System-Level Memory Power and Performance Optimization for System-on-a-Chip Embedded Systems

A Thesis Presented
by
Ke Ning
to
The Department of Electrical and Computer Engineering
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in
Computer Engineering
Northeastern University
Boston, Massachusetts
August 2008
Abstract

Power has become a first rate design issue in microprocessor design. Power efficiency is especially critical for battery-powered embedded systems. Technology trends are making data communication, both on-chip and off-chip, more expensive relative to computation. Evaluating power-performance design trade-offs at the architectural level still requires more research study.

In this dissertation, we will show how microprocessor power, especially in the memory sub-system, is consumed during program execution. We also show that the external memory system in a low power System-on-a-Chip (SOC) embedded system has significant impact on overall system power. The source of memory power consumption is due to the data transmission, bandwidth limitation, and memory access overhead.

We review and summarize the current research work on low power microprocessor architecture design in academic research community and in industry world. The work includes power modeling, power estimation tools and power optimization techniques. In addition, we summarize different power optimization into five categories and compare their effects and impacts to the overall system.
Two solutions are proposed to reduce data bandwidth and to improve the power efficiency on the external memory bus. We first propose an external bus arbitrator to schedule the external bus requests in order to achieve better bus utilization. We propose a series of power aware arbitration schemes for the external bus request scheduling. On average, we observe a 22% performance speed up and 13% power savings compared to traditional arbitration schemes. In our second approach, we present a hardware-based, programmable external memory page remapping mechanism which can significantly improve system performance and decrease the power budget on external memory bus accesses. We employ graph-coloring techniques to guide the page mapping procedure. Our algorithm can significantly reduce the memory page miss rate by 70-80% on average. For a 4-bank SDRAM memory system, we reduce external memory access time by 11%, while reducing the associated power consumed by 11%.
To my wife, my son and my family.
# Table of Contents

Abstract i

Table of Contents iv

List of Tables vii

List of Figures ix

Acknowledgment 1

1 Introduction 3
   1.1 Power Trends in Microprocessor Design .......................... 3
   1.2 Increasing Importance of System-Level Bus ........................... 5
   1.3 Issues Not Addressed ............................................ 7
   1.4 Research Contribution ........................................... 8
       1.4.1 Power Estimation ........................................... 9
       1.4.2 Power Optimization ........................................ 11
   1.5 Dissertation Organization ....................................... 13

2 Background 14
   2.1 Source of Power Consumption .................................... 14
   2.2 Power Model of Microprocessors ................................. 17
       2.2.1 Dynamic Power Model ...................................... 18
       2.2.2 Leakage Power Model ...................................... 19
   2.3 Power Efficiency .................................................. 20
   2.4 Power, Performance and Complexity ................................ 22

3 Microprocessor Power Estimation Techniques 23
   3.1 Approaches .......................................................... 24
   3.2 Dynamic Power Estimation ........................................ 25
       3.2.1 Instruction and Function Level Model ...................... 26
3.2.2 Architectural Model ................................................. 27
3.3 Static Power Estimation ............................................. 30

4 Memory Subsystem Power Estimation .................................. 32
4.1 Previous System Bus Power Modeling ................................. 32
4.2 Issues with Previous Approaches ..................................... 34
4.3 Improved Methodology ................................................ 35

5 Previous Work on Power Optimization of Memory System .......... 39
5.1 Gating ................................................................. 40
  5.1.1 Clock Gating ..................................................... 40
  5.1.2 Voltage Gating .................................................. 41
5.2 Scaling ................................................................. 42
  5.2.1 Voltage Scaling .................................................. 42
  5.2.2 Frequency Scaling .............................................. 44
  5.2.3 Resources Scaling ............................................. 45
5.3 Banking ................................................................. 46
  5.3.1 Vertical Subbanking .............................................. 46
  5.3.2 Horizontal Subbanking ......................................... 47
5.4 Clustering ............................................................... 48
5.5 Switching Reduction .................................................. 49
  5.5.1 Logic Re-design .................................................. 50
  5.5.2 Data Re-encoding ............................................... 51
5.6 Comparisons of Power Optimization Techniques .................... 54

6 Experimental Methodology .................................................. 56
6.1 System-on-a-Chip (SOC) System Architecture ....................... 56
  6.1.1 System Bus Hierarchy .......................................... 56
  6.1.2 External Bus Interface Unit .................................... 58
  6.1.3 External Memory (SDRAM) Model ............................... 59
6.2 Bus Power Modeling and Experimental Setup ......................... 63

7 Optimization I: Bus Traffic Arbitration ................................ 66
7.1 Introduction ........................................................... 66
7.2 Proposed Algorithm ................................................... 67
  7.2.1 Traditional Algorithms ......................................... 67
  7.2.2 Power Aware Algorithms ....................................... 68
7.3 Experiments ............................................................ 71
  7.3.1 Benchmarks ...................................................... 71
  7.3.2 Results ........................................................... 75
7.4 Comparison with Exhaustive Algorithm ................................ 85
7.5 Implementation Consideration ........................................ 86
  7.5.1 Power estimation unit (PEU) ..................................... 86
  7.5.2 Two Architectures ............................................... 89
  7.6 Summary of Bus Arbitration ........................................ 92

8 Optimization II: Memory Page Remapping ......................... 93
  8.1 Introduction ....................................................... 93
  8.2 Data Access Pattern in Embedded Multimedia Applications ... 94
    8.2.1 Fixed Stride ................................................. 96
    8.2.2 2-Way Stream ............................................... 97
    8.2.3 2-D Stride .................................................. 97
  8.3 Related work on Data Access Pattern ............................. 99
    8.3.1 Data Pre-fetching .......................................... 100
    8.3.2 Memory Customization ...................................... 101
    8.3.3 Data Layout Optimization .................................. 102
    8.3.4 Cost Prediction ............................................. 104
  8.4 External Memory Pages Revisited .................................. 104
  8.5 Page Remapping Algorithm ......................................... 108
    8.5.1 Problem Presented .......................................... 109
    8.5.2 Motivation of Page Remapping .............................. 111
    8.5.3 Remapping Algorithm ....................................... 112
    8.5.4 Page Transition Graph ..................................... 115
    8.5.5 Page Coloring .............................................. 116
    8.5.6 Example ..................................................... 120
  8.6 Proposed Architecture ............................................ 125
    8.6.1 Implementation Issues ..................................... 128
  8.7 Experiments ........................................................ 130
    8.7.1 Methodology ................................................ 130
    8.7.2 Results ....................................................... 132
    8.7.3 Stability .................................................... 138
  8.8 Summary of Memory Page Remapping .............................. 138

9 Summary and Future Work .............................................. 142
  9.1 Contributions .................................................... 142
    9.1.1 System Bus Power Estimation ............................... 142
    9.1.2 Bus Arbitration ............................................ 143
    9.1.3 Memory Page Remapping .................................... 144
  9.2 Future Work ........................................................ 145

Bibliography .................................................................. 146
List of Tables

5.1 The power, performance and complexity impacts of different power optimization techniques. ............................................. 55

6.1 SDRAM Commands Truth Table .................................................. 60

7.1 Benchmark Descriptions .............................................................. 72

7.2 External bus power savings of (1, 0) arbitration vs. fixed priority arbitration in sequential command SDRAM. ............................................. 82

7.3 External bus speedup of (1, 0) arbitration vs. fixed priority in sequential command SDRAM. ................................................................. 82

7.4 External bus power savings of (1, 0) arbitration vs. fixed priority arbitration in pipelined command SDRAM. ............................................. 83

7.5 External bus speedup of (1, 0) arbitration vs. fixed priority in pipelined command SDRAM. ................................................................. 83

7.6 Power improvements for pipelined vs. sequential command mode SDRAM. 84

7.7 Speed improvements for pipelined vs. sequential command mode SDRAM. 85

7.8 Performance Comparison between Greedy Algorithm and Exhaustive Algorithm .......................................................... 87

8.1 Benchmarks used in the experiments and their important characteristics. .......................................................... 130

8.2 Miss rate comparison of all multimedia benchmarks. "Original" and "Remap" columns are the page miss ratio without and with page miss module. .......................................................... 133
8.3 Benchmark page miss reduction rate of 2-bank, 4-bank and 8-bank SDRAM memory systems. .................................................. 134

8.4 H.264 encoder benchmark results with two different input data sets:
   (i) CL and (ii) FB. A profile of CL input data set was used to construct
   the PMT in both cases. .................................................. 139

8.5 H.264 decoder benchmark results with two different input data sets:
   (i) tennis and (ii) carousel. A profile of tennis input data set was used
   to construct the PMT in both cases. ................................. 140
## List of Figures

2.1 Structure of CMOS inverter .............................. 14  
2.2 (a)0 → 1 and (b)1 → 0 transitions of the CMOS inverter ... 16  
4.1 External Bus Power Estimation Data Flow. ....................... 36  
6.1 Target embedded multimedia system architecture. ................. 58  
6.3 Page Miss READ with PRECHARGE ......................... 62  
6.4 Timing diagram showing two memory accesses for both sequential and pipelined command SDRAM. ................................. 63  
6.5 ADSP-BF533 EZ-Kit Lite Board (courtesy of Analog Devices) .... 64  
7.1 Hamiltonian Path Graph ....................................... 69  
7.2 Benchmark external bus access frequency ........................ 74  
7.3 External bus power/delay results for MPEG-2 video encoder and decoder. 76  
7.4 External bus power/delay for H.264 Encoder and Decoder. ....... 77  
7.5 External bus power/delay for JPEG image encoder and decoder. .... 78  
7.6 External bus power/delay for PGP encryption and decryption. ..... 79  
7.7 External bus power/delay for G.721 Voice Encoder and Decoder. ... 80  
7.8 A case to compare greedy and exhaustive algorithm............... 86  
7.9 Block Diagram of Power Estimation Unit(PEU) .................. 88  
7.10 Shared power estimator unit (1, 0) power-aware bus arbitrator architecture ........................................... 91  
7.11 Dedicated power estimator unit (1, 0) power-aware bus arbitrator architecture ........................................... 91
7.12 Shared and dedicated implementation of MPEG-2 encoder/decoder benchmark performance. .......................................................... 91

8.1 Fixed stride data access pattern and pseudo code ......................... 97

8.2 2-way stream data access pattern and pseudo code ....................... 98

8.3 2-D stride data access pattern and pseudo code ......................... 98

8.4 Data memory page layout for a memory system of four banks. An "X" indicates that the corresponding page is in use. A "*" indicates that the page is open. .......................................................... 105

8.5 SDRAM access signal diagram. Two consecutive data reads are issued to SDRAM. In case (a), two reads are from same memory bank but different pages, which is a page miss situation. In case (b), two reads are from two different banks. If the second page is currently open, PRECHARGE and ACTIVATE commands are not needed. .............. 107

8.6 Memory page remapping system diagram. The "PAGE REMAPPING" block is the functional block to perform page remapping. .................. 113

8.7 Example memory page layout and Page Transition Graph of a 4-bank SDRAM memory system. Clear blocks indicate occupied memory pages and gray blocks indicate unused memory pages. ............. 115

8.8 Page coloring step 1. Figure contains memory page layout, the selected mapping edge and weight parameter updates. ..................... 123

8.9 Page coloring step 2. Figure contains memory page layout, the selected mapping edge and weight parameter updates. ..................... 124

8.10 Page coloring step 3. Figure contains memory page layout, the selected mapping edge and weight parameter updates. ..................... 124

8.11 Page coloring step 4. Figure contains memory page layout, the selected mapping edge and weight parameter updates. ..................... 124

8.12 Page coloring step 5. Figure contains memory page layout, the selected mapping edge and weight parameter updates. ..................... 124

8.13 Page coloring step 6. Figure contains memory page layout, the selected mapping edge and weight parameter updates. ..................... 125
8.14 Page coloring step 7. Figure contains memory page layout, the selected mapping edge and weight parameter updates. 

8.15 Page coloring step 8. Figure contains conflict resolving and final memory mapping.

8.16 Page remapping table (PMT) for example. "00", "01", "10" and "11" are two bits information and "XX" indicate undetermined values.

8.17 Bank address remapping block architecture. Memory in example is a 16 MByte 4-bank SDRAM.

8.18 Multimedia benchmark page miss ratio.

8.19 External bus average power and access delay of multimedia benchmarks in 2-bank external memory system with or without page remapping.

8.20 External bus average power and access delay of multimedia benchmarks in 4-bank external memory system with or without page remapping.

8.21 External bus average power and access delay of multimedia benchmarks in 8-bank external memory system with or without page remapping.

8.22 Average request delay in system bus clock cycle for multimedia benchmarks with 2-bank, 4-bank and 8-bank memory systems. Both original and page remapped programs are simulated.

8.23 Average external bus and memory power for multimedia benchmarks with 2-bank, 4-bank and 8-bank memory systems. Both original and page remapped programs are simulated.
Acknowledgment

This dissertation would not have been possible without the contributions and support from many people over the years. My sincere gratitude goes to my family, my adviser, my colleagues in Analog Devices and all my friends for their love and supports.

First of all, I thank my adviser, Professor David Kaeli, for his guidance, enlightenment, and encouragement. Professor Kaeli spent enormous amount of time to guide me through my graduate research. He encouraged me for keeping my faith, enlightened me for new ideas and helped me through the difficult time. He also gave me valuable advices both on my professional career and my personal life. He is truly a brilliant adviser, an excellent role model and a wonderful friend. My thanks also go out to my committee member Professor Xinping Zhu and Professor Waleed Meleis. Their suggestions provided great improvement to my research work.

My graduate studies would not have been the same without the supports of my colleges in Analog Devices. They assisted me in various ways during the course of my study. I am especially grateful to Marc Hoffman, Gabby Yi, Rick Gentile, Boris Lerner, Kaushal Sanghai and Seth Molloy for supporting me in the last five years and for their useful inputs to my dissertation. I also need to thank Rasekh Rifaat for
providing many suggestions and substantive ideas to help me improve my arguments.

Finally, I thank my wife and my parents who have done so much for me to make my graduate study possible. This dissertation is dedicated to them.
Chapter 1

Introduction

1.1 Power Trends in Microprocessor Design

In recent years, the pipeline depth for microprocessors has increased dramatically. Reducing the amount of work done in each stage of instruction execution is an efficient way to improve processor performance. Usually, microprocessor performance is measured by how many instructions a microprocessor can execute in unit time. Many techniques such as increasing the degree of instruction-level parallelism (ILP) and out-of-order instruction issue have contributed significantly to improving processor performance. Unfortunately, as performance keeps increasing, microprocessor designs have become much more complex. Currently, high-end processors contain on the order of 500 million transistors and researcher have predicted to have billion-transistor microprocessors [44, 93].

Power consumption has become a major concern for modern microprocessor designers, especially in mobile and hand-held devices. Even on non-battery powered devices, such as personal computers, microprocessor power is one of the most critical
design issues which must be planned in the earliest stages of architectural design for microprocessor designers [60].

The convergent applications in microelectronic devices lead to the design and integration of increasingly complex systems. The integration and complexity levels imposed by newer system-on-a-chip (SOC) approaches aim to integrate heterogeneous functional modules in a single VLSI circuit whose complexity can reach up to millions of logic gates. Designing these systems involves significant design complexity and requires us to work at higher levels of design abstraction to resolve these design issues effectively.

Due to the complexity of SOC embedded systems and restrictive power/performance design requirements, new design techniques have been proposed recently [7, 25, 35, 59, 92]. Real-time embedded systems are implemented as mixed software-hardware systems. Generally, software is used for features and flexibility, while hardware is used for performance. Design of embedded systems can be subject to many different types of constraints including timing, area, power consumption, reliability, and cost. The availability of a suitable hardware/software co-design methodology is a key issue for future designs of complex systems with stringent power and performance requirements [33, 80]. The hardware/software co-design approach for designing embedded systems requires specifying and designing hardware and software separately, along with the analysis of digital circuits and systems at the highest abstraction levels.
Computer-aided design (CAD) tools have also improved to accommodate the increasing design constraints. They have had a profound impact on our ability to employ a hardware/software co-design philosophy. Effectively exploiting those CAD design tools should increase system quality and shorten development cycle [58]. Furthermore, power estimation and optimization techniques can be integrated into the design tools for hardware/software co-design [45]. High-level power models are generally used in CAD design tools to provide an early estimation of the power budget. The high-level power budgeting is used to shorten the design turn-around time, while exploring a range of architectural design choices in early development stages. Accuracy and efficiency must be traded off in order to meet the overall power and performance requirements.

1.2 Increasing Importance of System-Level Bus

The performance gap between the speed of microprocessors and the speed of bus interfaces have forced designers of system-level buses (off-chip external buses) to increase bus bandwidth dramatically. Both data and address buses have become wide and hierarchical on modern microprocessor-based systems. In an SOC embedded system, system-level buses connect embedded processor cores, I/O peripherals, direct memory access (DMA) and off-chip memory. The power on external buses is a significant portion of the overall power in the system, often more than the microprocessors themselves [71]. Because of that, the careful power design on embedded system has become a challenging task.
In systems that involve high data bandwidth (e.g., image compression or video processing), the major cost of the area and power is not due to the data path or control path, but due to the global communication and memory interactions. Shiue and Chakrabarti showed that embedded applications expend 50-80% of their power on transfers data between microprocessors and off-chip memories [81]. In order to reduce system-level power consumption, it is very important to focus on design strategies that can reduce the power consumption produced by memory operations. Two parameters are commonly used to quantify the performance of the system-level memory system: 1) latency, and 2) bandwidth. While much research work in the past has focused on reducing memory latency and increasing bandwidth, the focus has just started that considers power in this tradeoff. Recently, a number of new techniques have been proposed from the circuit design community and have been applied at the system algorithm level. Amruter proposed to reduce energy by limiting the swing of bitlines and by controlling the local wordline drive strength [2]. Several other approaches have been proposed to use multi-threshold voltage CMOS to reduce the idle memory power [62]. Rixner et al. proposed memory access scheduling algorithms to reduce memory module energy [74].

As system-level power begins to draw more and more attention from the research community, the demand for simulation tools and power aware design methodologies has also increased. More advanced system level tools need to be built to resolve high level design issues, profile memory accesses and design hierarchical memory systems.
These issues can have a significant impact on the final system performance. As we mentioned before, there is clearly a need to study high-level design techniques and create appropriate tools to explore power/performance trade-offs. It is an especially critical issue for power critical applications such as portable audio and video algorithms.

1.3 Issues Not Addressed

Although there is increasing interest in power-aware designs, this field is still relatively young and open to new ideas. Many issues are still unresolved, especially when considering access to external memory.

For example, in order to find algorithms to optimize external memory and bus power, an efficient and accurate methodology to estimate the external bus power is required. Most previous external memory research has been based on the assumption that power is proportional to the Hamming distance of the data and address values of two consecutive memory I/O requests. However, the behavior of external bus power is correlated to external memory communication protocols, which depend on memory types and technologies. As of today, there are no external memory power estimation techniques that accurately take into account all of the system-level bus power based on dependency of memory technology.

Another open issue is external memory power optimization. Many memory optimization techniques have been developed in the past decade, but the majority of them
focus on the cache power. Most of the previous research tries to resolve the problem from a hardware and software comprehensive perspective. Very little research focuses on external memory power or utilizes the physical characteristics of external memories.

