. Benchmark results of FLP tasks

FLP Tasks	MIPS
Reverberation	224.26
Equalizer	147.82
3D Audio	23
Mixed	401.33

The MIPS requirement for each FLP task is tabulated in Table 6.2. The MIPS readings are obtained by observing the CYCLES registers of the Blackfin processor (explained in Chapter 2) in the VisualDSP++ development environment [56]. Note that the MIPS variation is important for our experiment since the difference of processor's voltage and frequency for each task will be more significant. An estimation of MIPS for the audio equalizer task can be calculated as follows. The equalizer consists of 15 FIR filters, each has 512 taps. Thus, there are $15 \times 512 = 7680$ multiplications per output sample. To produce 48,000 samples in a second, there should be around 369 millions of multiplications in a second. By exploiting the strength of the Blackfin architecture (dual MACs, circular buffer, hardware loop, etc.), the MIPS of equalizer can be reduced to around 148 as shown in Table 6.2. On the other hand, the high MIPS of reverberation program can be attributed to the memory access contention when writing or reading data from the filter delay buffers. Better buffer placement in

different memory banks should improve the MIPS reading in the reverberation program. However, program optimization is not our focus in this research.

Table 6.2 shows that the most processing-demanding task is the mixed audio effect, which combines equalizer, reverberation and 3D audio processing. As the processor's clock speed must always be higher than MIPS, the mixed audio effect task requires the processor to operate at a minimum clock rate of around 402 MHz. Similarly, the processor's minimum clock rate for the reverberation, equalizer, and 3D audio effect tasks are 225 MHz, 148 MHz and 23 MHz, respectively. Ideally, these values should be attained to achieve optimum power reduction. However, due to the limitations in the real-world case, the processor's operating clock speed is scaled to the closest possible number. As in our experiment, the minimum clock frequency step is 25 MHz.

6.1.2. VLP Benchmark

The MIPS requirement of an audio decoder is heavily dependent on the input audio file. Although our DVFS method does not require explicit MIPS information of VLP, the average benchmark reading of Ogg Vorbis decoding on a few sample songs are given to illustrate its workload.

Table 6.3. Benchmark results of VLP task

Bitrate (kbps)	MIPS			Average MIPS
	Clip 1	Clip 2	Clip 3	
64	44.4	45.5	38.2	42.7
128	49.9	50.2	39.9	46.7
256	63.3	67.3	48	59.5

The MIPS numbers given in Table 6.3 is relatively small compared to the maximum clock speed of the Blackfin processor. Thus, it is guaranteed that a high speed execution of this VLP task in a frame results in a large number of output samples, which can be stored in the buffer, and subsequently, retrieved for the next few frames.

6.2. Buffer Level

A closer look at the amount of output samples in the buffer, when the equalizer task is active, is shown in Figure 6.1. Each frame consists of 256 decoded data samples. The 14-second audio clip is *Boom Boom Boom – Vengaboys* is coded at a nominal bit rate of 112 kbps. The rest of the experiments in this chapter use the same audio clip. Figure 6.1 suggests that the VLP process neither constantly executes in every frame nor produces the same number of output samples in each of its execution.

(a)

(b)

Figure 6.1. Buffer level during Ogg Vorbis decoding with graphic equalizer for (a) the entire 14-second clip and (b) 1-second clip.

It is shown that the maximum buffer level of the Ogg Vorbis decoder with graphic

equalizer is 3328 samples or 13 frames. On the other hand, the minimum buffer level is 256 samples or one frame. For the BF537 processor, the internal memory capacity for data storage is 64 Kbytes. Therefore, in this processor, the maximum buffer size required is $3328 \times 4 = 13,312$ bytes or around 20% of the internal memory capacity. Note that 4 bytes are required to store 16-bit stereo data samples.

The buffer level for Ogg Vorbis decoder with 3D audio effect, reverberation audio effect, and mixed audio effect is shown in Figure 6.2, Figure 6.3, and Figure 6.4, respectively. The minimum buffer level is constant for all these tasks, while the maximum buffer level varies. The 3D audio effect task requires the largest buffer size, which is $3840 \times 4 = 15,360$ bytes or around 23% of the BF537 internal memory.

Figure 6.2. Buffer level during Ogg Vorbis decoding with 3D audio effect.

Figure 6.3. Buffer level during Ogg Vorbis decoding with reverberation audio effect.

Figure 6.4. Buffer level during Ogg Vorbis decoding with mixed audio effect.

6.3. Physical Current and Voltage Measurement Setup

The Blackfin BF537 EZ-KIT Lite is equipped with an input-output (I/O) connector that follows the Educational Laboratory Virtual Instrumentation Suite (ELVIS) interface specification from National Instruments (NI) [54]. The ELVIS interface allows the board to be connected directly to ELVIS workstations or indirectly to the NI data acquisition hardware (DAQ). Our measurement setup uses DAQ that is interfaced to the PC through PCI bus, as shown in Figure 6.5.

Figure 6.5. DAQ connection to BF537 EZ-KIT Lite [54].

The BF537 EZ-KIT Lite features a dedicated circuit that can be monitored with the DAQ to give measurement reading of the voltage and current of the core and I/O. In addition, using the DAQ, the Blackfin BF537 evaluation board can be stimulated with complex analog signals, and custom interrupts or timed events can be generated to/from the Blackfin processor.

6.3.1. Dedicated current and voltage measurement module on BF537 EZ-KIT Lite

An additional circuit is needed mainly to measure the current consumption of the core and I/O. The basic idea to obtain current reading is by measuring the voltage drop across a small value (smaller than one ohm) resistor that has been placed in series with the power supply. Small resistor is necessary as the voltage drop has to be kept small to ensure the main circuit (under investigation) still functions properly. Therefore, an amplifier circuit is added to magnify the voltage so that the value can still be observed clearly using normal device such as oscilloscope or DAQ.

Figure 6.6 shows the measurement module of core voltage and current. The core voltage reading can be obtained from the ACH2+ pin of the ELVIS interface. As this output voltage signal equals the active voltage of the Blackfin processor, the reading can be directly taken as the voltage level.

Figure 6.6. Dedicated circuit to measure core voltage and current on the BF537 EZ-KIT Lite [22].

The core current can be measured through the ACH0+ pin of the ELVIS interface. This output signal is in the form of a voltage proportional to the Blackfin processor's current. As the evaluation board is installed with a shunt resistor (in red rectangle), R_{INT_SHUNT} , the actual current consumption can be derived as follows. The gain, G_{INT} , of the two amplifier circuits, namely instrumentation amplifier, G_1 , and non-inverting amplifier, G_2 , is

$$\begin{aligned}
 G_{\text{INT}} &= G_1 * G_2 \\
 &= \left(1 + \frac{100 \text{ k}\Omega}{R_{60}}\right) * \left(1 + \frac{R_{58}}{R_{102}}\right) \\
 &= \left(1 + \frac{100 \text{ k}\Omega}{11 \text{ k}\Omega}\right) * \left(1 + \frac{90.9 \text{ k}\Omega}{10 \text{ k}\Omega}\right) \\
 &= 101.8173 \approx 100.
 \end{aligned} \tag{6.1}$$

The core current, I_{core} , is

$$\begin{aligned}
 I_{\text{core}} &= \frac{V_{\text{DDINT_SHUNT}} - V_{\text{DDINT}}}{R_{\text{INT_SHUNT}}} \\
 &= \frac{(V'/G_{\text{INT}})}{R_{\text{INT_SHUNT}}} \\
 &= \frac{(V'/100)}{0.1} \\
 &= V' * 0.1,
 \end{aligned} \tag{6.2}$$

where V' is the voltage reading obtained from ACH0+ of the ELVIS interface.

Figure 6.7. Dedicated circuit to measure I/O current on the BF537 EZ-KIT Lite [22].

The I/O current, I_{EXT} , can be obtained from the ACH1+ pin of the ELVIS interface.

Similar to core current, this output signal is in the form of a voltage proportional to the Blackfin processor's I/O current. The computation for I/O current is shown below.

$$\begin{aligned}
 G_{\text{EXT}} &= G_3 * G_4 \\
 &= \left(1 + \frac{100 \text{ k}\Omega}{R_{104}}\right) * \left(1 + \frac{R_{63}}{R_{64}}\right) \\
 &= \left(1 + \frac{100 \text{ k}\Omega}{25.5 \text{ k}\Omega}\right) * \left(1 + \frac{90.9 \text{ k}\Omega}{10 \text{ k}\Omega}\right) \\
 &= 49.6586 \approx 50.
 \end{aligned} \tag{6.3}$$

$$\begin{aligned}
 I_{\text{EXT}} &= \frac{V_{\text{DDEXT_SHUNT}} - V_{\text{DDEXT}}}{R_{\text{EXT_SHUNT}}} \\
 &= \frac{(V''/G_{\text{EXT}})}{R_{\text{EXT_SHUNT}}} \\
 &= \frac{(V''/50)}{0.1} \\
 &= V'' * 0.2,
 \end{aligned} \tag{6.4}$$

where V'' is the voltage reading obtained from ACH1+ pin of ELVIS interface. The I/O current can be used to estimate the external power consumption of the Blackfin processor. This external power dissipation is entirely related to the peripherals of the processor. In other words, the power is consumed to drive the output pins of the processor.