The microprocessor systems we consider in this dissertation are based on complex VLSI SOCs implemented in CMOS technology and designed for embedded applications. Two main topics have been considered: power estimation and optimization. Unlike most of previous research work, which addresses either power estimation or power optimization and treats them as separate goals, our research work demonstrates how power optimization can be much more effective when it is tightly integrated with power estimation. It can create more effective search algorithms for optimized solutions.

1.4 Research Contribution

In this dissertation, we have focused on software and hardware design optimization of the system bus and external memory subsystem. The main contributions of this dissertation are from two major areas. The first area concerns the definition of a power assessment framework from a system-level perspective. The second area aims at developing novel power optimization techniques. The experimental model we developed allows us to consider performance/power trade-offs on system-level memory accesses, and to optimize the overall system power efficiency. The proposed estimation and optimization techniques are coupled and integrated into our embedded
system-on-a-chip (SOC) system model.

Throughout our research work, substantial efforts have been devoted to demonstrate the effectiveness of the proposed methodologies. The context of implementation and application are embedded multimedia related benchmarks. Experimental results have been produced in order to show the practical applicability of the proposed estimation and optimization techniques. A large number of simulations are conducted to demonstrate the advantages provided by the proposed techniques with respect to previous approaches.

1.4.1 Power Estimation

The primary research in this thesis targets external bus power in a system-on-a-chip (SOC) microprocessor system. First, a model for static and dynamic power estimation on the external bus is defined. The power estimation model operates at software instruction level, which is tightly related to the target processor and memory simulation framework. The dependency of microprocessor and memory type is also taken into account. The microprocessor and memory bus architecture defined in the framework is a very generic single bus SOC embedded architecture, which makes our work to be easily adapted to other types of system architectures.

Our power estimation model includes the power consumption associated with system-level buses, which represents one of the major contributors to the overall power budget. The model evaluates the data activity on the off-chip buses at the system-level. It provides an experimental model to assess the system-level address, data and
control bus power, associated pin-pad power and corresponding memory/peripheral access power. One unanticipated contribution of the research was the definition of a framework suitable for achieving accurate and efficient estimation of the power consumption associated with a microprocessor system. This framework also accounts for the power of memory state transitions and memory stalls.

The relative power figures, such as power delay products etc., can also be generated from the power estimation models. They are intended to guide us in exploring the relative impacts of different design alternatives on the quality of the final design rather than to provide absolute and accurate power metric. The main purpose is to provide us the capability of analyzing and comparing different solutions in a large architectural design space. Our experimental results have shown how the model can be effectively used to configure the system-bus architecture from the power/performance stand point of view.

In the dissertation, the primary architecture model used is based on the Analog Devices Blackfin ADSP-BF532 embedded microprocessor architecture. It is a single core SOC system designed specifically for multimedia applications. The compiler and simulator tool-chains are used for the Blackfin processor design and Blackfin application development. An experimental environment is developed to complete the dissertation research. The architecture model has been used and validated in several of our previous publications[63, 64].
1.4.2 Power Optimization

The second goal of our research is to design and evaluate new power minimization techniques operating at a system level. The proposed techniques can reduce the power dissipation associated with the system-level (external) memory system. The power associated with system-level buses is primarily due to the large capacitance of the I/O lines, one of the largest contributors to total system power. Reducing the amount of switching activity on the memory bus pins can provide significant power savings to the overall system.

We have investigated the characteristics of external memory activities over a number of embedded benchmarks on the system-on-a-chip (SOC) embedded system architecture. To optimize the performance, we need to minimize energy dissipation while satisfying the given real-time and data throughput constraints. A major contribution of this dissertation is to utilize power estimation at a high-level of design abstraction to explore multiple power-aware design options and develop optimization techniques for power reduction and performance improvement. The focus is on minimizing the switching activity and the number of stall cycles on the external address and data buses.

The traditional optimization technique for the external bus is to rely on bus address and data encoding, including static and dynamic coding schemes. We compared dynamic power management techniques (as opposed to compiler-directed techniques) and studied the problem of determining power-aware memory traffic generation on the
external memory bus. Although some of the encoding techniques reduce the power consumption, their major limitations are that they have little impact on bus bandwidth and access delay. When the encoding scheme is too complicated, the negative impact on performance can be noticeable. To address some of the issues mentioned above, we have developed two different optimization techniques.

First, we propose a series of power-aware memory access arbitration schemes for external bus accesses[64]. Memory access requests in the model come from microprocessor cores, data caches, code caches, memory DMA’s and peripheral DMA’s. Experimental results show that the proposed arbitration schemes have significant advantages over traditional arbitration approaches on both power dissipation and access delays. From an implementation point of view, one of the schemes is preferred over the other in terms of hardware implementation complexity and with comparable performance. The arbitration algorithm is implemented in the external bus interface unit (EBIU), which is a common component in system-on-a-chip (SOC) embedded systems.

Second, we propose a new scheme that addresses communication between microprocessors, memories the I/O sub-system[65]. The proposed solution can take full advantage of the memory access patterns during application execution. Our research aims at reducing the switching activity of system-level buses and memory state transition stalls based on spatio-temporal correlation of memory accesses. Based on
previous findings on memory access patterns, a programmable hardware-based external memory page remapping mechanism is presented, which can significantly improve performance and decrease the power budget due to external memory bus accesses. The new approach is developed by studying common data access patterns presented in embedded multimedia applications. An efficient algorithm to map application data memory into external memory pages is developed, which employs graph-coloring techniques to guide the page mapping procedure.

1.5 Dissertation Organization

This dissertation is composed of nine chapters. First, the current state-of-the-art in power modeling for general-purpose microprocessors is reviewed in Chapter 2. A variety of architecture-level power estimation techniques is categorized in Chapter 3. Chapter 4 is focused on the power estimation techniques on the memory systems. Some of the issues with those techniques are presented and the improved power modeling framework to overcome some of the previous issues is described. Chapter 5 summarizes previous work on power optimization. Chapter 6 describes the experimental setup and system-on-a-chip (SOC) architecture assumed in this thesis. Chapter 7 presents a new bus arbitration algorithm and simulation results to demonstrate the potential for this design option. Chapter 8 presents a second optimization technique which recognizes data access patterns in order to remap external memory pages. Finally, Chapter 9 concludes the thesis and presents directions for future work.
Chapter 2

Background

2.1 Source of Power Consumption

Most microprocessor systems are fabricated using CMOS technology. The increasing importance of power related issues has driven research efforts to understand the power dissipation at both the circuit level and the system level. The main sources of power dissipation in digital CMOS circuits are reviewed in this section.

Power dissipation in CMOS circuits is composed of static, dynamic and short-circuit components. Static power is due to the leakage current. Dynamic power is

![Figure 2.1: Structure of CMOS inverter](image)

Figure 2.1: Structure of CMOS inverter
due to the charge and discharge of the circuit node capacitance at the output of each logic gate. Short-circuit power is the short-circuit current from the supply to the ground voltage during output transitions. Figure 2.1 shows the structure of the generic static CMOS inverter. The pull-up network is built with PMOS transistors ($T_1$ for the selected inverter) and it connects the output node $V_{out}$ to the power supply $V_{dd}$. Conversely, the pull-down network is composed of NMOS transistors ($T_2$ for the selected inverter) and it connects the output node to the ground node $V_{ss}$. In CMOS gates, the structure of the pull-up and pull-down network is to create an isolation that when the circuit is stable (i.e., after the output rises or falls and transients are exhausted) the output node is never connected to both $V_{dd}$ and $V_{ss}$ at the same time.

When an input transition causes a change in the conductive state of the pull-up and the pull-down network, the electric charge is transferred from the power supply to the output capacitance $C_L$ or from the output capacitance to ground. The transition causes power dissipation on the resistive pull-up and pull-down networks, as shown in Figure 2.2. The charge lost during the CMOS state transition is the main source of dynamic power, also called switching power.

Short-circuit power is due to the fact that, during the state transition of a CMOS transistor, the PMOS and NMOS may conduct simultaneously in a very short period of time. During the time that both PMOS and NMOS are open, they can form a short-circuit connection between $V_{dd}$ and $V_{ss}$. Short-circuit power is usually a small fraction of the total power dissipation, approximately 10% [8].
Leakage current in a CMOS transition consists of two components: the diode leakage current due to the reverse currents in the diffusion region of PMOS and NMOS area and the sub-threshold current flowing through the transistor when the transistor is off [14]. Leakage power is also called static power.

Minimization of the average power consumption is a crucial issue for the design of electronic components and it is the primary focus of our work. The reduction of the peak power dissipation is another important design concern. Excessive peak power may cause a circuit to fail due to electro-migration and voltage drops on supply and ground lines. However, peak power considerations are mainly related to design techniques for worst-case conditions and the study of the peak power is outside of the scope of the present work.
2.2 Power Model of Microprocessors

Power modeling at the register transistor level (RTL) is relatively mature and accurate. However, the microprocessor power consumption is from millions of CMOS transistors. To model the power behavior of a million-transistor microprocessor executing large scale application programs at the RTL level could take an unrealistically long period of time and it is impractical to use at early design stages. The modern power-aware design rule is: different levels of power abstraction are needed for different stages of the design process. The primary objective of power estimation aims at assisting during the system and architectural-level design steps. To speed up the power estimation process, some use of abstraction is needed to simplify the simulation process in order to achieve reasonable simulation throughput. The fundamental contributors to microprocessor power consumption are defined in the following equation:

\[ P = P_{\text{dynamic}} + P_{\text{leakage}} \]  

(2.2.1)

There are two terms in Equation (2.2.1). The first term measures the dynamic power consumption caused by the charging and discharging of the capacitive load on the output of each gate. The second term is the power lost due to leakage current that is present regardless of the state of the gate. It is the current that leaks through transistors even when they are turned off [57]. There is another source of power
consumption that is not shown in above equation. It is short-circuit current. Short-
circuit current is due to the momentary short circuit current flow through the power
supply and ground when the CMOS switches its gate. The loss due to short circuit
current is relatively small. It is associated with the dynamic power, and the equation’s
dynamic power part can dominate the total power [42].

2.2.1 Dynamic Power Model

In current CMOS technology, the dominant power consumption element for a micro-
processor is dynamic power. Every transition of a digital circuit consumes power,
because every charge or discharge of the digital circuit’s capacitance drains power.
It is the power that a microprocessor uses for computation. It is proportional to the
toggling frequency and the capacitance of the transistor load. In general, the dynamic
power can be expressed as:

\[ P_{\text{dynamic}} = ACV_{DD}^2 f \]  \hspace{1cm} (2.2.2)

Where \( f \) is the frequency of the operation of the system; \( A \) is called the activity
factor, which is dependent on the resource usage of the processor; \( V_{DD} \) is the sup-
ply voltage and \( C \) is total capacitance seen by the gate outputs. It is clear from
Equation (2.2.2) that reduction of \( V_{DD} \) is the most effective way to lower the power
consumption. However, lowering \( V_{DD} \) creates problems of increased circuit delay and
reliability. The correlation of circuit delay and supply voltage is:
where $\tau$ is the propagation delay of the CMOS transistor, $V_T$ is the threshold voltage, and $V_G$ is the input gate voltage [35]. The propagation delay restricts the clock frequency in a microprocessor. From Equations (2.2.2) and (2.2.3) we can see that there is a fundamental trade-off between switching speed and supply voltage. Processors can operate at a lower supply voltage, but only if the clock frequency is reduced to tolerate the increased propagation delay [70]. The implicit constraint is that the propagation delay of the critical path delay of a microprocessor must be smaller than $\frac{1}{f}$ in order to make the microprocessor functional.

### 2.2.2 Leakage Power Model

Dynamic power used to be the dominant source of power consumption in CMOS technology. However, with the increase in speed and chip density, total power dissipation from circuit leakage is approaching the same level as dynamic power, and it is predicted to exceed the dynamic power as technology drops below the 65-nm design rules. The most influential leakage power model developed so far is the Butts-Sohi model [14]. They quantify static power using a lumped parameter model that maps both technology and design effects.

$$P_{\text{leakage}} = V_{DD} N k_{\text{design}} I_{\text{leakage}}$$  \hspace{1cm} (2.2.4)
Chapter 2. Background

$N$ is number of transistors in the design. $k_{\text{design}}$ is a design dependent parameter, and it is an empirically determined parameter to represent the characteristics of the design. $I_{\text{leakage}}$ is a technology dependent parameter and it is highly correlated with threshold voltage $V_{th}$. Decreasing the threshold voltage can decrease the leakage power significantly, but it can also make the transistor slower. There are a few ways to reduce the static power. The first method is to reduce the supply voltage, using low $k_{\text{design}}$ circuit, and the second method is to introduce multiple threshold voltages.

2.3 Power Efficiency

Evaluation of architectural power consumption and power efficiency is complicated. There are no simple rules to evaluate a processor’s speed, power and complexity together. To analyze the power alone is not very convincing, because there is always a trade-off with performance. Decreasing power can have performance or complexity penalties; otherwise the original design can be optimized by itself. Energy consumption per instruction measured in Joules/Instruction or its inverse MIPS/W, has been shown to be an unbalanced metric to measure and compare microprocessor designs. As in Equation (2.2.2), power is proportional to the capacity, voltage and clock frequency. To reduce the power, one can simply: 1) decrease the capacitance by using smaller transistors, 2) reduce the supply voltage, or 3) reduce the clock frequency. All three adjustments would increase the delay of the circuits and make the microprocessor run slower. Therefore, the most power efficient processor becomes the slowest
To address the trade-off between power and performance, two metrics that have been shown to better capture the power/performance tradeoff are 1) energy delay product (inversely $MIPS^2/W$) and 2) energy delay squared product (inversely $MIPS^3/W$). They are commonly used in most of the recent literature. Both metrics are called power efficiency [28, 53]. The energy delay squared product has the property of being voltage and frequency independent. It means when the working condition of a chip is adjusted, the energy delay squared product is relatively constant. It is a good metric to compare the power efficiency of two chips, but it is hard to make the conclusion that the one with higher energy delay squared product is more power efficient. The energy delay product and the energy itself also need to be considered accordingly. The issue of using the right metric in the right context is still being debated in the low-power microprocessor community.

There are some issues with the $MIPS^n/W$ type of metrics. The exponent $n$ is global (homogeneous throughout a system) and $n$ can only be an integer number, while the power-performance trade-off is typically local and non-homogeneous in a chip. Hardware intensity ($\eta$) is a new power metric proposed recently to model the relationship between architectural complexity and the power supply [108]. It is supposed to overcome some of the above issues. The hardware intensity is the ratio of the relative increase in energy ($E$) to the corresponding relative gain in performance ($D$) achievable through logic redesign at a fixed operating condition.
Hardware intensity provides a number of new quantitative relations between power and performance. It is expressed as:

\[ \eta = -\left( \frac{D}{E} \frac{\partial E}{\partial D} \right)_V \] (2.3.1)

### 2.4 Power, Performance and Complexity

Besides speed and power dissipation, minimum complexity is another design goal for modern microprocessors. Higher complexity means higher verification costs and a larger number of transistors. Complexity leads directly to increased leakage power, but the power and complexity do not necessarily have complementary causal effects. The added complexity could control the component to use power more efficiently. For example, gating and scaling are two common techniques to decrease the power, as long as the newly introduced circuits consume less power than what it could possibly save. In that case, the increased complexity should reduce overall power consumption.

Design complexity is difficult to measure. It is not simply computed as the number of transistors on the die. It is related to the design complexity and product yield. Array structures such as caches and memories take up more than half of the transistors on typical microprocessors, but they are not as hard to manage as combinational or control logic, which is only a small portion of the area. Both power and complexity need to be considered separately due to their subtle interdependency [9].
Chapter 3

Microprocessor Power Estimation Techniques

A variety of techniques are available to reduce power, but it is difficult to quantify the power benefits at an early design stage. Traditionally, power estimation is performed at the CMOS level, typically relying on a gate-level CAD tool to provide power characterizations for each transistor. However, it is a very time-consuming process and transistor level design is not available at an early design stage, which makes it difficult to be utilized. The current power-aware design methodology uses different levels of power abstraction for different stages of the design process. High-level modeling and power estimation techniques can speed up the power estimation process. Certain forms of the abstraction are needed to simplify the simulation process in order to achieve higher simulation speed. The simulation process aims at ensuring a high degree of fidelity, rather than focusing on accuracy [37].
Chapter 3. Power Estimation Techniques

3.1 Approaches

Power estimation methodologies utilized at different levels of the design flow are fundamental elements of an effective and accurate low-power design methodology. At each abstraction level, the corresponding estimation tool should provide feedback on the results of the logic synthesis and optimization. This approach implies, at each level, a loop composed of tightly correlated optimization and estimation to allow for the exploration of multiple design alternatives [83].

Currently, simulation-based techniques are the dominant methodology for CMOS circuit power estimation. The simulation tools simulate the circuit with a set of inputs for well-known operations. The accuracy of the simulation relies on the accuracy of the circuit timing and power models. Another option is to use probabilistic methods which combine a large number of input events into a set of probability values that require a single pass through a simulation to collect all the statistics. The probabilistic methods require the specification of the typical behavior of the input through their probabilities, which might involve running non-trivial modeling applications to figure out the average switching activity. Another power modeling approach referred to as statistical methods, uses random inputs to feed into a simulation model until certain criteria are satisfied [78]. Statistical methods are usually faster than probabilistic techniques and pure simulation based methods, but sacrifice some accuracy.

In previous research, dynamic and static power dissipation were treated as two separate estimation processes. In reality, these two types of power dissipation are not
completely decoupled. In our research work, we focus on one estimation process to estimate dynamic power estimation and static (leakage) power together with a high degree of accuracy.

3.2 Dynamic Power Estimation

For microprocessor system architecture design, understanding the system-level power dissipation is much more important than the power dissipation for each transistor gate. The lower level model is based on the usage of gate-level simulation and power estimation tools on a transistor design of the processor. This is the most accurate method, assuming that a detailed transistor-level description of the processor is available. However, in most cases, this method cannot be applied due to the lack of information related to the processor description, especially if the processor is unavailable to the public. System level power estimation is less accurate, but much faster. Much progress has been made in recent years to apply system-level power estimation techniques to microprocessor system designs. Some approaches require models of the processor at the component level; some do not require any knowledge of the internal structure of a microprocessor at all. Some approaches quantify the whole system power, while others only apply to a portion of the system or to only consider a component of power dissipation (i.e., dynamic or static power).

General surveys of power estimation techniques at different abstraction levels can be found in [20, 34, 59, 100]. Up until now, a majority of prior power estimation
Chapter 3. Power Estimation Techniques

models have been proposed at the gate, circuit and layout levels, in which the state-of-the-art can be considered mature enough and most of the EDA vendors provide effective power analysis tools. In this section, we summarize previous work in this area compare the advantage and disadvantage between them.

3.2.1 Instruction and Function Level Model

The instruction level power model is typically used for off-the-shelf processors, whose circuit level structures are not available. It profiles a microprocessor’s power consumption after the chip is taped out. The model can provide feedback to the hardware design team for verification purposes. It can also provide improvement strategies for next generation designs. Instruction level power modeling is done by experimentally measuring the electronic current drawn by a microprocessor while executing the same instruction repeatedly in order to build a database that gives the base cost for each instruction. The inter-instruction effects and memory stalls are also considered and measured in similar steps. The overall power consumption is the average current multiplied by the supply voltage [91]. Instruction-level power modeling is based on the use of software instructions as stimulus for the simulation, and average power contribution of each execution instruction during each clock cycle. The average power can be calculated using Equation 3.2.1.

\[ P = V \times I \]  

(3.2.1)
Chapter 3. Power Estimation Techniques

Creation of such an instruction power database may require a non-trivial amount of work. Some improved methods have been proposed to leverage the experimental process. It has been observed that instructions can be arranged in classes such that the instructions in a given class have very similar power cost. A power estimation technique that uses this method is called JouleTrack [84]. It contains a first and second order model based on the classes of the instructions. The accuracy is proved to be very good. The error is less than 3% on StrongArm and Hitachi SH-4 processors based on their experiments.

Another way to accelerate the instruction level estimation is called function-level power estimation [72]. It relies on the use of power macro-models of a library of functions. Each function has its own power characteristics. Actual power dissipation needs to be measured for a large number of experiments. The overall power is the sum of the power for each individual function. The key step when using this technique is to build a "function power database", which is the power data for each function in a library [72].

3.2.2 Architectural Model

Architectural level power modeling is based on the evaluation of the power dissipation for individual components in a microprocessor. It requires knowledge of internal structure and power characteristics for each component blocks. This approach provides a mechanism to explore a variety of design alternatives at early design stages.
The goal of an architectural level model is to meet the design turnaround time while considering a wider array of architectural design alternatives. Accuracy and efficiency of a high-level analysis should contribute to meet the power requirements, avoiding a costly re-design process. In general, the relative accuracy in architecture level power estimation is considered much more important than the absolute accuracy.

The accuracy of any architecture level power estimation technique is determined by the level of hardware abstraction used inside power models. It can be used for a range of CPUs, including out-of-order superscalar cores and in-order VLIW cores. In the 1990’s, cycle accurate simulators were developed to study the performance of microprocessors. The most commonly used in academia is Simplescalar [12, 13]. Most of the architecture level power simulation techniques were implemented based on the Simplescalar framework. Cycle accurate simulation gives more detailed information about hardware activities and transactions. Among all the architecture level power estimation tools being developed today, a majority of them used activity based power estimation.

The Cai-Lim power model [16] is an activity sensitive power model built on top of SimpleScalar. It partitions the SimpleScalar architecture into 17 hardware structures which are further subdivided into 34 blocks. Blocks are partitioned into areas based on power density characteristics. All of the power density and area numbers are pre-computed and included in the simulator. Inactive areas account for 10% of the active power for leakage. The current associated with each active module represents
its power cost and it is assumed to be constant and independent of operand values, circuit state, and correlation with other active modules. TEMPEST is a new version of the Cai-Lim model [21], which includes the power density and thermal effects.

Wattch [10] tracks the activity of a hardware component in a manner similar to the Cai-Lim model, but it does not use the same level of granularity. It groups the processor components into four categories: array structures, fully associative content-addressable memory, combination logic, wires and clocking. This set of components is used to build a parametrized model of major hardware structures. Wattch uses the capacity of each component to calculate the power rather than power density and area (as done in the Cai-Lim model).

To improve the accuracy of modeling, some other architecture level power estimation techniques have been proposed recently. SimplePower [99] and Power Analyzer [57] are two models which are based on input/output transitions rather than activity itself. The transition-based models are more effective for estimating the data dependent power, such as the power consumption for the data bus, ALU, multiplier, register file, etc. The power consumption is different if the processor is fed with another test vector. That difference is more significant for low-end embedded processors.

One of the side effects of power are thermal effects. The thermal package must be designed for worst-case power dissipation. It must be designed for the most severe hot spot that could potentially arise. The HotSpot [85] thermal model is a robust
thermal modeling infrastructure to explore temperature-aware design at the architectural level. It provides for the modeling of temperature distribution and gradients in space and time.

It needs to be noted that, in architecture level power modeling, the power characteristics for hardware components should be obtained from a transistor level model or low level device simulator like HSpice. Some people use instruction execution power data to perform curve-fitting in order to arrive at the internal component power dissipation. Such approaches are misleading, since they make the architectural level approaches simply fall back to instruction level modeling and ignore the fundamental differences between them.

### 3.3 Static Power Estimation

As the number of transistors increase, static (leakage) power becomes a major concern. The previous two models are primarily used for dynamic power dissipation, although some of their implementations implicitly include the effects of static power. In some cases, only the static power is of interest. Designers can use the results of a static power model to redesign their circuit and reduce the leakage power. One of the most influential static power models developed so far are Butts-Sohi’s models [14]. This model quantifies static power using a lumped parameter model that maps both technology and design effects.
\[ P_{\text{static}} = V_{\text{DD}}N k_{\text{design}} I_{\text{leakage}} \quad (3.3.1) \]

In this equation, \( N \) is number of transistors in the design, \( I_{\text{leakage}} \) is a technology dependent parameter and is highly correlated with the threshold voltage \( V_{\text{th}} \), and \( k_{\text{design}} \) is a design dependent parameter that it is empirically determined to represent the characteristics of the design. Decreasing the threshold voltage will decrease the leakage power significantly, but it can also make the transistors slower. There are a few ways to reduce the static power: reducing the supply voltage, using low \( k_{\text{design}} \) circuits or using multiple threshold voltages.

An improved version of the Butts-Sohi’s model is called HotLeakage [106]. It provides the ability to dynamically recalculate leakage current as temperature and voltage change due to operating conditions or dynamic voltage scaling. The model has been integrated into SimpleScalar and it is used as a powerful tool to evaluate the power-performance trade-off.
Chapter 4

Memory Subsystem Power Estimation

It has been known that the memory system consumes a substantial portion of the power budget for modern microprocessors. Memory system usually includes an on-chip static RAM, level 1 and level 2 caches and external memory systems. The high regularity of cache and memory give researchers the opportunity to estimate the cache and memory power with high accuracy. The power characteristics for each type of memory system are different from one another. Our dissertation research work concentrates on the external memory systems and the system level memory. In this chapter, techniques to estimate system and memory power, as derived using some of the power models described in Chapter 3, are explained.

4.1 Previous System Bus Power Modeling

The switching activity of system-level buses is an indicator of the memory and bus related power. Modeling the bus switching activity essentially requires the knowledge
of the bus architecture, memory hierarchy and memory management scheme. By simulating the execution of a program on an SOC microprocessor system, one can evaluate the transition activity of the system-level buses, thus leading to an accurate estimate of the power consumption for those related memories and bus components.

CACTI (Cache Access and Cycle Time) [101] is a synthesis-driven cache power estimation framework. It implements a coarse-grained cache structure based on the hierarchy and size information, using circuit synthesis algorithms. The model is reported to have very good fidelity. It is within 6% of Hspice results, but at a much faster speed. The power dissipation can be estimated based on an access trace generated by a simulator (e.g., SimpleScalar), or with the help of some cache and memory simulators (e.g., Dinero IV) [23]. Dinero is a trace-driven memory hierarchy simulator consisting of various caches connected as one or more trees. The various parameters of each cache can be set separately. Dinero can generate a rich set of memory reference statistics and can also measure bandwidth usage.

Unlike cache and memory in which the majority of the power is drawn from storage memory cells, system bus power consumption is due to the wires that connect the components on the bus and also the driver circuits on each component. The system bus in a system-on-a-chip (SOC) design typically interconnect microprocessors, caches, DMA’s and some addressable devices. The power dissipation of the system bus has different characteristics from the cache power. Bus power dissipation is modeled as:
Chapter 4. Memory Subsystem Power Estimation

\[ P_{\text{bus}} = CV_{DD}^2 f_a + P_{\text{busleak}} \] (4.1.1)

C denotes the capacitance on the bus. \( V_{DD} \) is the bus voltage and \( f_a \) is the bus switching frequency. \( P_{\text{busleak}} \) is the bus leakage power, regardless of the activity on the buses. Some research has been done on the bus system that indicated by using Gray coding on the bus that we can decrease switching activity significantly and reduce dissipation [99]. This approach is successful because the adjacent data values have higher probability to be sent sequentially to the bus and Gray code can provide minimum bit changes in those cases.

4.2 Issues with Previous Approaches

All power-saving encoding techniques and activity based power estimation methods are actually based on an internal memory bus model, where the communication mechanism and memory cell properties are significantly different from what they are on external memory bus. There are many issues related to external bus power that have not been properly addressed in previous work:

- An external bus contains three different components: control bus lines, address bus lines and bus data lines. Previous neglected to consider the control bus power.

- For most of external memory modules (e.g., SDRAM, DDR-RAM), row and
column address lines are shared on the address bus, which is different from how an internal bus works. What we found out is that in many cases the wrong model was used in prior work.

- Memory state transitions and stalls (such as page misses), which impose power and performance penalties, were not considered in the previous models.

The major contribution of this dissertation is the creation of an accurate external bus power estimation model, which overcomes the issues listed above. Our improved power model enables us to evaluate heuristics that can balance power/delay trade-offs associated with external bus data transfers.

4.3 Improved Methodology

Our external bus power includes dynamic power to charge and discharge the capacitance along the external bus, and the pin power to drive the bus signals. There are several unique characteristics of external memory bus power that are different from internal memory bus power.

- Longer physical distance, higher bus capacitance, lower speed.

- Cross line interference, higher leakage current.

- Different communication protocols (memory/peripheral dependent).

- Multiplexed row/column address bus and narrower data bus.
External bus power is highly dependent on the specifics of the memory technology. In past work of bus power modeling, little attention has been paid to the impact of the target memory technology. We made adjustments in our framework to consider the memory model when estimating power consumption. The memory model includes the commands sent on the control bus, the row address and column address on the address bus, and the data values on the data bus. The corresponding leakage power is also considered in the model. All of the components included in the model are summarized in equation 4.3.1.

\[ P_{\text{ExtBus}} = P_{\text{PageMiss}} + P_{\text{BusTurnaround}} + P_{\text{Command}} + P_{\text{Address}} + P_{\text{Data}} + P_{\text{Leakage}} \] (4.3.1)

The proposed model is composed of three main cooperating sub-modules: the memory hierarchy model, external bus interface unit (EBIU) and the address/data
trace stream generator. This model is integrated in our microprocessor simulation environment. The power computation includes dynamic power to charge and discharge the capacitance along the external bus, the pin power to drive the bus current and the leakage current. The simulation data flow is shown in Figure 4.1. In the bus model, we assume that the power to drive the control bus and address bus is the same. During each read/write transaction, the series of commands needed to complete that request are determined. For each command, the bus state transitions, pins toggle and the utilization factor are recorded. Equation 4.1.1 is only applied to a single command or transaction case. For multiple transactions (which are typical in most external bus transactions), Equation 4.3.1 would have much better accuracy to model both system-level bus power and delay cycles. Equation 4.3.1 includes page miss power ($P_{PageMiss}$), bus turn-around power ($P_{BusTurnaround}$), power to issue the command, address or data ($P_{Command}$, $P_{Address}$, $P_{Data}$) and bus leakage power ($P_{Leakage}$). Finally, the average bus power dissipation is calculated over all the commands being executed over the external memory bus.

The external bus power can be quite different if the memory technology is different. In the experiments, we use SDRAM in the power model. The same approach is also applicable to other types of memory modules. The principle of the power modeling is activity driven. The external bus power in each transaction will be determined by the number of bus pins toggled and memory state transitions. The power consumption includes memory-technology-dependent commands sent on the control bus, the row
address and column address on the address bus and the data on data bus. The improved memory power model is the foundation of rest of research work on power optimizations.
Chapter 5

Previous Work on Power Optimization of Memory System

At the system level, the sources of the power dissipation can be grouped under three general categories: (i) processing units; (ii) types of memory; (iii) interconnection. The power optimization techniques can address one or more categories. Some techniques are narrowly focused on specific instances in one category [7], while some techniques are more generally applied to the overall system. Most power reduction techniques reduce power by reducing one or more factors in the power dissipation Equation 2.2.2. Power reduction can be achieved by reducing the activity factor ($A$), power supply ($V_{DD}$) or frequency ($f$). For example, two of the most common and effective ways to save power are gating and scaling. Gating involves turning off the circuits in total or partially if they are not being used. Scaling is to tune the circuits to a lower power consumption mode as long as they are still functional.

In this chapter, an overview is presented on architecture and system level design techniques to optimize power consumptions on both dynamic and static power of
memory related systems. This chapter categorizes four different techniques for power-aware design: gating, scaling, clustering and switch reduction. In each technique, some major achievements in that field are presented.

5.1 Gating

Power is consumed every time the transistor changes states by charging or discharging. Power is also leaked even though there are no state changes. Although some of the circuits in a microprocessor are not being used, they are still fed with the full voltage supply and clock rate. This laziness results in a substantial amount of unnecessary energy cost. Gating is used to turn off those circuits in total or partially for a period of time. The circuits are turned back on when it is time for them to be used again. There are two types of gating techniques. One is called clock gating and the other is called voltage gating.

5.1.1 Clock Gating

A substantial portion of the overall power of a microprocessor is actually spent on the clocks. It is reported to consume up to 40-45% of the total power for microprocessor architectures. The idea of clock gating is straightforward. It removes the clock supply for the part that is not running, which can remove all the dynamic power consumed by those clocks. The implementation of clock gating is non-trivial. The clock gating scheme will introduce clock skews and synchronization problems, which
would increase the circuit design challenges.

5.1.2 Voltage Gating

Voltage gating techniques shut off the voltage supply to unused circuits. Cache decay [41] is one of the successful attempts to reduce the cache leakage power. The \( \text{gated} - V_{DD} \) method turns off the cache lines for a period of time since it has been accessed. Adaptive strategies have been developed to adjust the off-time dynamically according to the program access pattern. The cache line must be reinstated to high-power mode before it can be reused again. The cache line power mode changes can introduce extra delay in the cache access. However, a well developed decay adaptive algorithm can substantially improve the system power-delay square efficiency metric.

In an out-of-order microprocessor, there are quite a few fully associative content-addressable memories (CAM) such as branch target buffers (BTB), translation lookaside buffers (TLB), the instruction decoder queue, integer and floating point queues, register rename tables and load-store queues. Pipeline gating [54] utilizes the branch prediction results to selectively shut off those speculative out-of-order execution units or make them work at a slower rate. There will be some degree of performance loss associated with it, but it proves to be very useful for high-performance out-of-order processors, which have a high degree of control complexity and speculation logic. Results show up to a 38% power reduction with a negligible 1% performance loss.
5.2 Scaling

Scaling is another technique to adjust the power consumption level of a running microprocessor. Instead of shutting down the component completely, scaling transitions a sub-system into a low power consumption mode and still maintains its functionality. The biggest difference between scaling and gating is that scaling allows circuits to maintain their present state, while gating does not. There are three types of scaling techniques: voltage scaling, frequency scaling and resources scaling.

5.2.1 Voltage Scaling

Dynamic Voltage Scaling (DVS) is the most commonly used scaling technique. It is very effective to reduce the dynamic power, because the dynamic power is proportional to the voltage squared. The basic goal of DVS is to quickly adjust a microprocessor’s operating voltage to the minimum performance level required by an application. To scale the voltage by a factor of $\gamma$, there would be a $\gamma^2$ savings in power. More aggressively, the voltage supply can even be gated off. In this case, that part of the system will work in a shut-down mode, which can lose all the old states in it. If those states are still needed in future transactions, some recovery logic is needed to restore the component back to the original state.

The main drawback of voltage scaling is that the consequent increment in transistor propagation delay, which causes the microprocessor to run at a lower speed. As reported in previous research, utilizing supply voltages above $3V$ has little impact
on performance. The speed decreases dramatically as $V_{dd}$ approaches the threshold voltage $V_{th}$ [18].

Voltage scaling can be used to reduce the static power too [27]. To overcome the performance loss of using a lower supply voltage, we can lower the transistor threshold voltage. However, a low threshold voltage can produce increased static power. In order to achieve large potential gains of using a lower voltage supply, the circuits need adaptive controls on both the supply and threshold voltage. Utilizing dual threshold voltages CMOS [89] is one example of employing threshold voltage scaling, and provides an efficient way in CMOS level to reduce the circuit static power.

Another similar approach is the drowsy cache [38], which is a voltage scaling approach on cache lines. During a fixed period of time, the activity in the cache is only centered on a small subset of the cache lines. Such behavior can be exploited to cut the leakage power of large caches by putting the cold cache lines into a state preserving, low-power drowsy mode [41]. The cache with a drowsy mode is called a drowsy cache. Moving lines into and out of the drowsy state incurs a slight performance loss. Drowsy caches are able to reduce the total energy (static and dynamic) consumed in the caches by 50%-75%.
5.2.2 Frequency Scaling

Because the operating voltage is also related to the operating frequency as described in Equation (2.2.3), the voltage scaling and frequency scaling are tightly coupled. Both techniques can be used both on microprocessors and the memory system. Scheduling support from the operation system is needed to dynamically adjust the operating point according to the application workload [66]. The basic concept behind frequency scaling is that when the clock frequency is lower, there could be less power consumed per cycle. However, the microprocessor has to operate for a longer time to execute the same task.

Decreasing the clock frequency causes a linearly proportional decrease in the power dissipation. The power saving from frequency scaling is not as much as the savings obtained from voltage scaling. Therefore, it is not a viable solution for high-performance digital systems, where the clock frequency is the primary design target. Frequency scaling does not change the energy dissipation (power-delay square efficiency metric) for a given task. It can only cause the power consumption over a period of time could be reduced.

Furthermore, for portable battery-operated systems, the total amount of energy provided by the batteries is not a fixed constant. It depends on the rate of battery discharge. There are some advantages in reducing the clock frequency, since the batteries are more efficiently used when the discharge current is small. When the discharge current is decreased, the total amount of energy that can be drawn from
a battery is actually increased. Frequency scaling is typically used in conjunction with other design optimization techniques in order to achieve power savings without significant performance loss.

### 5.2.3 Resources Scaling

Microprocessor resources can be scaled; one such technique is called *resource scaling*. It requires the functional units to be able to work at several energy consumption levels. To reduce power dissipation in the datapath, resource allocations can be dynamically adjusted based on the demands of an application. The sizes of the issue queue (IQ), the reorder buffer (ROB) and the load/store queue (LSQ) can be scaled based on the periodic sampling of their occupancies to achieve significant power savings with minimal impact on performance [69]. This approach can achieve average power savings of 53% with a performance penalty of 5% for SPEC95 benchmarks. A similar approach was done by effectively scaling functional units with different power and latency characteristics and using the critical path information to decrease issue logic power [79].

In the memory system, the memory access time [96] can also be scaled. DRAM technology has matured enough to provide multiple low energy operating modes and refresh rates without losing any memory states. When equipped with compiler and hardware enhancements, utilizing multiple DRAM power modes can achieve a 60% power reduction in DRAM energy [96]. Most scaling techniques follow a similar
philosophy that requires software or hardware to dynamically scale the voltage to achieve improved power utilization while consider the associated performance penalty.

5.3 Banking

Most modern microprocessors employ one or two levels of on-chip caches and large off-chip memory in order to improve performance. Those memory systems often occupy a large portion of chip area and consume a significant amount of power. A lot of work has been done in the circuit design level to minimize the power for each memory cell. At an architectural level, memory could be more power efficient if part of the memory can be turned off or operate in a low power state when it is not being used. It is usually achieved by splitting a large memory into small pieces (called banks), or adding one smaller sized memory in addition to the original memory. Similar techniques have been applied to all levels of the memory hierarchy using different configurations. The side effect of memory subbanking is that it typically introduces extra delay when the power-saving memory bank is turned back on. Software or hardware dynamic algorithms are typically needed to control the memory states.