6.3.2. Software Setup

Data logging was carried out in the LabVIEW 8.0 environment from National Instruments [55]. The graphical program shown in Figure 6.8 reads the analog input pins of the DAQ at a sampling rate of 50 kHz. These readings are then formatted and

recorded into a separate file, which can be stored and retrieved easily for the next processing. The data comprises of core current, core voltage, I/O current, and trigger signal. Trigger signal is used to indicate the duration when the data being read are of interest. This trigger signal is particularly useful in getting fair comparisons of power consumption in different modes of our application. In our experiment, the trigger is an active high signal.

Figure 6.8. LabVIEW graphical program to log data from DAQ.

6.4. Power Measurement Results

The log file is further processed in MATLAB to detect the trigger signal, remove redundant data, convert data, compute power consumption, and plot results. The

computation consists of instantaneous power consumption (P), energy consumption (E), and average power consumption (P_{avg}). Each of these parameters can be formulated using the following equations. Instantaneous power consumption is defined as,

$$P = V \times I, \quad (6.5)$$

where V and I are the voltage and current, respectively, that can be obtained directly from the measurements.

Energy consumption is the integration of instantaneous power consumption over a specified duration of time, D .

$$E = \int_0^D P dt. \quad (6.6)$$

Average power consumption is thus derived as,

$$P_{avg} = \frac{E}{D}. \quad (6.7)$$

This average power consumption number is used to compare the efficiency of power consumption among the modes and methods presented in this section.

Current and voltage measurements were taken on the Blackfin processor while running Ogg Vorbis decoder with one FLP task active at a time. Trigger signal marks the 5-second duration, which is long enough to show the general trend of current/voltage levels. The clock frequency step is limited to a minimum of 25 MHz, which is the frequency of external oscillator on the BF537 EZ-KIT Lite.

6.4.1. Ogg Vorbis Decoder with Graphic Equalizer

6.4.1.1. Inactive DVFS mode

When DVFS is not enabled, the voltage level, current and instantaneous power consumption during Ogg Vorbis decoding with graphic equalizer are shown in Figure 6.9. It can be seen that the voltage level stays around 1.15 Volts, as the frequency is constant at 550 MHz throughout the application. Consequently, the instantaneous power consumed by the processor is relatively high and inefficient. This inefficiency is due to the long idling time incurred in this processing mode. Figure 6.10 shows the screenshot of the statistical profiling tool in the VisualDSP++ which highlights that the processor idles for more than 45% of the application runtime.

Figure 6.9. DAQ measurement results for Ogg Vorbis decoder with graphic equalizer (inactive DVFS mode).

Figure 6.10. Statistical profiling tool to monitor processor's utilization.

6.4.1.2. Active DVFS mode

When DVFS method is being applied, the core voltage and frequency automatically adapt to the performance demand of tasks. This adaptation is summarized in Table 6.4.

Table 6.4. Voltage and frequency settings

Mode	Voltage (V)	Frequency (MHz)
Decoding + Equalizer	1.15	550
Equalizer	0.8	175

When the DVFS is activated, the voltage, current, and instantaneous power levels are shown in Figure 6.11. As the core voltage is dynamically adjusted, the current varies

accordingly. A more detailed graph over a 1-second period is shown in Figure 6.12. It clearly highlights the duration of high and low voltage levels. The voltage stays at a low level (0.8 V) for most of the time. The duration of low voltage level is not constant, due to the variation in number of samples stored in the buffer. On the other hand, the steady-state high voltage lasts for only one frame, before it is reduced back to the low level of 0.8V. The same trends are also reflected in the current consumption level.

Figure 6.11. DAQ measurement results for Ogg Vorbis decoder with graphic equalizer (active DVFS mode).

Figure 6.12. Detailed graph of DAQ measurement results for Ogg Vorbis decoder with graphic equalizer (active DVFS mode).

The power consumption is tuned more efficiently to the tasks. This can be quantified by the processor's idle time, as depicted in Figure 6.13. The processor's idle time has now reduced to only about 11% of the application runtime.

Figure 6.13. Statistical profiling tool window.