5.3.1 Vertical Subbanking

Vertical memory subbanking techniques are one type of banking technique. They add an additional layer of memory or a additional cache module to the memory hierarchy. The additional layer can be inserted physically above or below the existing memory
module that is targeted to be optimized.

Vertical subbanking can be used on cache memory systems. Cache power includes power for the address decoder, tag array, comparators, data array, and output drivers. It is very power consuming to complete a cache access. The Filter cache [43] is one successful case for cache subbanking. A small on-chip cache is added above the regular level 1 cache. Cache accesses are first filtered by the filter cache, and then access the lower hierarchy. When an access hits in the filter cache, the lower level cache can remain in standby mode and obtain significant power savings.

5.3.2 Horizontal Subbanking

Unlike vertical memory subbanking that changes the number of layers in the memory hierarchy, horizontal subbanking optimizes power performance at the same memory level. These techniques re-allocate the same level memory structure to be more power efficient.

Scratchpad memory [40] and region-based cache [47] are examples of horizontal memory subbanking. They are small on-chip caches which are located at the same level as the level 1 cache. They can either be controlled by hardware logic or by compiler-generated software algorithms. Some frequently accessed data (e.g., stack and global variables), are stored in scratchpad memory. It provides direct access to commonly used data without involving the high powered cache system, which can save substantial dynamic power. Cool cache [95] is a similar approach to scratchpad
memory. It is specially optimized for multimedia applications.

Memory paging and SRAM banking are also examples of horizontal subbanking. They split big homogeneous memories into small pages. Each page has its own power and clock. If the page is not used for a while, it can be shut off or left running at a low power mode. Memory paging can have some performance penalty, because the power-up process of a page usually takes a long time. Control algorithms need to be designed to optimize the power with minimum performance loss.

5.4 Clustering

Clustering is not originally targeted at power savings, but to address the need for long wires in a design. Clustering is an architectural design technique to partition the processor resources into several groups. Clustering can have a dramatic impact on reducing system power. Each of partition groups is referred to as a cluster. The components of each cluster are simpler and thus less power consuming than those of a unified micro-architecture [1]. There are many ways to partition a microprocessor component. Pipelining and parallelism are two ways of clustering to improve the processor performance in terms of instruction per cycle (IPC or MIPS). Unfortunately, both techniques will inevitably increase the processor dynamic and leakage power due to the extra gates needed to handle pipelining and parallelism. Parallelism would double or duplicate the function units, and pipelining would require additional register latches to store the pipeline states. However, in the case of power efficiency (energy
delay square product), a good design should have much more MIPS gain than the associated power penalty. Appropriate pipelining depth and parallelism width can create a much more power-efficient design. Prior work has shown, both mathematically and experimentally, how to derive the optimal issue width and pipeline depth [30, 107] in term of power efficiency. Optimality can provide microprocessor architects a fundamental design philosophy to employ at the beginning of the design phase. The fidelity of that optimality heavily depends on the mathematical power model that is being used. It has been reported that the optimal logic depth per pipeline is 6-8 FO4 [32]; a section report suggested 18 FO4 [87]. These discrepancies are due to underlying architectural differences.

The performance of clustered microprocessors strongly depends on the ability of the software/hardware to distribute the instructions/operations among clusters in such a way that workload is balanced and inter-cluster communications are minimized. A code generation scheme for clustered micro-architectures through graph partitioning techniques has been shown to be effective to produce microprocessor speedup and reduce power consumption [1].

\subsection{5.5 Switching Reduction}

It is well known that transistor switching is the major source of dynamic power. Once the supply voltage and the system throughput have been fixed for a microprocessor, power savings can be achieved from the minimization of the system switching
activity. Microprocessor activity includes operational and nonoperational activities. Many techniques have been proposed in literature, and despite the fact that those approaches are intrinsically complicated, they can still provide significant power savings.

5.5.1 Logic Re-design

In the power-aware optimization process, designers can re-optimize their design to reduce switching activity. A more power efficient implementation would design the same component with the same functionality but with less switching activities. The key issue is to minimize the number of transitions needed to perform a given task. Typically, new power efficient designs will be more complex than the previous designs. The added logic must consume less power than what it could possibly save. The technique is mainly focused on reducing the power consumed by data storage and communication related to the memory system.

Register files are small on-chip memories on the processor core. Although register files are moderate in size, they are very power hungry, because registers are read/written in every cycle. The charge and discharge of register cells can take up to 16% of the processor power. In Tseng and Asanovic’s paper, seven different techniques, such as the modified storage cell, precise read, bypass skip etc., are presented to reduce the switching activity in the register file [94]. Some new ideas also take advantage of the asymmetric usage of registers (R0, the first register in register file,
be much more frequently used than other registers) and register renaming to decrease the switching activity.

Cache tag lookup and comparisons consume a significant amount of dynamic power in the cache system. A direct addressed cache [103], span cache [102] and soft cache [25] are different hardware-software designs for an energy-efficient microprocessor cache. They allow software to access cached data without a hardware cache tag check, but with the help of a compiler and some pre-knowledge information of the software. These tag-unchecked loads and stores save the power of cache tag check, where the compiler can guarantee one access is located in same line as an earlier access.

In an out-of-order superscaler microprocessor, the instruction execution sequences are not always in contiguous cache location. The non-contiguous fetches can cost the instruction fetching unit more cycles and more power. Trace cache overcomes that limitation by putting instructions contiguous in cache using the trace from dynamic instruction stream [75]. The fill-unit [24] was proposed as one of the trace cache solutions to compact smaller code units into large execution units and store them in decoded instruction cache.

5.5.2 Data Re-encoding

In microprocessor systems, significant power savings can be achieved through the reduction of the data communication activity of the on-chip and off-chip buses. It is
due to the reduction of total capacitance being switched when a voltage change occurs on a bus line. That capacitance charge for off-chip buses is usually much larger than the capacitive load charged or discharged for internal nodes. Encoding paradigms for reducing the switching activity on the bus lines in the processor-to-memory interface have been recently investigated. They encode the binary data before transmit them over the bus, depending on the distinctive characteristics of the data streams to be exchanged.

The Gray code address can achieve minimized switching address bus activity. It is based on the fact that address changes are often sequential and continuous, which Gray code can switch the least number of signals over the bus. A Gray encoder must be placed at the transmitting end of the bus, and a Gray decoder is required at all the receiving end. The trade-off between cost of data encoding/decoding and the power saving on the address buses is investigated in [56].

However, other research shows that better performance than Gray code can be obtained by redundant code [6]. Redundant codes are proposed to add extra signals on the bus in order to reduce the transition. Bus-invert code [88] is one of the redundant codes. Stan and Burleson proposed the use of a redundant encoding scheme to limit the average power. It adds a redundant bus line (INV signal) on the bus, which represents the polarity of the data on the bus. The INV signal is chosen by minimizing the hamming distance between last data on the bus and the current one. If the Hamming distance between two successive patterns is larger than $N/2$ (where
N is the bus width), the current data is transmitted with inverted polarity and the redundant line is asserted; otherwise, the current data is transmitted as it is, and the INV line is de-asserted. If the data transmitted on the bus are independent and uniformly distributed, the average number of transitions per clock cycle is lowered by less than 25% of the original value, due to the binomial distribution of the distance between consecutive patterns. Major drawbacks of this approach are related to the required redundant bus line and the overhead due to the logic to implement the voter to decide whether the Hamming distance exceeds N/2.

A more advanced encoding method for decreasing the address bus activity relies on the locality property of the memory references. The working zone encoding (WZE) scheme [22, 61]. It is suitable for data/ address buses or for shared address buses. The WZE method restores sequentiality by memorizing the reference address of each working zone on the receiver side and by sending only the highly sequential offset. Whenever the memory access moves towards a new working zone, this information is sent to the receiver with a special codeword so the receiver changes the default reference address and the offset transmission can resume. The main limitation of the WZE is its larger encoder and decoder logic overhead, which limits the power benefits of the I/O switching activity reduction and introduces additional delays on signal paths.

Frequent value cache (FVC) [104] uses fewer bits to encode the frequently used value and uses regular bits for other values. The objective of FVC is to reduce the
activities of cache value load and store. It is reported to have 28.8% power reduction in SPEC95.

Code compression has many benefits, not only just for low power. It is also used to decrease the instruction memory, especially for VLIW machine which typically has a very large compiled binary. The smaller instruction size can save power for instruction cache and instruction fetching unit. The new instruction decoding logic will add more power overhead to decompress the instruction, which should not be more than the power saving from the front-end of the pipeline. The code compression can be done by Huffman coding or table hookup. The power saving can be as high as 22% to 82% [49].

5.6 Comparisons of Power Optimization Techniques

In this chapter, five categories of microprocessor architecture level power optimization techniques are presented. We address the power efficiency and five power-aware approaches in architecture-level design and come to the conclusion that pure performance-driven, power-driven or efficiency-driven design could not lead to a desirable design. Although all of the techniques are proposed to save power, the objectives and results of the above five categories are different. Table5.1 lists and compares their impacts on the system performance, power consumption, power efficiency and hardware complexity.

Gating, scaling and memory banking have similar system impacts. They all save
Table 5.1: The power, performance and complexity impacts of different power optimization techniques.

<table>
<thead>
<tr>
<th>Power-Aware Optimization Category</th>
<th>Performance (MIPS)</th>
<th>Power Consumption (W)</th>
<th>Power Efficiency ($MIPS^3/W$)</th>
<th>Hardware Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gating</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\sim$ or $\nearrow$</td>
<td>$\nearrow$</td>
</tr>
<tr>
<td>Scaling</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\sim$ or $\nearrow$</td>
<td>$\nearrow$</td>
</tr>
<tr>
<td>Memory Banking</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\nearrow$</td>
<td>$\nearrow$</td>
</tr>
<tr>
<td>Clustering</td>
<td>$\nearrow$ or $\downarrow$</td>
<td>$\nearrow$</td>
<td>$\nearrow$</td>
<td>$\nearrow$</td>
</tr>
<tr>
<td>Switching Reduction</td>
<td>$\sim$ or $\downarrow$</td>
<td>$\downarrow$</td>
<td>$\nearrow$</td>
<td>$\nearrow$</td>
</tr>
</tbody>
</table>

† Symbol note: $\downarrow$ decreasing; $\nearrow$ increasing; $\sim$ similar;

power by sacrificing some degree of performance. Clustering optimization focuses on power efficiency. The result of clustering increase power consumption and system performance. The performance gain is more dominant than the power consumption increase, and therefore clustering could have overall benefits on the microprocessor power efficiency. Switching reduction techniques introduce power-aware concepts in the design. It could reduces the power consumption and increases the power efficiency at the same time. All of those approaches increase the hardware complexity due to the extra logic introduced to the system. It is important for every power-aware optimization technique to prove that the power for newly introduced logic is smaller than the power saved somewhere else.
Chapter 6

Experimental Methodology

6.1 System-on-a-Chip (SOC) System Architecture

6.1.1 System Bus Hierarchy

Modern system-on-a-chip embedded media systems include many components: a high-speed processor core, hardware accelerators, a rich set of peripherals, direct memory access (DMA), on-chip cache and off-chip memory. The system architecture considered in the study includes a single-core microprocessor, several peripherals, and off-chip SDRAM memory, and is similar to many current embedded platforms. Without losing generality, the system architecture defined can then be used to conduct majority of the experiments.

For multimedia applications, data throughput requirements are increasing higher than what they are ten years ago. Today, for a D1 (720x480 pixel resolution) video codec (encoder/decoder) media node, it needs to be able to process 10 million pixels
per second. This workload requires a multimedia-specialized processor for computation, peripheral devices to support high speed media streaming and data conversion via a parallel peripheral interface (PPI), and a synchronous serial port (SPORT) for interfacing to high speed telecom interfaces. The high data throughput requirements associated with this platform make it impossible to store all the data in an on-chip memory or cache. Therefore, a typical multimedia embedded system usually provides a high-speed system-on-a-chip microprocessor and a very large off-chip memory. The Analog Devices Blackfin family processors [3], the Texas Instrument OMAP [90], and the SigmaDesign EM8400 series [82] are all examples of low-power embedded media chip-sets which share many similarities in system design and bus structure. The system architecture used during the dissertation study is based on those designs and it is shown in Figure 6.1.

Another key component in the architecture model is the system bus and external memory. Memory bandwidth is a great challenge for systems to process streaming data in real-time. To insure sufficient bandwidth, hardware designers usually provide multiple buses in the system, each having different bus speeds and different protocols. An external bus is used to interface to the large off-chip memory system and other asynchronous memory-mapped devices. The external bus has a much longer physical length than other buses, and thus typically has much higher bus capacitance and greater power dissipation. The goal of the architectural model is to accurately model power dissipation in a complete system power model so that new power-efficient design
techniques for the external memory bus can be explored.

### 6.1.2 External Bus Interface Unit

In the system design of Figure 6.1, there are four buses shown. Two buses that are clocked at the frequency of the processor are used to interconnect the processor and caches. There is one internal bus and one external bus that are clocked at a slower frequency. The internal bus and external bus are bridged by an external bus interface unit (EBIU), which provides a glue-less interface to external memory.

There are two sub-modules inside the EBIU, a bus arbitrator and a memory controller. When the units (processor or DMA’s) in the system need to access the external memory, they only need to make a request to the EBIU buffer through the

---

**Figure 6.1: Target embedded multimedia system architecture.**
internal bus. The EBIU read the request and handle the off-chip communication
tasks through the external bus. Because of the potential contention between users
on the bus, arbitration for the external bus interface resources is required. The bus
arbitrator grants requests based on a pre-defined order. Only one access request can
be granted at one time. When a request has been granted, the memory controller
communicate with the off-chip memory directly based on the specific memory type and
protocol. The EBIU can support SDRAM, SRAM, ROM, FIFOs, flash memory and
ASIC/FPGA designs, while the internal units do not need to discriminate between
different memory types. In the experimental model, a multi-banked SDRAM is used
as an example of memory technology and integrate SDRAM state transitions into the
external bus model. The advantage of the modeling framework allows us to consider
different memory technologies, without changing the base system-on-a-chip model.

6.1.3 External Memory (SDRAM) Model

SDRAM is commonly used in cost-sensitive embedded applications that require large
amounts of memory. The SDRAM model used is Micron MT48LC16M16A2 Syn-
chronous DRAM. Figure 6.2 shows a block diagram of the SDRAM interface in the
system architecture. The SDRAM can be organized as multiple banks. Inside each
bank, there are many pages, which can be selected by row address. The size of each
page can be 1 KB, 4 KB or larger. The address inside one page is called the col-
umn address. The external bus is a multi-line bus, which can be grouped into three
Table 6.1: SDRAM Commands Truth Table

<table>
<thead>
<tr>
<th>Command</th>
<th>$SMS$</th>
<th>$SCAS$</th>
<th>$SRAS$</th>
<th>$SWE$</th>
<th>$SCKE$</th>
<th>$SA10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRECHARGE</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>low</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>ACTIVATE</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>READ</td>
<td>low</td>
<td>low</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>WRITE</td>
<td>low</td>
<td>low</td>
<td>high</td>
<td>low</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>REFRESH</td>
<td>low</td>
<td>low</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>low</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
</tbody>
</table>

sub-buses: a control bus (including SCAS, SRAS, SWE, SCKE, SA10, DQM[1:0], BS[1:0]) which carries the SDRAM command signals and bank address, an address bus (including A[12:11], A[9:0]) which multiplexes the row address and the column address, and a data bus (DATA[15:0]) which transmits the loaded or stored data between the SDRAM interface and SDRAM.

The SDRAM operates on a command-by-command basis. Before every access to SDRAM, the EBIU sends one or more commands on the control bus to signal to the SDRAM what the requested data is. The commonly used commands and their associated pin values are listed in Table 6.1. Between each pair of commands, a set amount of delay is required to meet the SDRAM specification. The delay cycles are pre-programmed into the SDRAM. $t_{RAS}$ is the required delay between issuing an ACTIVATE command and a PRECHARGE command. $t_{RP}$ is the required delay after a PRECHARGE command. $t_{RCD}$ is the delay between an ACTIVATE command and a READ/WRITE command. Column Address Strobe (CAS) latency $t_{CAS}$ is the delay from a READ/WRITE command being issued to data ready.
SDRAM has its own context states. There are certain overheads for state transitions. Page miss is one of the state transitions. It occurs when the EBIU executes a PRECHARGE command followed by a bank ACTIVATE command, before executing the READ or WRITE command. There are several cycles of bus delay and that latency is called the page miss penalty. If there is a page hit, the READ or WRITE command can be transmitted immediately without requiring the PRECHARGE and ACTIVATE commands. Figure 6.3 is a timing diagram for processing a read page miss operation. Some latency is required after the PRECHARGE and ACTIVATE commands. For SDRAM READ commands, there is latency from the start of the READ command to the availability of data from the bus. This latency is always present for the first read in the burst and for any single read transfer. Subsequent read bursts do not have any latency, because those operations can be pipelined [3].
When a page miss occurs (about 70% of time for MPEG-2 video encoding and decoding), more power and bus cycle delay are needed to complete the SDRAM access request. Another type of SDRAM state transition occurs when the bus direction is switched (i.e., a READ after WRITE or WRITE after READ.) The bus controller pin needs time to turn around the bus, and is called the *bus turnaround penalty*.

To maximize memory bandwidth, modern SDRAM components allow for pipelining memory commands [74], which eliminates unnecessary stall cycles and NOP commands on the bus. While these features increase the memory bandwidth, they also reduce the bus command power. Consecutive accesses to different rows within one bank have high latency and cannot be pipelined, while consecutive accesses to different rows in different banks can be pipelined. Figure 6.4 shows a timing diagram for processing two read operations in sequential access SDRAM and pipelined access SDRAM. The diagram also shows the cycle savings of pipelined access SDRAM from sequential access SDRAM.
6.2 Bus Power Modeling and Experimental Setup

In the experimental study, the power model of the Analog Devices Blackfin family system-on-a-chip processors is used as primary system model. The code was developed and run on ADSP-BF533 EZ-Kit Lite development board using the VisualDSP++ tool-set. The development board provides a 500 MHz ADSP-BF533 microprocessor, 16 MB SDRAM, and CCIR-656 video I/O interface. Inside the ADSP-BF533 microprocessor, there are both L1 instruction and data caches. The instruction cache is 16 KB 4-way set associative. The data cache is 16 KB 2-way associative. Both caches use a 32 byte cache line size. The SDRAM module selected is Micron MT48LC16M16A2 16 MB SDRAM. The SDRAM interface connects to a 128 Mbit (x8) SDRAM devices to form one 16 MB of external memory. The SDRAM contains 4 banks, with a 1
KB page size. It also has following characteristics to match the on-chip SDRAM controller specification: supply voltage 3.3 V, operating frequency 133MHz, burst length of 1, column address strobe (CAS) latency $t_{\text{CAS}}$ 3 system clock cycles, $t_{\text{RP}}$ and $t_{\text{RCD}}$ 2 system clock cycles, refresh rate programmed at 4095 system clock cycles.

The models described in the previous chapter have been defined to profile the system level communication from the processor to the memory sub-system. The SDRAM memory timing model and power model are applied to the power estimator. The models enable us to simulate many processor-to-memory configurations for a typical system-on-a-chip (SOC) embedded system. It can allow us to model power accurately, and has been validated with physical measurements as described in
Chapter 6. Experimental Methodology

[97]. The experimental ADSP-BF533 EZ-Kit Lite board is shown in Figure 6.5. The model also enables us to evaluate the performance impact of bus traffic arbitration and access pattern based optimization over the external bus traffic. Both optimization algorithms introduced in following two chapters are simulated inside the EBIU simulation module.
Chapter 7

Optimization I: Bus Traffic Arbitration

7.1 Introduction

The bandwidth and latency of external memory system are heavily dependent on the manner in which accesses interact with the three-dimensional SDRAM structure. The bus arbitration unit in the EBIU determines the sequencing of load/store requests to SDRAM, with the goals of reducing contention and maximizing bus performance. The requests from each unit are queued in the EBIU’s wait queue buffer. When a request is not immediately granted, the request enters stall mode. Each request can be represented as a tuple $(t, s, b, l)$, where $t$ is the arrival time, $s$ identifies the request (load or store), $b$ is the address of the block, and $l$ is the extent of the block. The arbitration algorithm schedules requests sitting in the wait queue buffer with a particular performance goal in mind. The algorithm needs to guarantee that bus starvation will not occur.
7.2 Proposed Algorithm

7.2.1 Traditional Algorithms

A number of different arbitration algorithms have been used in microprocessor system bus designs. The simplest algorithm is *First Come First Serve* (FCFS). In this algorithm, requests are granted on the bus based on the order of arrival. This algorithm simply removes contention on the external bus without any optimization and pre-knowledge of the system configuration. Because FCFS schedules the bus naively, the system performs poorly due to instruction and data cache stalls. The priority of cache accesses and DMA access are equal (though cache accesses tend to be more performance critical than DMA accesses). An alternative is to have a *Fixed Priority* scheme where cache accesses are assigned higher priority than DMA accesses. For different DMA accesses, peripheral DMA accesses have higher priority than memory DMA accesses. This differentiation is needed because if a peripheral device access is held off for a long period of time, it could cause the peripheral to lose synchronization or time out. The Fixed Priority scheme selects the request with highest priority in the waiting queue instead of just selecting the oldest. Using Fixed Priority may provide similar external bus performance as the FCFS algorithm, but the overall system performance should be better if the application is dominated by cache accesses. For real-time embedded applications which are dominated by DMA accesses, cache accesses can be tuned such that cache misses are infrequent. Cache fetches can be controlled to occur only at non-critical times using cache pre-fetching and locking.
mechanisms. Therefore, for real-time embedded applications, the FCFS and Fixed Priority schemes produce very similar external bus behavior.

### 7.2.2 Power Aware Algorithms

To achieve efficient external bus performance, FCFS and Fixed Priority are not sufficient. Power and speed are two major factors of bus performance. In previous related work, dynamic external bus arbitration and scheduling decisions were primarily driven by bus performance and memory bandwidth [51, 74]. If a power-efficient arbitration algorithm is aware of the power and cycle costs associated with each bus request in the queue, each request can be scheduled to achieve more balanced power/performance. The optimization target can be to minimize power $P$, minimize delay $D$, or more generally to minimize $P^n D^m$. This problem can be formulated as a shortest Hamiltonian path (SHP) on a properly defined graph. The Hamiltonian path is defined as the path in a weighted directed graph that visits each vertex exactly once, without any cycles. The shortest Hamiltonian path is the Hamiltonian path that has the minimum weight. The problem is NP-complete. In complexity theory, the NP-complete problems are the most difficult problems with non-deterministic polynomial (NP) solving time and they are a class of computational problems for which no efficient solution algorithm has been found. Hence in practice, heuristic methods are used to solve the problem [76].

Let $R_0$ denote the most recently serviced request on the external bus. $R_1$, $R_2$, ...
Chapter 7. Optimization I: Bus Traffic Arbitration

... $R_L$ are the requests in the wait queue. Each request $R_i$ consists of four elements $(t_i, s_i, b_i, l_i)$, representing the arrival time, the access type (load/store), the starting address, and the access length. The bus power and delay are dependent on the current bus state and the following bus state for each request. The current bus state is the state of the bus after the previous bus access has completed. $P(i, j)$ represents the bus power dissipated for request $R_j$, given $R_i$ was the immediate past request. $D(i, j)$ is the time between when request $R_j$ is issued and when $R_j$ is completed, where $R_i$ was the immediate past request. The cost associated with scheduling request $R_j$ after request $R_i$ can be formulated as $P^n(i, j)D^m(i, j)$. A directed graph $G = (V, E)$ can be defined, whose vertices are the requests in the wait queue, with vertex 0 representing the last request completed. The edges of the graph include all pairs $(i, j)$. Each edge is assigned a weight $w(i, j)$, and is equal to the power delay product of processing request $R_j$ after request $R_i$. 

Figure 7.1: Hamiltonian Path Graph
\[ w(i, j) = P^n(i, j)D^m(i, j), \text{ where both } n, m = 0, 1, 2. \] (7.2.1)

The problem of optimal bus arbitration is equivalent to the problem of finding a Hamiltonian path starting from vertex 0 in graph \( G \) with a minimum path traversal weight. Figure 7.1 describes a case when there are 3 requests in the wait queue. One of the Hamiltonian paths is illustrated with a dotted line. The weight of this path is \( w(0, 3) + w(3, 1) + w(1, 2) \). In each iteration, a shortest Hamiltonian path need to be computed to produce the minimum weight path. The first request after \( R_0 \) on that path will be the request selected in next bus cycle. After the next request is completed, a new graph is constructed and a new minimum Hamiltonian path will be found.

Finding the shortest Hamiltonian path has been shown to be NP-complete. To produce a shortest path, heuristics must be used. Whenever the path reaches vertex \( R_i \), the next request \( R_k \) with minimum \( w(i, k) \) is chosen. This is a greedy algorithm, which selects the lowest weight for each step. The bus arbitration algorithm only selects the second vertex on that path. To avoid searching the full Hamiltonian path, the bus arbitration algorithm can simply select a request based on finding the minimum \( w(0, k) \) from request \( R_0 \). The complexity of this heuristic is \( O(N) \), where \( N \) is the number of requests in the queue. When \( w(i, j) = P(i, j) \), arbitration algorithm is to minimize power. When \( w(i, j) = D(i, j) \), then overall delay is minimized. To consider the power efficiency, the power delay product can be used. Selecting different
values for $n$ and $m$ changes the trade-off between power and delay using weight parameter $w(i,j)$.

### 7.3 Experiments

#### 7.3.1 Benchmarks

Experiments are run on a set of multimedia workloads, which is a group of benchmarks that require higher external bandwidths and larger memory footprints.[52]. MPEG-2 and H.264 are chosen for video processing, JPEG for image compression, PGP for cryptography and G.721 for audio processing. All benchmark suites are representative and commonly used applications for multimedia processing. They have unique characteristics, which can be differentiated from other generic benchmarks, such as high data access bandwidths, large memory footprints, etc.

MPEG-2 is the dominant standard for high-quality digital video transmission and DVD. Real-time MPEG-2 encoder and decoder benchmarks are optimized Blackfin MPEG-2 libraries. The input data-sets used are the *cheerleader* for encoding (the size is 720x480 and the format is interlaced video) and *tennis* for decoding (this image is encoded by the MPEG-2 reference encoder, the size is also 720x480, and the format is progressive video). Both inputs are commonly used by the commercial multimedia community.

H264 was developed by the ITU-T Video Coding Experts Group (VCEG) together
Table 7.1: Benchmark Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>MPEG-2 Video encoder with 720x480 4:2:0 input frames.</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>MPEG-2 Video decoder of 720x480 sequence with 4:2:2 CCIR frame output.</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>H.264/MPEG-4 Part 10 (AVC) digital video encoder for achieving very high data compression.</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>H.264/MPEG-4 Part 10 (AVC) video decompression algorithm.</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>JPEG image encoder for 512x512 image.</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>JPEG image decoder for 512x512 image.</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>Pretty Good Privacy encryption and digital signature of text message.</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>Pretty Good Privacy decryption of encrypted message.</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>G.721 Voice Encoder of 16bit input audio samples.</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>G.721 Voice Decoder of encoded bits.</td>
</tr>
</tbody>
</table>
with the ISO/IEC Moving Picture Experts Group (MPEG) as the product of a collective partnership effort. It is also called AVC, for Advanced Video Coding. H.264/AVC is the latest video standard that would be capable of providing good video quality at bit rates that are substantially lower than what previous standards would need (e.g., relative to MPEG-2, H.263, or MPEG-4 Part 2). The implementation complexity is much higher than those previous standards.

JPEG is a standard lossy compression method for full color images. The JPEG encoder and decoder used also employ optimized Blackfin libraries. The input image is Lena (the size is 512x512 in a 4:2:2 color space).

Pretty Good Privacy (PGP) provides encryption and signature for data. The signature used is a 1024 bit cryptographically-strong one-way hash function (MD5) of the message. To encrypt data, PGP uses an asymmetric block-cipher IDEA and RSA for key management and digital signature.

G721 is specified in ITU recommendation G.721. G.721 is a 32kbps ADPCM (Adaptive Differential Pulse Code Modulation) voice compression algorithm. For G.721 encoder, it uses 16bit audio PCM input samples called clinton.pcm, which is included in the MediaBench test suite. For G.721 decoder, the encoded file clinton.g721 was fed into the decoder and output 16bit PCM samples. Audio applications typically have much smaller memory footprint, both instruction and data. G721 is the smallest benchmark amount all the profiled benchmark suites.

In order to measure the external bus power and be able to assess the impact
of the power-efficient bus arbitration algorithm, the following simulation framework is developed. First, we modified the Blackfin instruction-level simulator to include the system bus model and cache activity. From the memory and processor models, all accesses generated on the external bus are fed to an EBIU model. The EBIU model faithfully simulates the external bus behavior, capturing detailed SDRAM state transitions and allows us to consider different bus arbitration schemes. The bus utilization for all benchmarks is compared in Figure 7.2, which is represented in terms of number of external bus requests per one million bus cycles. The average bus power and performance are computed from the simulation results produced by the integrated simulator.
7.3.2 Results

There are eleven different bus arbitration schemes evaluated in the simulation environment. Two traditional schemes are considered for comparison: (1) Fixed Priority (FP) and (2) First Come First Serve (FCFS), along with nine different power-aware schemes. For Fixed Priority, following priority order is assigned (from highest to lowest): instruction cache, data cache, PPI DMA, SPORT DMA, memory DMA.

In the power-aware schemes, each scheme is represented by the pair of power/delay coefficients \((n, m)\) of the arbitration algorithm. \(n\) and \(m\) are the exponents shown in Equation 7.2.1. Different \(n\) and \(m\) values will favor either power or delay. \((1, 0)\) is the minimum power scheme, \((0, 1)\) is the minimum delay scheme, and \((1, 1)\), \((1, 2)\), \((2, 1)\), \((1, 3)\), \((2, 3)\), \((3, 2)\), \((3, 1)\) consider a balance between power and delay by using different optimization weights. Experimental results are presented for both power and delay. The MPEG-2 encoder and decoder simulation results are shown in Figure 7.3, JPEG encoder and decoder are shown in Figure 7.5 and PGP encrypter and decrypter are shown in Figure 7.6. All the experiments consider both sequential command mode SDRAM and pipelined command mode SDRAM.

In all applications, the power dissipation for the power-aware arbitration schemes is much lower when compared to using Fixed Priority or FCFS. The power-aware schemes also benefit from fewer bus delays. These conclusions are consistent across all of the applications studied and are also consistent when using either sequential
Figure 7.3: External bus power/delay results for MPEG-2 video encoder and decoder.
Figure 7.4: External bus power/delay for H.264 Encoder and Decoder.
Figure 7.5: External bus power/delay for JPEG image encoder and decoder.
Figure 7.6: External bus power/delay for PGP encryption and decryption.
Figure 7.7: External bus power/delay for G.721 Voice Encoder and Decoder.
command SDRAM or pipelined command SDRAM. In the MPEG-2 case, the power-aware scheme (1, 0) enjoys an 18% power savings relative to a Fixed Priority scheme for encoder and 17% for decoder. The same power-aware scheme also achieved a 40% reduction in cycles when compared to the Fixed Priority scheme on the MPEG-2 decoder, and a 10% reduction for the MPEG-2 encoder.

To factor out the impact of sequential versus pipelined command mode from the power savings, Table 7.2 shows the bus power savings. Table 7.3 shows the cycle savings. Inspecting the two tables, it can be found that the power-aware arbitration scheme achieves an average power savings of 13% and an average speedup of 22% over all ten applications. There exist some variations in the amount of power savings and speed up achieved. These variations are primarily due to differences in bus utilization across the different applications. For high traffic applications, external memory access requests are more bursty. In those cases, the power-aware schemes provide a larger improvement than in low traffic applications, in which the requests are less bursty. The greater number of requests is in the queue, the greater opportunity the arbitrator can make an improvement from them. Similarly, in Tables 7.4 and 7.5, the pipelined command SDRAM obtains on average a 14% power savings and a 17% performance speedup.

From inspecting the results shown in Tables 7.2-7.5, we can see that the choice of using sequential versus pipelined command modes is not a factor when considering the
Table 7.2: External bus power savings of (1, 0) arbitration vs. fixed priority arbitration in sequential command SDRAM.

<table>
<thead>
<tr>
<th></th>
<th>Fixed priority power (mW)</th>
<th>Arbitration (1, 0) Power (mW)</th>
<th>Power savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>55.85</td>
<td>45.94</td>
<td>18%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>58.47</td>
<td>48.62</td>
<td>17%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>25.02</td>
<td>21.23</td>
<td>15%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>9.82</td>
<td>9.14</td>
<td>7%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>17.64</td>
<td>16.57</td>
<td>6%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>13.77</td>
<td>12.99</td>
<td>6%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>6.33</td>
<td>5.66</td>
<td>11%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>6.81</td>
<td>5.94</td>
<td>13%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>1.95</td>
<td>1.63</td>
<td>16%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>2.56</td>
<td>1.90</td>
<td>26%</td>
</tr>
<tr>
<td>Average savings</td>
<td></td>
<td></td>
<td>13%</td>
</tr>
</tbody>
</table>

Table 7.3: External bus speedup of (1, 0) arbitration vs. fixed priority in sequential command SDRAM.

<table>
<thead>
<tr>
<th></th>
<th>Fixed priority delay (SCLK)</th>
<th>Arbitration (1, 0) Delay (SCLK)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>140.36</td>
<td>126.10</td>
<td>10%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>171.94</td>
<td>101.52</td>
<td>41%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>127.87</td>
<td>68.74</td>
<td>46%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>6.16</td>
<td>5.46</td>
<td>11%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>13.30</td>
<td>10.19</td>
<td>23%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>51.22</td>
<td>36.04</td>
<td>30%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>34.87</td>
<td>25.21</td>
<td>28%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>40.28</td>
<td>35.22</td>
<td>13%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>9.37</td>
<td>8.36</td>
<td>11%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>13.38</td>
<td>11.96</td>
<td>11%</td>
</tr>
<tr>
<td>Average Speedup</td>
<td></td>
<td></td>
<td>22%</td>
</tr>
</tbody>
</table>
Table 7.4: External bus power savings of (1, 0) arbitration vs. fixed priority arbitration in pipelined command SDRAM.

<table>
<thead>
<tr>
<th></th>
<th>Fixed Priority Power (mW)</th>
<th>Arbitration (1, 0) Power (mW)</th>
<th>Power savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>52.51</td>
<td>42.84</td>
<td>18%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>56.29</td>
<td>44.58</td>
<td>21%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>24.07</td>
<td>20.35</td>
<td>15%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>9.60</td>
<td>8.94</td>
<td>7%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>16.97</td>
<td>15.93</td>
<td>6%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>12.41</td>
<td>11.68</td>
<td>6%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>6.05</td>
<td>5.39</td>
<td>11%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>6.40</td>
<td>5.54</td>
<td>13%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>1.89</td>
<td>1.57</td>
<td>17%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>2.44</td>
<td>1.78</td>
<td>27%</td>
</tr>
<tr>
<td><strong>Average Savings</strong></td>
<td></td>
<td></td>
<td><strong>14%</strong></td>
</tr>
</tbody>
</table>

Table 7.5: External bus speedup of (1, 0) arbitration vs. fixed priority in pipelined command SDRAM.

<table>
<thead>
<tr>
<th></th>
<th>Fixed priority delay (SCLK)</th>
<th>Arbitration (1, 0) Delay (SCLK)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>136.73</td>
<td>122.79</td>
<td>10%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>128.82</td>
<td>97.57</td>
<td>24%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>72.71</td>
<td>45.73</td>
<td>37%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>5.34</td>
<td>4.88</td>
<td>9%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>9.50</td>
<td>7.93</td>
<td>16%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>48.02</td>
<td>28.20</td>
<td>41%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>27.61</td>
<td>24.92</td>
<td>10%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>37.34</td>
<td>34.78</td>
<td>7%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>7.88</td>
<td>7.17</td>
<td>9%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>10.98</td>
<td>9.88</td>
<td>10%</td>
</tr>
<tr>
<td><strong>Average Gain</strong></td>
<td></td>
<td></td>
<td><strong>17%</strong></td>
</tr>
</tbody>
</table>
Table 7.6: Power improvements for pipelined vs. sequential command mode SDRAM.

<table>
<thead>
<tr>
<th></th>
<th>Average power in sequential mode (mW)</th>
<th>Average power in pipelined mode (mW)</th>
<th>Power reduction of pipelined commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>47.45</td>
<td>44.40</td>
<td>6%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>50.13</td>
<td>46.64</td>
<td>7%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>21.94</td>
<td>21.19</td>
<td>3%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>9.28</td>
<td>9.09</td>
<td>2%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>16.86</td>
<td>16.26</td>
<td>4%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>13.21</td>
<td>11.89</td>
<td>10%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>5.78</td>
<td>5.50</td>
<td>5%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>6.11</td>
<td>5.70</td>
<td>7%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>1.71</td>
<td>1.64</td>
<td>4%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>2.08</td>
<td>1.95</td>
<td>6%</td>
</tr>
<tr>
<td><strong>Avg. speedup</strong></td>
<td></td>
<td></td>
<td><strong>5%</strong></td>
</tr>
</tbody>
</table>

Table 7.6 and Table 7.7 summarize the results between the sequential command mode and pipelined command mode SDRAMs. The results show that the pipelined command mode SDRAM can produce a 5% power savings and a 14% speedup.
Table 7.7: Speed improvements for pipelined vs. sequential command mode SDRAM.

<table>
<thead>
<tr>
<th></th>
<th>Avg. delay sequential mode (SCLK)</th>
<th>Avg. delay pipelined mode (SCLK)</th>
<th>Speedup of pipelined commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>128.31</td>
<td>124.99</td>
<td>3%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>113.38</td>
<td>103.38</td>
<td>9%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>77.67</td>
<td>48.98</td>
<td>37%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>5.54</td>
<td>4.92</td>
<td>11%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>10.68</td>
<td>8.10</td>
<td>24%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>38.74</td>
<td>31.28</td>
<td>19%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>26.95</td>
<td>25.40</td>
<td>6%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>36.09</td>
<td>35.25</td>
<td>2%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>8.47</td>
<td>7.23</td>
<td>15%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>11.99</td>
<td>9.80</td>
<td>18%</td>
</tr>
</tbody>
</table>

Average speedup: 14%

7.4 Comparison with Exhaustive Algorithm

The input to Hamiltonian path problem is an N-vertex undirected graph, where the vertices are numbered 0 through N. The edges are specified by a symmetric N x N adjacency matrix. An exhaustive search requires starting at every possible vertex \( R_i \), and traversing all the possible paths from the starting point \( R_0 \). The complexity is an \( O(n!) \) problem. A greedy algorithm as shown in Figure 7.1 has a complexity only \( O(n) \). However, greedy algorithm may not always produce the most optimal solutions. Figure 7.8 shows a case that greedy can generate non-optimal solution. The greedy algorithm indicate that request \( R_1 \) should be the next request in the queue, which lead to a total Hamiltonian path cost \( 18 + 18 + 5 = 41 \). The exhaustive
Figure 7.8: A case to compare greedy and exhaustive algorithm

algorithm indicates the next request should be \( R_3 \) instead, because it foresee the possible penalty after the first step. The total Hamiltonian path cost by exhaustive search is \( 20 + 5 + 15 = 40 \), which is lower than the greedy search result.