To compare the power consumption of an application when DVFS is being turned on and when DVFS is being turned off, average power consumption has to be calculated using (6.7). The power saving results for Ogg Vorbis decoder with graphic equalizer are tabulated in Table 6.5. Note that the power saving, η , can be obtained by the following formula,

$$\eta = \left(1 - \frac{P'_{\text{avg}}}{P_{\text{avg}}} \right) \times 100\%, \quad (6.8)$$

where P'_{avg} and P_{avg} are the average power consumption with and without DVFS, respectively.

Table 6.5. Comparison of power consumption for Ogg Vorbis decoder with graphic equalizer

Decoder + Equalizer	Average Power (mW)	Power Saving
Inactive DVFS	317.65	N.A.
Active DVFS	103.35	67.46%

The external current consumption, I_{EXT} , is constant at 28.32 mA. This can be explained by the fact that the buffer is implemented in the internal memory of the Blackfin processor, so that the external current consumption is relatively unaffected. The external power consumption, P_{EXT} , can be calculated as follows.

$$P_{\text{EXT}} = V_{\text{DD_EXT}} \times I_{\text{EXT}}, \quad (6.9)$$

where $V_{\text{DD_EXT}}$ is set to 3.3 V for the output pins of the Blackfin processor. Thus, using (6.9), the external power consumption is around 93.46 mW.

6.4.2. Ogg Vorbis Decoder with 3D Audio Effect

The voltage, current, and power consumption levels for Ogg Vorbis decoding with 3D audio effect, with and without DVFS, can be shown in Figure 6.14 and Figure 6.15, respectively. Similar to the results in the previous section, the fluctuation of current consumption level follows the voltage level. The instantaneous power consumption is calculated using (6.5).

Table 6.6 presents two frequencies and voltage levels, in which the processor operates while executing the application. When the processor runs only the 3D audio effect task, the clock frequency is set to the lowest allowable value of 50 MHz. This value is

determined by the minimum frequency requirement imposed by the peripheral devices. For example, the default setting of SDRAM interface on the BF537 EZ-KIT Lite is operating at the clock frequency ranging from 54 MHz to 133 MHz.

Figure 6.14. DAQ measurement results for Ogg Vorbis decoder with 3D audio effect (inactive DVFS mode).

Figure 6.15. DAQ measurement results for Ogg Vorbis decoder with 3D audio effect (active DVFS mode).

Table 6.6. Voltage and frequency setting for Ogg Vorbis decoder with 3D audio effect (active DVFS mode)

Task	Voltage (V)	Frequency (MHz)
Decoding + 3D Audio	1.15	550
3D Audio	0.8	50

Table 6.7. Comparison of power consumption and processor's utilization for Ogg Vorbis decoder with 3D audio effect

Decoder + 3D Audio	Average Power (mW)	Power Saving	Idle Time
Inactive DVFS	318.00	N.A.	75.96%
Active DVFS	77.29	75.69%	49.74%

The average power, power saving, and processor's utilization are tabulated in Table 6.7. There is a significant reduction (up to 75%) in power consumption. However, the

processor's utilization is not optimum, considering that 50 MHz, the lowest operating frequency, is more than twice as high as the MIPS of the 3D audio effect task. This is indicated by the idle time, which still hits 49.74% after DVFS is being applied.

6.4.3. Ogg Vorbis Decoder with Reverberation Audio Effect

Figure 6.16 shows the voltage, current, and instantaneous power consumption levels when there is no DVFS applied to the Ogg Vorbis decoding, combined with reverberation audio effect as the FLP task. Figure 6.17 illustrates the same parameters when the DVFS is active.

Figure 6.16. DAQ measurement results for Ogg Vorbis decoder with reverberation audio effect (inactive DVFS mode).

Figure 6.17. DAQ measurement results for Ogg Vorbis decoder with reverberation audio effect (active DVFS mode).

The operating frequency and voltage of the processor during application runtime are shown in Table 6.8. Furthermore, significant power reduction of 62.35% has been achieved by applying our DVFS method, as tabulated in Table 6.9. The utilization of the processor is quite high, as processor idles in only 11.47% of the time.

Table 6.8. Operating frequency and voltage settings for Ogg Vorbis decoder with reverberation audio effect (active DVFS mode)

Task	Voltage (V)	Frequency (MHz)
Decoding + Reverberation	1.15	550
Reverberation	0.85	250

Table 6.9. Comparison of power consumption and processor's utilization for Ogg Vorbis decoder with reverberation audio effect

Decoder + Reverberation	Average Power (mW)	Power Saving	Idle Time
No DVFS	335.30	N.A.	41.57%
DVFS	126.23	62.35%	11.47%

6.4.4. Ogg Vorbis Decoder with Mixed Audio Effect

Figure 6.18 shows the voltage, current, and instantaneous power consumption levels when there is no DVFS applied to the Ogg Vorbis decoding, combined with mixed audio effect as the FLP task. Figure 6.19 illustrates the same parameters when the DVFS is active.

Figure 6.18. DAQ measurement results for Ogg Vorbis decoder with mixed audio effect (inactive DVFS mode).

Figure 6.19. DAQ measurement results for Ogg Vorbis decoder with mixed audio effect (active DVFS mode).

The operating frequency and voltage of the processor during application runtime are shown in Table 6.10. Furthermore, significant power reduction of 42.20% has been achieved by applying our DVFS method, as tabulated in

Table 6.11. The utilization of the processor is quite high, as processor idles in only 10.74% of the time.

Table 6.10. Operating frequency and voltage settings for Ogg Vorbis decoder with mixed audio effect (active DVFS mode)

Mode	Voltage (V)	Frequency (MHz)
Decoding + Mixed	1.15	550
Mixed	0.9	425

Table 6.11. Comparison of power consumption and processor's utilization for Ogg Vorbis decoder with mixed audio effect

Decoder + Mixed	Average Power (mW)	Power Saving	Idle Time
No DVFS	337.28	N.A.	28.94%
DVFS	194.95	42.20%	10.74%

6.5. Conclusions

This chapter has presented the experiments that were carried out to prove the effectiveness of the proposed DVFS method. The experiments also highlighted the practicality of our method by using the APIs that have been described in the previous chapters. In contrast, it was difficult to demonstrate the practicality of other DVFS algorithms, because they either lack of the actual implementation or are specifically designed for other platforms.

The power consumption is greatly reduced with the increase in processor's utilization when the DVFS is activated. Furthermore, the quality of the application is completely unaffected when the proposed DVFS method is used. The whole DVFS process is transparent to the users, as users should not be able to tell whether the DVFS mode is active or inactive. The proposed DVFS algorithm can achieve significant power reductions, ranging from 42% in Ogg Vorbis decoder with mixed audio effect to 75% in Ogg Vorbis decoder with 3D audio effect. In addition, if a case of Ogg Vorbis decoder without any post processing is to be considered, the voltage and frequency settings, as well as the buffer size, will be very similar to the case of Ogg Vorbis decoder with 3D audio effect. The reason is that the Ogg Vorbis decoder with 3D audio effect is already executed at the minimum allowable clock frequency of the processor core in the BF537 EZ-KIT Lite.

Chapter 7

Conclusions and Future Works

7.1. Conclusions

A common feature available in the latest embedded processors is the voltage-frequency scaling. However, not much has been written in application reports or manuals on how to best use this feature to save power/energy. In this research work, we investigate and propose a novel method to constantly track the events of the application code and select a best voltage-frequency mix to achieve low energy consumption and real-time performance.

The existing DVFS techniques mostly depend on the prediction scheme, which inherently carry a risk of missed real-time deadlines. Generally, the accuracy of the prediction is proportional to the complexity of the algorithm. This means that a complex algorithm may achieve highly accurate prediction, but at the expense of high overhead when implemented in the processor. However, the complexity and accuracy trade-off does not always hold, especially in our proposed algorithm that can achieve near optimum voltage and frequency settings, while keeping the processor overhead low during the activation of DVFS mode. The major difference of our algorithm is

that voltage and frequency settings are dynamically adapting to the current application workload, instead of compile-time prediction.

The proposed DVFS algorithm deals with FLP and VLP tasks differently. The FLP task has a constant workload, while the VLP task has a varying workload during its execution. Our approach consists of three steps. Firstly, deadline is determined depending on the size and rate of the input bit stream. For an event-triggered system, real-time deadline is relatively constant and can be determined as the interval between two consecutive interrupts. Secondly, number of execution cycles of task is continuously tracked and compared with deadline. Thirdly, frequency and voltage are adjusted based on the difference of deadline and execution time. The experimental results of our implementation showed that execution cycle closely lags behind the deadline cycle to shorten the idling time and reduce power wastage during idling stage.