The performance of both algorithms is compared in table 7.8. The table includes the results of both power and delay. From the results, it is clear that the performance difference between exhaustive algorithm and greedy algorithm are marginal. It is beneficial in term of computation cost from the greedy algorithm which still derives reliable arbitration results.

### 7.5 Implementation Consideration

#### 7.5.1 Power estimation unit (PEU)

From the study of different hardware implementations for power-aware arbitration algorithms, it can be found that the \((1, 0)\) scheme (i.e., the minimum power approach)
Table 7.8: Performance Comparison between Greedy Algorithm and Exhaustive Algorithm

<table>
<thead>
<tr>
<th></th>
<th>Greedy Algorithm</th>
<th>Exhaustive Algorithm</th>
<th>Performance Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Power (mW)</td>
<td>Delay (SCLK)</td>
<td>Power (mW)</td>
</tr>
<tr>
<td>MPEG2-ENC</td>
<td>45.94</td>
<td>126.1</td>
<td>45.76</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>48.62</td>
<td>101.52</td>
<td>48.1</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>21.23</td>
<td>68.74</td>
<td>20.97</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>9.14</td>
<td>5.46</td>
<td>9.05</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>45.94</td>
<td>10.19</td>
<td>45.63</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>12.99</td>
<td>36.04</td>
<td>12.95</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>5.66</td>
<td>25.21</td>
<td>5.64</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>5.94</td>
<td>35.22</td>
<td>5.93</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>1.63</td>
<td>8.36</td>
<td>1.634</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>1.9</td>
<td>11.96</td>
<td>1.92</td>
</tr>
</tbody>
</table>

was actually most favorable with regards to design implementation. The (1, 0) scheme involves several Hamming distance (XOR) computation units and integer adders to form a basic module for power estimation. The design also requires a power estimation unit (PEU) and multi-port integer comparator to select the minimum power request. Each power estimation unit (PEU) takes the current external memory state and address bus pin state as one input source. The other input is the request of interest that resides in the request queue. The memory/bus state includes the last bank address, the open row address (page address) for each bank and the last column address. Each of these states is updated after all memory access commands have been sent out to the external bus. In the case of SDRAM, the last bank address and row address can be used to find out the state information for SDRAM controller.
Therefore, those fields should be shared between the power estimation unit and a SDRAM controller.

There are three steps to complete power estimation. First, if the bank address is not equal to last bank address, a bank miss power number is generated and sent to the accumulator. Second, the bank address is used as an index to inspect the open row address. If the row address is not the same as the next row address, a row miss (page miss) penalty power is sent to the accumulator. The power penalty includes the SDRAM PRECHARGE command power and ACTIVATE command power. If a row address is sent to the bus, the address bus pin state will be changed. The last column address register needs to be updated, because the row address and the column address share the same bus pins. The third step involves comparing the last column address and the next column address to calculate the bus power due to switching in the column addresses. The major hardware component in this third step is the Hamming distance calculator. Finally, all the power is accumulated and output as
an estimated power number. The data flow diagram for the power estimation unit (PEU) is shown in Figure 7.9.

The power estimation unit (PEU) becomes a basic element for the power-aware arbitrator. Figures 7.10 and 7.11 show two possible hardware designs of the (1, 0) power-aware arbitrator. They both contain three major components: 1) a request queue, 2) a minimum power comparator (which include one or more power estimation units), and 3) a memory access command generator (in this case, an SDRAM controller). As memory accesses arrive, they allocate storage space while waiting for service in the request queue. The power estimator computes the latest request’s power relative to the previous request. The comparator selects the minimum power request and stores it. When the currently active request finishes, the access with the lowest power requirements will be sent to the bus by the access command generator. The command generator also updates the memory/bus state information register, which is used to perform request selection in the next cycle.

7.5.2 Two Architectures

The only difference between these two designs is the number of power estimation units (PEUs). Figure 7.10 has only one PEU. Power estimation is performed serially for each request in the queue. All requests share the same power estimator logic. By sharing this structure, hardware complexity is reduced, but this can also introduce some extra delay due to limited computational resources. For instance, if the number
of requests in the request queue is high, and the power estimation unit (PEU) cannot complete all estimations before the previous request finishes, the bus controller would insert idle commands until next request is ready. The bus idle cycles increase the average request access delay. To minimize the probability of encountering idle bus cycles, multiple power estimation units can be used to perform the estimation job. In Figure 7.11, one power estimation unit (PEU) is dedicated to each slot in the request queue so that the power estimation task can be executed in parallel. The estimated power number for all requests in the queue would be computed at the same time. The hardware cost for providing such replication in the estimator design is much higher than the shared architecture, but the hardware replication results in better performance.

To investigate the performance differences between a shared and a dedicated PEU design, both designs were implemented and ran through the MPEG-2 encoder and decoder benchmarks. Figure 7.12 shows the experimental results. As we can see, the average delay can increase for both shared and dedicated architectures when the logic delay for each power estimation unit (PEU) increases. When the PEU logic delay is 5 system clock cycles or higher, the performance degradation for the shared architecture becomes quite unacceptable. It is true in the results for both the MPEG-2 encoder and decoder. Therefore, if the latency of the PEU is less than 5 clock cycles, designers could consider sharing the PEU architecture. If the latency is higher, a dedicated architecture should be considered to avoid severe performance degradation.
Chapter 7. Optimization I: Bus Traffic Arbitration

Figure 7.10: Shared power estimator unit (1, 0) power-aware bus arbitrator architecture.

Figure 7.11: Dedicated power estimator unit (1, 0) power-aware bus arbitrator architecture.

Figure 7.12: Shared and dedicated implementation of MPEG-2 encoder/decoder benchmark performance.
7.6 Summary of Bus Arbitration

The contribution of the proposed technique presented in this chapter is mainly related to the effects of bus arbitration schemes. After a review of the state-of-the-art bus arbitration algorithms, we demonstrate that the new external bus arbitration schemes proposed can balance bus power and delay. The experiments are based on modeling a system-on-a-chip (SOC) embedded multimedia architecture while running six multimedia benchmarks. The power model uses SDRAM as an implementation entity. However, the overall implementation is to preserve the generality of the proposed approach. The experiment results in the chapter show that significant power reductions and performance gains can be achieved using power-aware bus arbitration schemes compared to traditional arbitration schemes. The impact of using both sequential and pipelined SDRAM models is also considered. Finally, two hardware implementations of (1, 0) power-aware arbitrator, shared PEU architecture and dedicated PEU architecture are described and the experiments of those two designs are presented to illustrate the trade-off between design complexity and performance.
Chapter 8

Optimization II: Memory Page Remapping

8.1 Introduction

While designing a low power embedded system, one of the challenges in the design of the architecture is to supply the functional units with data at a sufficient rate while minimizing the associated power consumption. As pointed out in previous chapters, those power is consumed to transmit data across the large capacitance of external system buses. That power consumption is dominant for data hungry embedded multimedia systems.

The multimedia applications discussed in the previous chapter (MPEG-2/H.264 encoding/decoding, JPEG encoding/decoding, speech processing G.721) are characterized by their excessive data memory accesses but with very regular access patterns. Embedded systems, especially multimedia related applications, show even stronger access patterns. Unlike the bus arbitration schemes presented in the last chapter,
the optimization techniques presented in this chapter utilize the embedded application memory access patterns to optimize power and performance of the memory bus subsystems.

It is observed that access pattern aware cache and memory layout techniques can be used to improve the performance of external memory access [48]. However, data access pattern related research is primarily driven by performance improvement. Very little research discusses its effects on power. In this chapter, the data access pattern for the multimedia embedded applications is presented first. Then, a hardware-based, programmable external memory page remapping mechanism is proposed, which can significantly improve performance and decrease the power budget due to external memory bus accesses. The proposed approach was developed by studying common data access patterns present in embedded multimedia applications. An efficient algorithm is developed to map application data into external memory pages, which employs graph-coloring algorithm to guide the page mapping procedure. The objective of the algorithm is to avoid page misses by remapping conflicting pages to different memory banks.

8.2 Data Access Pattern in Embedded Multimedia Applications

Past research has shown that embedded applications exhibit deterministic data access patterns. Since memory delay is a substantial portion of program execution, one
need to effectively exploit this knowledge of data access patterns to tune embedded applications.

In many embedded multimedia applications, execution is dominated by loop bodies that cycle through arrays or matrices of data elements. Loop transformations (e.g., loop unrolling, loop tiling) and array restructuring can improve data access and performance by increasing the locality associated with these memory accesses [15]. Previous work has shown that reordering memory accesses can significantly improve application performance [39, 55].

Loops and arrays are the fundamental program flow and data structures of high performance signal processing applications. The majority of execution time is spent in the loops, accessing data from arrays. In particular, loop and array access patterns inside multimedia applications are at a much higher percentage than in other types of benchmarks. By analyzing the multimedia benchmark hot-spot memory trace, data access patterns can be classified based on the stride and frequencies between successive accesses. Stride is the memory difference between the previous reference and the current reference.

Some common ways to characterize the external memory access patterns associated with multimedia data are stride and inter-reference gap [68]. Brown et al. [11] characterized memory access patterns using a tool that could capture the number of open pages, number of accesses between each reference to the same page, and a number of related statistics. More recently, Lee [48] also studied the performance and
power trade-offs in embedded multimedia applications. He reported that multimedia applications possess three types of memory access patterns. Based on his finding, he used prediction-based pre-fetching techniques to exploit the regularity in access patterns.

Data accesses discussed in this chapter are external memory access after the cache filtering. The memory space traversed by a particular type of algorithm must be large enough to penetrate the cache system. Therefore, some temporary buffers, scratch memory and stack accesses are filtered out by the caches. In a multimedia environment, a typical application includes several large buffers to store video frames, audio samples and a bitstream jitter buffer. The accesses to these data buffers are high in frequency and non-contiguous with a large stride between accesses, which cannot fit into cache memory well. Three dominant patterns are shown in Figure 8.1, Figure 8.2 and Figure 8.3.

8.2.1 Fixed Stride

Fixed stride access is the basic pattern to traverse a linear memory space. There is a fixed stride between each access. The stride can be the same or greater than the access data size. The access timing diagram and pseudo code of an associated algorithm is shown in Figure 8.1. In multimedia applications, the fixed stride memory access pattern is used in memory copy, stream buffer read/write and audio sample I/O.
8.2.2 2-Way Stream

The majority of signal processing algorithms involve filtering operations. The filter functions are one dimensional filtering if they are for speech processing or two dimensional filtering for image/video processing. The characteristics of the filtering process is that the program loads the input samples and coefficient from two different memory locations, executes certain mathematical operations and stores the results back to memory. The access timing diagram and pseudo code of an associated algorithm are shown in Figure 8.2. In multimedia applications, the 2-way stream memory access pattern is commonly used in FIR digital filters, 2D image blending, etc.

8.2.3 2-D Stride

The 2-D stride access pattern is an extension of the 1-D fixed stride pattern. Almost all the double loop programs accessing two dimensional arrays exhibit a similar access pattern. For video and image processing in particular, the frame buffer is a large two dimensional array. Most operations in video and image processing involve a tile
Figure 8.2: 2-way stream data access pattern and pseudo code

Figure 8.3: 2-D stride data access pattern and pseudo code
based data access, which means the data (pixels) may not be linearly adjacent to each other in physical memory, but they are adjacent pixels in a frame with a fixed stride distance. The access timing diagram and pseudo code of an associated algorithm is shown in Figure 8.2. The algorithms using that access pattern are motion estimation, image pre-processing/post-processing, motion compensation and pixel rendering.

8.3 Related work on Data Access Pattern

Optimal memory system design has gained attention recently for embedded systems as a part of design space exploration. Unlike general purpose systems, embedded systems have a higher likelihood of memory optimization because it has been shown that there are only a few memory access patterns for a given set of applications.

Most of the work done in the pattern based optimization is in the instruction space. Compiler researchers have long observed common instruction patterns which can be used for optimized compilation output. The instruction flows are represented in dataflow sub-graphs and considered by the compiler as atomic idioms. The idioms can be reduced into multiple machine operations dependent on the instruction set architecture (ISA) of a target machine. Significant research has been done in the instruction level pattern to optimize the source-level repetition and the iterative nature of integer code. The dynamic frequency of these assembly-level patterns has been found to be quite high. For instance, Spadini et al. have shown that over 25% of dynamic instructions in Spec2000int can be replaced with 10 trivial idioms per
Despite the applicability of idiom extraction to ISA design and code compression, data access patterns have not been investigated thoroughly in the research area. Any algorithm for extracting the most common patterns reduces to sub-graph isomorphism, which is an NP-complete problem [26]. Additionally, this extraction process commonly requires loading the application’s complete dataflow graph (DFG) into memory and performing sub-graph analysis afterward. Both of these requirements make an exhaustive search for common dataflow idioms burdensome, especially on non-trivial applications. This is shown in prior work in instruction pattern analysis.

### 8.3.1 Data Pre-fetching

Other researches have focused on pre-fetching techniques to exploit the stream-like access pattern. They use physical memory address traces to obtain the prediction information for data pre-fetching. Pre-fetching can be triggered by a hardware mechanism, a software instruction, or by a combination of both [5, 36, 105].

The hardware approach detects accesses with regular patterns and speculatively issues pre-fetches at run time, which may cause some run-time overhead and a cache pollution problem in case of misprediction. Some hardware architectures to exploit the regularity of memory accesses have also been proposed for high performance computing [5, 36]. Those proposed hardware architectures pre-fetch data with rather complex prediction logic [105]. These pre-fetching schemes use a reference prediction
table to predict the next access. Each entry of the reference prediction table stores the program counter (PC) of the load instruction, previous effective address, stride, and the state. If there is an entry in which the PC is the same as the look-ahead PC, it pre-fetches the cache line. The look-ahead PC is the next PC expected to be executed. After every memory access it updates the reference prediction table. This scheme can exploit some of the regularity of the memory accesses. But it requires complex hardware logic and pollutes the cache if the prediction fails.

The software approach relies on the compiler to analyze the program and to insert pre-fetching instructions, which may increase the code size. There are also software techniques to pre-fetch both data and instructions [19]. Those pre-fetching approaches are primarily based on prediction from the static analysis of the source code.

8.3.2 Memory Customization

Traditional local memory optimizations have concentrated on increasing hit ratio or reducing transition activity through address coding or program transformations. By customizing the local memory modules according to the prevalent access modes in the application, it is possible to fine-tune the local memory management mechanism for the different types of locality, resulting in a more judicious main memory bandwidth management, and substantial power savings. Grun et al. [29] used different types of local memory modules to store variables with high temporal and high spatial locality, to improve the utilization of the main memory bandwidth, and generate
power reduction. The approach Grun took was a compilation based technique to analyze the data access pattern and to classify cache into two different categories. A temporal cache with a smaller line size and a spatial cache with a bigger line size are introduced because the data access pattern between them is different. The approach used doesn’t require extra complexity and power overhead in the hardware systems.

Instead of using compiler based techniques, reconfigurable cache architectures were proposed recently to improve the cache behavior for general purpose processors, dynamically targeting a large set of applications [98]. Extra control is needed for adaptability and dynamic prediction of the access patterns. Adve et al. [73] also proposed a reconfigurable cache, which allows using the available on-chip SRAM space in CPU-bound applications for different hardware supported optimizations (such as hardware look-up tables for instruction reuse or hardware pre-fetching). Dynamic approaches allow design parameters to change in a defined range toward optimal parameters during the program execution. It gives better “adaptivity” to the executing program, but with a cost of extra hardware.

8.3.3 Data Layout Optimization

Memory access pattern analysis has also been used to guide source code transformations and data layout. The goal of data layout optimization is to reduce the number of cache misses. Previous research looked at how to reduce conflict misses to improve performance and reduce power [17, 77]. De La Luz [50] showed that optimizations
that target reducing off-chip memory energy can generate very different binaries and
power budgets from those that target cache locality only. De La Luz used a memory
system model wherein a subset of memory banks can be controlled individually and
be placed in low power operating mode. Their compiler-based strategy modifies the
original execution order of loop iterations in array-dominated applications to increase
the length of the time period in which memory banks are idle. Their design allows
them to control the operating mode for a given memory bank. They reported that
their scheme can reduce the memory system energy expended by as much as 36.8%
over a strategy that only uses low-power mode. Lebeck et al. [46] described OS-based
strategies for optimizing energy consumption for the banked memory architecture.
The idea is to perform page allocation in an energy-efficient manner by taking into
account the banked structure of the main memory.

Catthoor et al. [17] observed that the source code should be first optimized
to generate the optimal memory access patterns for a given memory architecture.
They proposed several optimization techniques. The data structures could be packed
according to their size and bit-width and allocated into memory modules to minimize
the memory cost. They examined the memory access patterns in the source code to
estimate the memory system performance and apply transformation techniques.

Pettis and Hansen [67] proposed a methodology for remapping the instruction
address space. First, they constructed a graph, where vertices corresponded to basic
blocks. An edge is the weight between two vertices whenever the corresponding code
blocks followed directly in sequential execution. The Pettis and Hansen algorithm then attempts to place procedures that frequently call one another close together in memory in order to improve cache performance. Hashemi and Kaeli [31] proposed a color mapping memory layout algorithm to map program procedures, such that they can be cached more efficiently and took into consideration the procedure size and cache organization. They showed that the instruction cache rate can be reduced by 40% over the original mapping and by 17% over the mapping algorithm of Pettis and Hansen.

8.3.4 Cost Prediction

Byna and et al. [15] found a fast and simple analytical model to predict the memory access cost, classified the data access patterns that appear in the nested loops of scientific applications and applied their model to optimizing the performance of derived data types in a Message Passing Interface (MPI) implementation. However, their model is only applicable to array data accesses, which is not general enough for more complicated applications.

8.4 External Memory Pages Revisited

The main focus of the research is to improve the efficiency of access to multi-banked external synchronous DRAM (SDRAM), which is commonly used in embedded systems. SDRAM memory system can be viewed as a group of banks containing a large
Figure 8.4: Data memory page layout for a memory system of four banks. An ”X” indicates that the corresponding page is in use. A ”*” indicates that the page is open.

The number of pages in a memory system depends on the specific memory manufacturer and configuration. Typically, the size of a page is 1024 bytes (1 KB). A diagram of a typical synchronous DRAM (SDRAM) is shown in Figure 8.4. Inside each bank, only one page can be opened or accessed at a time. The maximum number of open pages is equal to the total number of memory banks.