In dealing with the VLP task, our algorithm handles it neatly without increasing latency and compromising real-time performance. In any VLP task, there should be at least one sub-task that is actually a FLP task. For example, data formatting in audio CODEC has a constant workload. In a digital audio player, FLP tasks are more apparent, which are the audio post-processing tasks. Exploiting the co-existence of FLP and VLP tasks, the proposed algorithm inserts a buffer to decouple the two processes. Basically, the idea is to execute the VLP task at a high frequency and voltage so that more decoded samples are stored in the buffer. As the FLP task subsequently extracts constant small number of samples from this buffer, only this

FLP task is run in the next few frame intervals. Thus, voltage and frequency can be reduced during the FLP processing frames. Once the buffer level becomes low, the VLP task is executed again to refill the buffer and the FLP task follows.

Our proposed method is a generic, efficient, and dynamic voltage-frequency scaling power management mechanism. This method is written as a set of library functions or a framework that can be inserted to any application under development. It expedited the process of creating power-aware application in the real-world situation. This means that the user is freed from the hassle of measuring the code's execution time, deadline, idle time, optimum voltage and operating frequency. Consequently, the time necessary to market their product is being shortened. In addition, our method also works in the background and monitors the application behavior without the user realizing it. Thus, it is implied that there is no quality difference of the application when being run with and without the proposed DVFS algorithm. The measurement results presented in Chapter 6 show that significant power reduction up to 75% has been achieved using the proposed algorithm.

7.2. Future Works

The implementation of DVFS algorithm in a multi-threaded application may be considered. However, the DVFS algorithm for FLP tasks has to be slightly modified to suit this multi-threaded application. Instead of finding deadline and execution time, the processor's idling time has to be quantified and then used to determine a suitable operating frequency. This is possible because, usually in a multi-threaded

programming environment, there is a predefined idle thread that has the lowest priority and becomes active only when there is no other user thread running or queuing. A possible way to implement the proposed DVFS in a multi-threaded application is by introducing an intermediate thread between user threads and idle thread. This intermediate thread has priority one level higher than that of the idle thread. Therefore, it can be preempted by all other threads except the idle thread. This intermediate thread's main function is to detect how much time the processor is supposed to spend in the idle thread. Based on the obtained idle time, calculation of suitable operating frequency and voltage can also be inserted in this intermediate thread. Figure 7.1 shows the flow between threads in an application with the possible DVFS implementation. Notice that the idle thread will not be visited anymore by the processor as it is being replaced by intermediate thread.

Figure 7.1. Intermediate thread acts as a replacement of idle thread.

A complete embedded system with power management feature may also be realized

by not only focusing on minimizing the processor's power consumption, but also optimizing the power dissipated by other peripheral devices, such as external memory, display, audio, or even hard disk. Low-power mode has become a common technique in reducing power consumption in these devices. For example, various power modes (seek, active read/write, standby, sleep) in the latest hard disks, or brightness control in LCD display. Consequently, power management techniques in the processor have to be extended in order to maximize the utilization of power reduction capabilities of these devices.

The proposed DVFS algorithm can be applied to video decoding applications. As workload variation also exists in the video decoding applications, buffer mechanism is still required to separate the variable-length decoding process and fixed-length process. The size of the buffer might be significantly larger than what can be contained in the internal memory of the processor. As the usage of external memory may be inevitable, the contribution of external memory and cache to overall power consumption has to be considered.

Author's Publications

- [1] Harliono, D. and Gan, W. S., "Implementation of dynamic voltage and frequency scaling on Blackfin processors," in *Proc. International Symposium on Intelligent Signal Processing and Communication Systems*, Dec. 2005, pp. 193-196.
- [2] Harliono, D. and Wijaya, T., "Power-conscious software implementation of signal processing algorithms," presented at International Exhibition Contest on DSP Creative Designs, Tainan, Taiwan, Nov. 2005.
- [3] Harliono, D. and Gan, W. S., "Implementation of dynamic voltage and frequency scaling on digital audio player," (in preparation).

Bibliography

- [1] eTForecasts. <http://www.etforecasts.com>.
- [2] Research and Markets. <http://www.researchandmarkets.com>.
- [3] FCW.COM. <http://www.fcw.com>.
- [4] Mobile Handset Design Line.
http://www.mobilehandsetdesignline.com/howto/batteries_power_management.
- [5] Portable Design Magazine. <http://pd.pennnet.com/>.
- [6] ADSP-BF534/ADSP-BF536/ADSP-BF537 Blackfin® Embedded Processor Datasheet. <http://www.analog.com>.
- [7] C55x Low Power DSPs. <http://www.ti.com>.
- [8] Embedded Processors - Intel Corporation. <http://www.intel.com>.
- [9] Xiph.org foundation. <http://www.vorbis.com>.
- [10] Kao, J., Narendra, S., and Chandrakasan, A., "Subthreshold leakage modeling and reduction techniques," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, Nov. 2002, pp 141-148.
- [11] Nam, S. K., et al., "Leakage current: Moore's Law meets static power," *Computer*, vol. 36, pp 68-75, Dec. 2003.
- [12] Chandrakasan, A., Sheng, S., and Brodersen, R. W., "Low power CMOS digital design," *IEEE Journal of Solid State Circuits*, vol. 27, pp. 473-484, Apr. 1992.
- [13] Schmitz, M. T., Al-Hashimi, B. M., and Eles, P., *System-level design techniques for energy-efficient embedded systems*. Boston: Kluwer Academic Publishers, 2004.
- [14] Burd, T. D., Pering, T. A., Stratakos, A. J., and Brodersen, R. W., "A dynamic voltage scaled microprocessor system," *IEEE Journal on Solid-State Circuits*, vol. 35, pp. 1571-1580, Nov. 2000.
- [15] Chandrakasan, A. and Brodersen, R. W., "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, pp. 498-523, Apr. 1995.
- [16] De, V. and Borkar, S., "Technology and design challenges for low power and

- high performance,” in *Proc. International Symposium on Low Power Electronics and Design*, 1999, pp 163-168.
- [17] Qing Wu, Pedram, M., and Xunwei Wu, “Clock gating and its application to low power design of sequential circuits,” *IEEE Trans. Circuits and Systems*, vol. 47, pp. 415-420, March 2000.
- [18] Benini, L., Siegel, P., and De Michelli, G., “Synthesizing gated clocks for sequential circuits,” *IEEE Design and Test of Computers*, vol. 11, pp. 32-41, 1994.
- [19] Benini, L., Bogliolo, A., Paleologo, G. A., and De Micheli, G., ”Policy optimization for dynamic power management,” *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 813-833, June 1999.
- [20] Jha, N. K., “Low power system scheduling and synthesis,” in *Proc. IEEE/ACM International Conference on Computer Aided Design*, Nov. 2000, pp. 259-263.
- [21] ADSP-BF537 Blackfin® Processor Hardware Reference. <http://www.analog.com>.
- [22] ADSP-BF537 EZ-KIT Lite® Evaluation System Manual. <http://www.analog.com>.
- [23] MPEG-2 and MPEG-4 Advanced Audio Coding (AAC). <http://www.iis.fraunhofer.de/fhg/iis/EN/>.
- [24] MPEG Layer 3 (MP3) Audio Coding. <http://www.iis.fraunhofer.de/fhg/iis/EN/>.
- [25] Games that use Vorbis – XiphWiki. http://wiki.xiph.org/index.php/Games_that_use_Vorbis.
- [26] Vorbis.com. http://www.vorbis.com/music_links/.
- [27] Montnémery, E. and Sandvall, J., “Ogg Vorbis in Embedded Systems,” Master Thesis, Technical University of Lund, February 2004.
- [28] Princen, J. P. and Bradley, A. B., “Analysis/synthesis filter bank design based on time domain aliasing cancellation,” *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP-34 (5), pp. 1153-1161, Oct. 1986.
- [29] Kosaka, A., Okuhata, H., Onoye, T., and Shirakawa, I., ”Design of Ogg Vorbis decoder system for embedded platform,” *IEICE Trans. Fundamentals*, vol. E88, pp. 2124-2130, Aug. 2005.
- [30] Razavi, B., O'Neill, W. E., and Paige, G. D., “Both interaural and spectral cues