When the first memory reference is issued to the memory subsystem, there are no memory pages open and the memory controller must issue a full memory address (both row and column addresses) to the selected DRAM bank. Subsequent references to the same page (with the same row address) can be accessed by simply providing the column address since the memory controller keeps the current row address active. This method allows lower access latency to the same page than would be possible if the controller had to provide a full address for each access. Any page which is accessed would remain open until a different page in the same bank is accessed. The currently opened page is forced to close in order to precharge the new row address strobe. This situation is referred to as an external memory page miss.
In a multi-banked memory system, if consecutive page accesses are to the same bank, the memory controller has to issue a PRECHARGE command to close the open page in that bank, followed by a ACTIVATE command to open the new page to be accessed. During the PRECHARGE and ACTIVATE cycles, the memory is only handling state transitions, and there is no actual data being transferred. This transition time is called the page miss penalty. If the next page access is in a different bank, the associated PRECHARGE and ACTIVATE can be overlapped with current on-going data transfers. Therefore, the page miss penalty can be hidden by the current transfer.

Page misses contribute major power and performance issues for SDRAM accesses due to the extra SDRAM command overhead. The page miss ratio is a useful metric to quantify the percentage of memory accesses that involve page miss operations. This ratio is computed as the number of times a page had to be opened divided by the total number of DRAM accesses. The more DRAM banks are in a memory subsystem, the greater number of possible open pages.

Figure 8.5 shows a 2-way stream data access with and without the page miss overhead. It can clearly identify the delay and power penalty imposed by the extra PRECHARGE and ACTIVATE commands, which are issued over the external bus when a page miss occurs. In the example of Figure 8.5, a page miss penalty can introduce an overhead of $t_{RP} + t_{RCD}$ external bus cycles. The external bus power is highly dependent on the memory technology chosen and it is closely proportional
Figure 8.5: SDRAM access signal diagram. Two consecutive data reads are issued to SDRAM. In case (a), two reads are from same memory bank but different pages, which is a page miss situation. In case (b), two reads are from two different banks. If the second page is currently open, PRECHARGE and ACTIVATE commands are not needed.

to the number of bus pins that toggled during each transaction. A page miss can cause more pin toggling due to the extra bus commands, but the size of the penalty depends on the state of the external bus.

The use of open pages in the memory sub-system is governed not only by the memory activity of the application, but also by the refresh interval of the DRAMs used in the memory sub-system as well as the number of DRAM banks used to implement the memory structure. All open pages are closed during a refresh cycle. It is necessary to consider all these factors when modeling a memory sub-system.
8.5 Page Remapping Algorithm

Minimizing the page miss ratio is beneficial to both power and performance. One can reduce that ratio by reordering isolated data accesses into a single page or by using some memory layout strategies that can place sequential, cross-page, accesses possessing similar access patterns into different banks. The first approach has been investigated previously by either using dynamic allocation strategies in the operating system or by using compiler-based strategies. However, those approaches may not achieve the maximum benefit for multimedia applications wherein the dominant activities are within large monolithic data buffers possessing deterministic reference patterns. Those buffers generally contain many memory pages, but are viewed as single entities by a compiler or an operating system, and generally cannot be further optimized. Working at a page level, the hardware can be guided to tune accesses and break through the data buffer boundary. The page level strategy is not limited to data only. It tunes both data and instruction accesses to external memory.

The proposed solution has several advantages over previous work. First, previous solutions required access to the original source code, while the proposed approach does not. Second, previous approaches limited their optimization to complete data entity boundaries (buffers, arrays, etc.) or entire code functions. From the analysis of data access patterns presented in multimedia applications, the majority of the execution time is spent on processing large monolithic data entities in embedded programs. Few of the previous approaches addressed optimizing access within individual data
entities. The proposed solution can break through that boundary and demonstrate the value of considering sub-entity optimization.

8.5.1 Problem Presented

In this section, the mathematical notion is defined to model the SDRAM accesses. From Figure 8.4, an SDRAM can be broken down into $M$ number of independent banks, each bank contains $N$ number of pages. Each page can be represented as $I_{i,j}$, where $i$ is the bank index number and $j$ is the page index inside that bank. For a given benchmark, the profiling tool can generate statistics information of how many times the application has a page transition from page $I_{i,j}$ to page $I_{p,q}$, which called $S(I_{i,j}, I_{p,q})$. Then, the memory cost generated by that benchmark can be quantified as:

$$
\Theta = \sum_{b=0}^{M} \sum_{j=0}^{N} \sum_{q=0}^{N} W_i(I_{b,j}, I_{b,q}) S(I_{b,j}, I_{b,q}) + \sum_{a \neq b=0}^{M} \sum_{j=0}^{N} \sum_{q=0}^{N} W_x(I_{a,j}, I_{b,q}) S(I_{a,j}, I_{b,q})
$$

(8.5.1)

There are two terms inside Equation 8.5.1. The first term accumulates the cost related to the in-bank transitions and the second term accumulates the cross page transition activities. Both terms are calculated by multiplying the benchmark transition frequency with a weight function. $W_i(I_{b,j}, I_{b,q})$ is the in bank transition weight function, which computes the power and delay related to a in-bank page transition reference from page $I_{b,n}$ to page $I_{b,q}$. $W_x(I_{a,j}, I_{b,q})$ is the cross bank transition weight.
function, which computes power and delay associated to a cross bank transition from page $I_{a,j}$ to page $I_{b,q}$. Both $W_i$ and $W_x$ can be either power or delay. More generically, they can also be a product factor of $P^m D^m$ as discussed in previous chapters.

Returning to Figure 8.4, an SDRAM can be broken down into $M$ number of independent banks. Each bank contains $N$ number of pages. Each page can be represented as $I_{i,j}$, where $i$ is the bank index and $j$ is the page index inside that bank.

For a given program, a profiling tool can generate statistics on the frequency of page transitions occurring, where the data access moves from page $I_{i,j}$ to page $I_{p,q}$, which is called $S(I_{i,j}, I_{p,q})$ in the notation.

The objective of optimization is to minimize the cost $\Theta$ in Equation 8.5.1. There are a few scenarios that can simplify the computation load for that equation. When only delay is considered in that equation, $W_i$ and $W_x$ become constant. Both weight factors can be extracted outside the sum operations and the cost $\Theta$ can be grouped summation of benchmark page transition matrix. When only power is considered in the equation, $W_i$ and $W_x$ can be approximated as the hamming distance between two transition page addresses. The cost $\Theta$ could be a weighted group summation. On top of those usage scenarios, the page miss ratio can also be derived from the cost equation as shown in Equation 8.5.2.

$$R_{PageMiss} = \frac{\Theta_{W_i=0, W_x=1}}{\Theta_{W_i=1, W_x=1}}$$ (8.5.2)
8.5.2 Motivation of Page Remapping

Page remapping techniques are widely used in memory management systems of modern computer systems. Modern 32-bit processors support a physical address extension (PAE) mode that allows the hardware to address up to 64 GB of memory. However, external devices (e.g., DMAs) can address only the lower 4GB of memory. One way to solve this problem is to copy the data into the lower 4GB memory space for DMA operations. The overhead associated with this approach is significant because it involves expensive memory copying which can impose significant overhead. To avoid costly memory copies, some computer systems provide programmable hardware support that can be used to remap memory for data transfers using a separate memory management unit (MMU).

Operating system loaders map new pages into memory by either selecting a pool of available pages or allocating memory linearly until memory runs out. If page profiling information can be used to instruct the operating system to remap pages into new physical locations, it can be used to optimize external memory access. It only requires small modifications to the operating system and the linker.

Figure 8.6 shows an embedded system with data caches and I/O DMAs both accessing external memory space. A "PAGE REMAPPING" block is inserted before the external bus access unit (EBIU). The function of the "PAGE REMAPPING" block is to provide a mapping function \( f_{\text{remap}} \) between two page indexes that belong to the same memory page space.
Chapter 8. Optimization II: Memory Page Remapping

By assigning weights \( W_i \) and \( W_j \), a mapping function \( f_{\text{remap}} \) can be found such that the page miss overhead is minimized. The mapping function is represented in the form of a mapping table, which called page remapping table (PMT).

One PMT is provided to the operating system or linker along with each executing program. The operating system loader could load the program into memory pages according to the information present in the PMT and the operating system memory management system can program the remapping block as shown in Figure 8.6. These remapping algorithms are low overhead, particularly since multimedia programs tend to execute for a long period of time and remapping only occurs when the program is first loaded.

8.5.3 Remapping Algorithm

In this section, the remapping algorithms is described in detail to generate the page remapping table (PMT). The basic idea of this algorithm is to treat the external memory as a two-dimensional page array. The pages on the same row in the array share the same row address. The pages in the same column share the same bank. The purpose of the remapping process is to re-allocate a page’s physical location, such that transitions within the same column or bank are minimized. In the proposed algorithm, remapped page is limited to reallocation within the same row. This restriction can
Figure 8.6: Memory page remapping system diagram. The "PAGE REMAPPING" block is the functional block to perform page remapping.
drastically reduce the size of the PMT, because the remapped address is much larger than the size of a single memory bank. It has the added benefit of limiting cache aliasing due to the remapping since it is beyond the address range of typical on-chip caches.

The two dimensional nature of memory page layout allows us to define the algorithm as a graph coloring problem. Each bank can be represented as a distinct color. For each page, it is first assigned a tentative color or bank. The process requires to keep track of the total weight for the current assignment. If a remapped page conflicts with another page, then a new color is assigned to minimize the cost of a conflict. The algorithm constructs a page transition graph. Each node represents a page, and the edge between two nodes is weighted with the number of times there is an immediate page transition between the two pages. The algorithm only focuses on eliminating the first generation conflict between page transitions. A multi-step temporal relationship is the subject of future work.

When mapping a page to physical memory, the algorithm tries to avoid the transition penalties by assigning a color or bank which does not have any conflict or page transitions with previously mapped pages. If a conflict is inevitable, all pages in that row is selected to follow a re-coloring process. By using the re-coloring process, it can be guaranteed that the new mapping would not increase the overall weight by only resolving the first generation conflict in the transition graph.
8.5.4 Page Transition Graph

The transition graph captures page traversal in the program execution. The mapping algorithm starts by building a page transition graph with weighted edges. An edge weight is captured through the profiling processes and represents the number of times such transition occurs. The heavier the weight associated with an edge, the more popular that page transition.

Every memory page in a program binary output is represented by a node in the graph and each edge between nodes represents an access transition across two different pages. The edge value is accumulated through the profiling processes and represents the number of times such a transition happened. The sum of the edge weights entering and exiting the node indicates the number of access transitions from or into that page. The bigger that number, the more popular that particular page is.
Figure 8.7 shows an example of the page transition graph based on the compiler-generated memory layout in the left side of the figure. It shows a 4-bank memory system with 4 rows of memory pages, for a total of 16 memory pages. The program uses 7 out of 16 pages. To illustrate the algorithm, the 7 pages are arbitrarily distributed in memory. Each page is represented by its bank address and page row address. Page $I_{0,0}$ and $I_{0,1}$ are contiguous in physical memory. Page $I_{1,1}$, $I_{1,2}$ and $I_{1,3}$ are contiguous in physical memory. Page $I_{2,1}$ and $I_{3,1}$ are non-contiguous and reside in banks 2 and 3, respectively. Non-contiguous memory layout is commonly encountered, because the compiler can place different data and instructions into different memory sections. These memory sections have predefined sizes in order to accommodate changes in the input data. Some memory is used as DMA buffers. Those memory sections tend to be located in a specific area in memory. On the right side of Figure 8.7 is a page transition graph, which is generated using the page profiling tool. The edge weights capture the page transition frequencies between any two pages during the entire program execution time.

### 8.5.5 Page Coloring

Page address remapping defines a page remapping table (PMT) such that the page transition within a bank is minimized, which means the page miss ratio defined in Equation 8.5.2 is minimized. For example, if a large number of page transition are from $I_{0,2}$ (bank 0, page 2) to $I_{0,3}$ (bank 0, page 3) and $I_{1,3}$ in bank 1 is empty, the
memory system would be more efficient by moving page $I_{0,3}$ to the current physical location of page $I_{1,3}$. The only thing system needed to guarantee the program continues to be functional without any modification is to indicate those changes in the mapping table (PMT).

The job of page address remapping is to define a page remapping table (PMT) such that page transitions within SDRAM memory banks are minimized. The details of the page coloring algorithm is described using an example in Figure 8.7 to demonstrate the procedure. To find an optimal page coloring is a NP complete problem, so heuristics is used to solve the problem.

The first step of the algorithm is to sort the edges based on their transition weight. The weight represents the cost of transitions between two pages. Then, mapping is performed using the sorted set of edge weights and assign the pages connected by the edges to different colors. When the remapping process is started, no pages have been mapped. When processing each edge, the pages associated with the edge are remapped. There are three types of information considered during mapping: 1) whether or not the page has been mapped, 2) the bank number for already assigned pages and 3) the weight parameter array. Every page has one unique weight parameter array, which contains the relationship of that page with every other page, and is organized on a per memory bank basis. It indicates the cost of putting one page into each of the memory banks. For example, if a page has a weight parameter array containing entries 500, 0, 100, 0 respectively, then the cost of mapping the page to
bank 0 is 500 and bank 2 is 100, and there no cost for mapping to the two other banks. The goal of the algorithm is to minimize the final cost. Obviously in this case, the page should be mapped to either bank 1 or bank 3. All bank weight parameters are initialized to 0.

After a page is mapped, the parameter array for the mapped page and the page it connects to need to be updated. For all the edges being processed, there are five possible situations.

1. The first case occurs when neither of the two pages has been mapped yet into the physical memory and they are from different page rows, and there is an open slot in one of those two rows, but not in the same bank. It is the simplest case since they can be assigned different colors and remapped to different bank numbers. The weight parameter array can be updated according to their bank locations.

2. The second case is similar to the first, except that one of the two pages is already mapped, and the unmapped page can be placed in a non-conflicting location.

3. The third case is similar to the first two, except that both pages are already mapped in separate banks. No remapping is needed, and only the weight parameter for those two pages needs to be updated.

4. The fourth case occurs when an edge connecting two pages that have already been processed are mapped to the same row. They are guaranteed to be conflict
free. There is no need to consider those transitions, so no action is needed.

5. The first three cases require that there is empty space in memory to allow pages be placed in different banks. Case five is used to resolve the situation when that requirement does not hold. In this case, two pages can only be placed in the same bank (i.e., a conflict situation). Those pages are still placed in the only available banks. Then, one of the page rows is selected to re-assign the bank number for all the pages on that row in order to resolve the conflict.

Each page has a weight parameter array indicating the cost of mapping it to each potential bank. Those weight parameters do not change and are independent of how pages are mapped. Only when the page is mapped into a bank that has a non-zero weight parameter is that weight then accounted for in the final cost. In case five, there are two options to minimize the conflict cost. One option is to apply an exhaustive search by iterating through all possible permutations of bank assignments and selecting the minimum cost. The complexity of this option is $O(N!)$ ($N$ is the number of banks), and is probably suitable for a small number of banks (2 or 4). The other option is a greedy approach, which only swaps the conflicting page with one of the other mapped page. The complexity of this option is much lower ($O(N)$) and is suitable when a larger number of banks is used (8 or 16). In many cases, conflict free assignments can be produced. However, in some applications, conflicts can only be minimized.

The above process is repeated until all of the edges in the transition graph have
been processed. Finally, after all the pages have been mapped into physical memory, the page re-mapping table (PMT) is generated according to the new page mapping. The PMT is used when a program is loaded on a target platform with page remapping enabled.

### 8.5.6 Example

In order to clearly describe the algorithm, the sample page transition graph in Figure 8.7 is used to demonstrate how to use the page coloring algorithm to remap pages in physical memory. After all the edge weights in the graph are sorted in descending order, edge weights ranging from 500 to 30, and there are 8 distinct edges. The individual mapping step for each of the edges in the page transition graph is discussed from Figure 8.8 to Figure 8.15.

There are eight edges in the graph. The algorithm starts with the heaviest edge in the transition graph, $I_{0,0} \rightarrow I_{0,1}$. Both page $I_{0,0}$ and page $I_{0,1}$ have not been mapped into memory yet, and so they can be placed into the first available empty page slot in the memory as long as they are not in the same bank. $I_{0,0}$ can be mapped into bank 0 and mark it as $I_{0,0}[0]$. $I_{0,1}$ can be mapped into bank 1 and mark it as $I_{0,1}[1]$. The parameter array for both pages is updated in to record the mapping costs associated with all banks. Since $I_{0,0}[0]$ has a transition weight of 500, and $I_{0,1}[1]$ is currently mapped in bank 1, it means that if page $I_{0,0}$ is mapped in bank 1 it would incur 500 page transitions, though it could incur zero cost if it is mapped into
another bank. Therefore the weight parameter array for page $I_{0,0}[0]$ is updated from 
{0, 0, 0, 0} to {0, 500, 0, 0}. Similarly, the weight parameter array for page $I_{0,1}[1]$ is 
set to {500, 0, 0, 0}. Processing edge in step (2) follows the same procedure as the 
first edge. The edge with the second largest weight (200), $I_{1,1} \rightarrow I_{1,2}$ can be mapped. 
Both nodes are mapped into memory and their weight parameters are updated.

When processing the next three edges, the difference between these edges and the 
first two that have been processed are: one of the two pages that the edge connects 
has already been mapped into memory. Only the other unmapped page needs to be 
mapped. In step (3), edge $I_{0,0} \rightarrow I_{3,1}$ is processed. Page $I_{0,0}[0]$ is already mapped 
into bank 0. The only action needed is to find an open page slot in page row 1 and 
not in bank 0, because that would create a conflict with page $I_{0,0}[0]$. In that case, 
page $I_{3,1}[2]$ is mapped into bank 2 since it is available and then the parameter array 
to {0, 500, 0, 0} is updated for page $I_{0,0}[0]$. Because $I_{3,1}[2]$ is in bank 2, its newly 
updated array becomes {0, 500, 100, 0}. For the next two edges, the same process is 
used to map pages $I_{2,1}[3]$ and $I_{1,3}[0]$, respectively.

Next, edge $I_{1,1} \rightarrow I_{3,1}$ connects two pages which are both on page row number 1. 
No matter how remapping is done on that row, there could not be a conflict created 
between them. Therefore, no action is needed and no weight parameters need to be 
updated. This edge is processed using case 4.

The next edge $I_{2,1} \rightarrow I_{1,3}$ connects two pages which have both been previously 
mapped and have been mapped into different banks. No mapping action is needed in
Chapter 8. Optimization II: Memory Page Remapping

this case. The new edge does not affect any mapping, but the information it carries can be still useful and is reflected in the updated weight parameters. This edge is processed using case 3.

For the first seven edges, no conflict is encountered. For every page being mapped into the physical memory, there has been an open page slot available to assign the page to a different bank number (i.e., to a different color). No color conflicts occurred. In my experience, it is generally the case for most multimedia applications. However, there may be situations where no conflict-free mapping can be found. The algorithm is designed to handle those cases as well, otherwise performance would suffer. For the last edge, it is a conflict situation. Edge \( I_{0,0}[0] \rightarrow I_{1,1}[0] \) has both incident pages already mapped into bank 0. This is an example of case 3 above except that the two pages are conflicting. When such a conflict occurs, one of the page rows needs to be remapped in order to resolve the conflict. In practice, the row that has a smaller number of mapped pages is chosen to resolve the conflict. For this example, it could be a lot easier to remap the page on page row number 0 because there is only one mapped page on that row. In order to illustrate the algorithm without making the example too complicated, pages on row number 1 are used for demonstration. Page row number 1 contains four mapped pages \( I_{1,1}[0], I_{0,1}[1], I_{3,1}[2] \) and \( I_{2,1}[3] \), and each of them has a weight parameter array associated with it. In this example, exhaustive search is used to find the page remapping permutation that produces the lowest transition cost. Figure 8.15 shows the results of the exhaustive search. A new mapping scheme for
Figure 8.8: Page coloring step 1. Figure contains memory page layout, the selected mapping edge and weight parameter updates.