- impact sound localization in azimuth,” in *Proc. International IEEE EMBS Conference on Neural Engineering*, March 2005, pp 587-590.
- [31] MIT Media Laboratory. <http://www.media.mit.edu/>.
- [32] Moorer, J. A., "About this reverberation business," *Computer Music Journal*, vol. 3, pp. 13-28, 1979.
- [33] Shin, D., Lee, S., and Kim, J., “Intra-task voltage scheduling for low-energy hard real-time applications,” *IEEE Design & Test of Computers*, vol. 18, pp. 20-30, March-April 2001.
- [34] Vaidya, P. N., Khan, M. H., Morgan, B., and Sakarda, P., “System level adaptive framework for power and performance scaling on INTEL® PXA27X processor,” in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 2005, pp. 657-660.
- [35] Chakraborty, S., Ye Wang, and Wendong Huang, “A perception-aware low-power software audio decoder for portable devices,” in *Proc. Workshop on Embedded Systems for Real-Time Multimedia*, Sep. 2005, pp. 13-18.
- [36] Lorch, J. R. and Smith, A. J., “PACE: A new approach to dynamic voltage scaling,” *IEEE Trans. Computers*, vol. 53, pp. 856-869, July 2004.
- [37] Mosse, D., Aydin, H., Childers, B., and Melhem, R., “Compiler-assisted dynamic power-aware scheduling for real-time applications,” in *Workshop on Compilers and Operating Systems for Low Power*, Philadelphia, Oct. 2000.
- [38] Gan, W. S. and Kuo, S. M., “Teaching DSP software development: from design to fixed-point implementations”, *IEEE Trans. Education*, vol. 49, pp. 122-131, Feb. 2006.
- [39] Gan, W. S. and Kuo, S. M., “Code optimization and power management,” chap. 8 in *Embedded Signal Processing with the Micro Signal Architecture*. N.J.: Wiley-IEEE Press, 2007.
- [40] Pering, T., Burd, T., and Brodersen, R., “The simulation and evaluation of dynamic voltage scaling algorithms,” in *Proc. International Symposium on Low Power Electronics and Design*, 1998, pp.76-81.
- [41] Bavier, A., Montz, A., and Peterson, L., “Predicting MPEG execution times,” in *Proc. International Conference on Measurement and Modeling of Computer Systems*, 1998, pp.131-140.

- [42] Ying Tan, Malani, P., Qinru Qiu, and QingWu, "Workload prediction and dynamic voltage scaling for MPEG decoding," in *Proc. Asia and South Pacific Conference on Design Automation*, Jan. 2006, pp. 911-916.
- [43] Ben Lee, Nurvitadhi, E., Dixit, R., Chansu, Y., and Kim, M., "Dynamic voltage scaling techniques for power efficient video decoding," *Journal of Systems Architecture*, vol. 51, pp. 633-652, Oct.-Nov. 2005.
- [44] Choi, K., Cheng, W. C., and Pedram, M., "Frame-based dynamic voltage and frequency scaling for an MPEG player," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, Nov. 2002, pp. 732-737.
- [45] Chung, E., Benini, L., and De Micheli, G., "Contents provider-assisted dynamic voltage scaling for low energy multimedia applications," in *Proc. International Symposium on Low Power Electronics and Design*, Aug. 2002, pp.42-47.
- [46] Pouwelse, J., Langendoen, K. and Sips, H. J., "Application-directed voltage scaling," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.11, pp. 812-826, Oct. 2003.
- [47] Huang, Y., Chakraborty, S., and Wang, Y., "Using offline bitstream analysis for power-aware video decoding in portable devices," in *Proc. ACM International Conference on Multimedia*, 2005, pp. 299–302.
- [48] Lu, Y., Benini, L., and De Micheli, G., "Dynamic frequency scaling with buffer insertion for mixed workloads," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp.1284-1305, Nov. 2002.
- [49] Huffman, D.A., "A method for the construction of minimum redundancy codes," *Proceedings of Inst. Radio Engineers*, Sept. 1952, pp. 1098-1102.
- [50] Choi, S. B., and Lee, M. H., "High speed pattern matching for a fast Huffman decoder," *IEEE Trans. Consumer Electronics*, vol. 41, pp. 97-103, Feb. 1995.
- [51] Siemiński, A., "Fast decoding of the Huffman codes," *Information Processing Letters*, vol. 26, pp. 237-241, Jan. 1988.
- [52] Katz, D. and Gentile, R., "Optimize performance and power with dynamic power management."
<http://www.embedded.com/columns/showArticle.jhtml?articleID=193101162>.
- [53] Analog Devices: Blackfin Online Learning and Development (BOLD).
http://www.analog.com/processors/pdf/bold/Blackfin_System_Services_slides.p

[df.](#)

- [54] Using the NI ELVIS DAQ Adapter on the Analog Devices ADSP-BF537 EZ-KIT for Stimulus Response Testing.

[http://zone.ni.com/devzone/cda/tut/p/id/2937.](http://zone.ni.com/devzone/cda/tut/p/id/2937)

- [55] LabVIEW 8.0. [http://www.ni.com/labview/.](http://www.ni.com/labview/)

- [56] VisualDSP++ Development Software.

[http://www.analog.com/processors/blackfin/evaluationDevelopment/visualDSPdevSoftware.html.](http://www.analog.com/processors/blackfin/evaluationDevelopment/visualDSPdevSoftware.html)