Page row number 1 is generated, $I_{2,1}[0]$, $I_{3,1}[1]$, $I_{0,1}[2]$ and $I_{1,1}[3]$. In this example, the conflict has been resolved completely. Finally, the page weight parameter for $I_{0,0}[0]$ needs to be updated accordingly to reflect the location of the newly mapped page $I_{1,1}$.

Since a conflict still exists and page $I_{1,1}[0]$ is mapped in bank 0, the weight parameter for $I_{0,0}[0]$ is $\{30, 500, 100, 0\}$. After the remapping, page $I_{1,1}[3]$ is moved to bank 3, and the weight parameters for $I_{0,0}[0]$ become $\{0, 500, 100, 30\}$

In case 5, conflict resolution is done by using the weight parameters, which does not guarantee that a conflict is resolved in an optimal way. Weight parameters are accumulated metrics of one page to all other pages. The remapping process could affect the weight parameters for all other pages that have non-zero transitions in the other row which is being remapped. The remapping process could make future remapping for those pages potentially less optimal. The only redeeming grace is that conflicts tend to only occur for lightweight edges, and so should have a very low impact on the resulting performance and power.

At this point, all the pages have been mapped back to physical memory. The final mapping is shown from Figure 8.8 to Figure 8.15. Each of the pages need only 2 bits
Figure 8.9: Page coloring step 2. Figure contains memory page layout, the selected mapping edge and weight parameter updates.

Figure 8.10: Page coloring step 3. Figure contains memory page layout, the selected mapping edge and weight parameter updates.

Figure 8.11: Page coloring step 4. Figure contains memory page layout, the selected mapping edge and weight parameter updates.

Figure 8.12: Page coloring step 5. Figure contains memory page layout, the selected mapping edge and weight parameter updates.
Figure 8.13: Page coloring step 6. Figure contains memory page layout, the selected mapping edge and weight parameter updates.

Figure 8.14: Page coloring step 7. Figure contains memory page layout, the selected mapping edge and weight parameter updates.

(for 4 banks) to signal the new bank it is assigned to. The final page re-mapping table (PMT) is shown in Figure 8.16.

### 8.6 Proposed Architecture

The motivation of page remapping is to place consecutive page accesses into different memory banks. As a result, a large number of page misses can be absorbed or eliminated as shown in Figure 8.5, which can increase the memory bandwidth throughput, speedup program execution and decrease energy cost. The external memory can be viewed as a 3 dimensional space. The first dimension is the bank number, and translates into the first several most significant bits (MSBs) of the external memory address. It indicates to the memory controller which bank the current address is trying to access. The width of the bank address depends on the number of banks in
Figure 8.15: Page coloring step 8. Figure contains conflict resolving and final memory mapping.

Figure 8.16: Page remapping table (PMT) for example. "00", "01", "10" and "11" are two bits information and "XX" indicate undetermined values.
the memory system.

The second dimension is the row address, and makes up the middle bits of the external memory address. It indicates to the memory controller which page inside that bank should be accessed. The number of bits required depends on the page and memory size. The last dimension is the column address. It is the address inside the accessed page. Typically, modern SDRAMs have a 1 KB page size. Some manufacturers still support 512 bytes or even smaller. For example, if a 16 MB 4-bank SDRAM is used, the total number of bits needed to address that memory is 24 bits. Out of those 24 bits, the 2 MSBs are the bank address, the next 12 bits are the row address and the last 10 LSBs are the column address.

From the external memory address structure, we realize that by only remapping the bank address of every memory access, contiguous page accesses can be effectively distributed into different memory banks. The implementation of remapping blocks is tightly coupled with system memory architecture. Today there are mainly two types of embedded memory architectures: 1) systems without a virtual memory management unit (non-MMU) and 2) systems with a virtual memory management unit (MMU). The strategy that performs remapping in both types of systems is significantly different.
8.6.1 Implementation Issues

In a non-MMU memory system, there is no virtual memory translation hardware. All of the memory references from caches or DMA are direct accesses. The memory space is linear and flat. The "PAGE REMAPPING" logic has to be designed with the lowest implementation cost.

The function of page remapping is done inside the "PAGE REMAPPING" block, and the whole process is transparent to the original software program. The hardware architecture to perform the bank address remapping is done through a lookup table (LUT). Figure 8.17 shows an example of a page remapping block for a 16 MByte 4-bank SDRAM system. For every external memory address, the first 14 bits are extracted to be the index into the page remapping table (PMT). The entry inside
the PMT is the new bank address (2 bits) for that particular page. For the given memory size and configuration, the size of the PMT is only 4 KB. If a 2-bank SDRAM is used, only 1 bit is needed for each page and for an 8-bank SDRAM memory, 3 bits are needed.

For a system that has a MMU memory system, the memory system itself already possesses an address translation mechanism. There is no need to insert more hardware into the system, since the existing memory unit can be used. In those systems, the page remapping table (PMT) becomes a page allocation requirement for an operating system loader. Before a program can be executed, the operating system has to allocate some number of open page slots (frames) in memory for the new program. Those page slots are allocated from a pool of pages which is managed by the operating system. Instead of treating all the pages the same (as the operating system usually does), the remapping algorithm suggests \( N \) number of different page pools, where \( N \) is the number of memory banks of physical memory. Each pool contains the open pages from one memory bank. The operating system needs to allocate open pages from different memory pools according to the entry in page remapping table (PMT). For example, if the entry in PMT for page \( I_{0,0} \) is "10", it means that the operating system should allocate one open memory page for \( I_{0,0} \) of a program from page pool of bank number 2("10" = 2). It could provide the program with better memory efficiency.
Table 8.1: Benchmarks used in the experiments and their important characteristics.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total Cycle</th>
<th>Utilization Ratio</th>
<th>Request Per Cycle</th>
<th>Instruction Size</th>
<th>Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>82450027</td>
<td>0.377</td>
<td>0.0335</td>
<td>40 KB</td>
<td>4.5 MB</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>100000006</td>
<td>0.423</td>
<td>0.0348</td>
<td>41 KB</td>
<td>4.0 MB</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>55621679</td>
<td>0.198</td>
<td>0.0162</td>
<td>63 KB</td>
<td>4.6 MB</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>36106904</td>
<td>0.110</td>
<td>0.0150</td>
<td>32 KB</td>
<td>4.0 MB</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>15410883</td>
<td>0.108</td>
<td>0.0079</td>
<td>19 KB</td>
<td>1.8 MB</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>7666674</td>
<td>0.160</td>
<td>0.0110</td>
<td>18 KB</td>
<td>1.2 MB</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>462588015</td>
<td>0.002</td>
<td>0.0001</td>
<td>11 KB</td>
<td>0.6 MB</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>436908402</td>
<td>0.002</td>
<td>0.0001</td>
<td>11 KB</td>
<td>0.6 MB</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>142895424</td>
<td>0.017</td>
<td>0.0015</td>
<td>16 KB</td>
<td>1.3 MB</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>93717428</td>
<td>0.008</td>
<td>0.0023</td>
<td>16 KB</td>
<td>2.1 MB</td>
</tr>
</tbody>
</table>

8.7 Experiments

8.7.1 Methodology

In the experiments, the same system-on-a-chip processor model is used in previous chapters and used for experiments of bus arbitration schemes. It is based on the Analog Devices Blackfin family microprocessor and all the benchmarks were compiled and simulated by Analog Devices commercial tool chain. The timing/power model developed in previous chapters and previous research work [64] has been calibrated against actual hardware. It can produce profiles of external memory bus activity and allows me to simulate many processor-to-memory configurations accurately. The simulator is extended to generate cycle accurate benchmark trace. A new module is added to simulate page remapping unit.

First pass of simulation, the page transition graph is generated using a program
trace profiler with page remapping disabled. The graph is fed into the page coloring algorithm to compute the optimal page remapping table (PMT). The generated page remapping table (PMT) is then reload back to the modified cycle accurate simulator with page remapping enabled. The page remapping module loads the page remapping table (PMT) and filters all the address traffic going through the external bus. Both power and delay statistics are monitored. Cache models are also built into the simulator to accurately simulate the benchmark behavior, as the remapped external memory space could cause different cache behaviors. The results presented are based on the ADSP-BF533 processor model presented in previous chapter.

The goal of this study is to compare the power and performance benefits offered by the page remapping scheme. In the environment, an external SDRAM memory with 2, 4 and 8 memory banks with a 1 KB page size is simulated. The SDRAM model is modeled after a Micron 16 MB SDRAM. The detailed SDRAM configuration is described in more detail in an Analog Devices engineering note [4]. In the color mapping model experiments, the number of colors is equal to 2, 4 and 8, which is equal to the number of memory banks.

Table 8.1 describes the key attributes of the benchmarks. Experiments were run on the same set of multimedia workloads used in previous chapter, which are representative and commonly used applications for multimedia processing. In Table 8.1, utilization factor indicates how much load each benchmark places on external memory. It is calculated as the percentage of execution time when external bus is busy. The
embedded benchmarks used in experiments do not contain any dynamically allocated memory. All the data buffers present are statically mapped into physical memory. From the data in Table 8.1, we can see that the video benchmarks (MPEG-2 and H.264) tend to place more pressure on memory than the other benchmarks. PGP places the lightest load among all benchmark studied. The workload and memory sizes could affect the results of page remapping algorithms.

8.7.2 Results

Miss Rate

Table 8.3 compares the page miss rates for the original program and the page remapped program. It reports the percentage improvement due to page remapping in SDRAM. The results show that by incorporating the page remapping module, the system can achieve on average a 40.2% improvement on a 2-bank SDRAM system, 69.7% on 4-bank SDRAM system and 81.4% on an 8-bank SDRAM system.

Video benchmarks (MPEG-2 and H.264) and image processing benchmark (JPEG) are the primary targets for performance optimization due to their high memory demand and high power consumption. Table 8.3 shows that for video and image programs, the effect of page remapping steadily increases as the number of banks increases. For a 4-bank SDRAM, the page remapping can eliminate almost half of the page misses. For PGP encoding and decoding, page remapping can remove almost all of the page misses. The reason is that the PGP benchmark has a relatively small
Table 8.2: Miss rate comparison of all multimedia benchmarks. "Original" and "Remap" columns are the page miss ratio without and with page miss module.

<table>
<thead>
<tr>
<th></th>
<th>2 Bank DRAM</th>
<th>4 Bank DRAM</th>
<th>8 Bank DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Remap</td>
<td>Original</td>
</tr>
<tr>
<td>MPEG2-ENC</td>
<td>31.50%</td>
<td>28.90%</td>
<td>29.90%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>41.70%</td>
<td>29.10%</td>
<td>37.80%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>37.10%</td>
<td>31.40%</td>
<td>34.90%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>18.40%</td>
<td>9.40%</td>
<td>15.40%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>46.80%</td>
<td>22.00%</td>
<td>47.30%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>62.60%</td>
<td>45.10%</td>
<td>56.30%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>62.40%</td>
<td>18.70%</td>
<td>62.30%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>62.10%</td>
<td>23.40%</td>
<td>61.40%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>10.40%</td>
<td>5.90%</td>
<td>10.00%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>10.70%</td>
<td>6.10%</td>
<td>10.20%</td>
</tr>
</tbody>
</table>

footprint, so page remapping can easily distribute the conflicting pages into different banks and resolve all the conflicts.

**Power and Delay**

To examine the impact of page remapping optimization on the performance of these benchmark programs, Figures 8.22 and 8.23 show the average delay and power for the original and page-remapped programs. For these results, the average delay is the number of external bus cycles to fulfill one external memory read or write request. The average power reported includes the external bus and memory power associated with the benchmark. Figure 8.22 indicates consistent results that the page remapping module can significantly reduce delay due to external bus requests. We can see a 17% speedup for MPEG2-ENC and a 20% speedup for MPEG2-DEC in a 4-bank SDRAM system. In an 8-bank memory system, the improvement is even higher. The speed
Table 8.3: Benchmark page miss reduction rate of 2-bank, 4-bank and 8-bank SDRAM memory systems.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>2-BANK</th>
<th>4-BANK</th>
<th>8-BANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG2-ENC</td>
<td>8.3%</td>
<td>46.2%</td>
<td>66.5%</td>
</tr>
<tr>
<td>MPEG2-DEC</td>
<td>30.2%</td>
<td>54.5%</td>
<td>69.1%</td>
</tr>
<tr>
<td>H264-ENC</td>
<td>15.4%</td>
<td>60.7%</td>
<td>83.2%</td>
</tr>
<tr>
<td>H264-DEC</td>
<td>48.9%</td>
<td>79.9%</td>
<td>93.5%</td>
</tr>
<tr>
<td>JPEG-ENC</td>
<td>53.0%</td>
<td>76.3%</td>
<td>83.7%</td>
</tr>
<tr>
<td>JPEG-DEC</td>
<td>28.0%</td>
<td>48.1%</td>
<td>68.1%</td>
</tr>
<tr>
<td>PGP-ENC</td>
<td>70.0%</td>
<td>99.8%</td>
<td>100.0%</td>
</tr>
<tr>
<td>PGP-DEC</td>
<td>62.3%</td>
<td>93.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>G721-ENC</td>
<td>43.3%</td>
<td>72.0%</td>
<td>75.0%</td>
</tr>
<tr>
<td>G721-DEC</td>
<td>43.0%</td>
<td>66.7%</td>
<td>74.5%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>40.2%</td>
<td>69.7%</td>
<td>81.4%</td>
</tr>
</tbody>
</table>
Figure 8.19: External bus average power and access delay of multimedia benchmarks in 2-bank external memory system with or without page remapping.

Figure 8.20: External bus average power and access delay of multimedia benchmarks in 4-bank external memory system with or without page remapping.
Figure 8.21: External bus average power and access delay of multimedia benchmarks in 8-bank external memory system with or without page remapping.

Improvements are direct results of page miss rate reduction. The more banks in a memory system, the more improvement the page remapping algorithm can produce. External bus power can be reduced if the signal toggling rate on the external bus decreases [64]. Figure 8.23 shows a mixture of results in terms of power savings. In the MPEG-2 benchmarks, the average power benefits are small. For other benchmarks, such as H.264 and JPEG, page remapped programs show significant power savings.

Reducing the number of page misses reduces the number of external bus commands and stall cycles, but may not lead to lower signal toggling per second. However, due to the reduced number of bus commands and stall cycles, the benchmark program can execute much faster, which could inevitably lead to lower energy consumption. Other benchmarks like H.264 and JPEG demonstrate both power and energy improvements. Their power improvements range from 4% to 30%.
Figure 8.22: Average request delay in system bus clock cycle for multimedia benchmarks with 2-bank, 4-bank and 8-bank memory systems. Both original and page remapped programs are simulated.

Figure 8.23: Average external bus and memory power for multimedia benchmarks with 2-bank, 4-bank and 8-bank memory systems. Both original and page remapped programs are simulated.
8.7.3 Stability

To investigate the stability of the page remapping table (PMT) for different program inputs, two raw pixel sequences are used to drive the H.264 encoder. Both input sequences, CL and FB, have an image size of 720x480 (typically new data may arrive, but the image size is not changed for a single application). The PMT table was generated for the CL sequence. Table 8.4 shows that the performance of FB is very close to the CL image. This is because multimedia programs possess a somewhat fixed data access pattern that is similar across different program inputs. Table 8.5 shows the performance of H.264 decoder benchmark with two different benchmark inputs. The results again are very close. H.264 decoder is also tested for two different sets of inputs, tennis and carousel, as shown in Table 8.5. The PMT table was generated from tennis. Using the same PMT table, the performance of carousel is very close to tennis. It provides some evidence regarding the stability and resilience of the proposed approach.

8.8 Summary of Memory Page Remapping

External memory system performance is critical for embedded applications. Previous approaches have focused on either compile-time or operating system implementations. The proposed algorithm takes into account the page access pattern of embedded multimedia applications. An efficient coloring algorithm is proposed to intelligently map pages into the external memory in order to avoid page misses. The experimental
Table 8.4: H.264 encoder benchmark results with two different input data sets: (i) CL and (ii) FB. A profile of CL input data set was used to construct the PMT in both cases.

<table>
<thead>
<tr>
<th>SDRAM Banks</th>
<th>Experiment Results</th>
<th>CL</th>
<th>FB</th>
<th>Improve</th>
<th>CL</th>
<th>FB</th>
<th>Improve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Remap</td>
<td>Improve</td>
<td>Original</td>
<td>Remap</td>
<td>Improve</td>
<td></td>
</tr>
<tr>
<td>2-BANK</td>
<td>Miss Rate (%)</td>
<td>37.1%</td>
<td>31.4%</td>
<td>15.4%</td>
<td>36.2%</td>
<td>31.6%</td>
<td>12.7%</td>
</tr>
<tr>
<td></td>
<td>Delay (cycle)</td>
<td>59.9</td>
<td>59.1</td>
<td>1.3%</td>
<td>61.2</td>
<td>60.3</td>
<td>1.5%</td>
</tr>
<tr>
<td></td>
<td>Power (mW)</td>
<td>22.4</td>
<td>21.5</td>
<td>4.0%</td>
<td>28.1</td>
<td>27.1</td>
<td>3.6%</td>
</tr>
<tr>
<td>4-BANK</td>
<td>Miss Rate (%)</td>
<td>34.9%</td>
<td>13.7%</td>
<td>60.7%</td>
<td>34.1%</td>
<td>14.1%</td>
<td>58.7%</td>
</tr>
<tr>
<td></td>
<td>Delay (cycle)</td>
<td>59.1</td>
<td>52.4</td>
<td>11.4%</td>
<td>60.3</td>
<td>53.7</td>
<td>11.0%</td>
</tr>
<tr>
<td></td>
<td>Power (mW)</td>
<td>21.7</td>
<td>19.4</td>
<td>10.7%</td>
<td>27.2</td>
<td>24.3</td>
<td>10.8%</td>
</tr>
<tr>
<td>8-BANK</td>
<td>Miss Rate (%)</td>
<td>29.8%</td>
<td>5.0%</td>
<td>83.2%</td>
<td>29.7%</td>
<td>5.7%</td>
<td>80.8%</td>
</tr>
<tr>
<td></td>
<td>Delay (cycle)</td>
<td>58.3</td>
<td>48.5</td>
<td>16.8%</td>
<td>59.4</td>
<td>50.1</td>
<td>15.6%</td>
</tr>
<tr>
<td></td>
<td>Power (mW)</td>
<td>20.1</td>
<td>17.7</td>
<td>11.9%</td>
<td>25.3</td>
<td>22.3</td>
<td>11.7%</td>
</tr>
</tbody>
</table>
Table 8.5: H.264 decoder benchmark results with two different input data sets: (i) tennis and (ii) carousel. A profile of tennis input data set was used to construct the PMT in both cases.

<table>
<thead>
<tr>
<th>SDRAM Banks</th>
<th>Experiment Results</th>
<th>tennis</th>
<th>carousel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Remap</td>
<td>Improve</td>
</tr>
<tr>
<td>8-BANK</td>
<td>Miss Rate (%)</td>
<td>15.40%</td>
<td>1.00%</td>
</tr>
<tr>
<td></td>
<td>Delay (cycle)</td>
<td>58.98</td>
<td>54.22</td>
</tr>
<tr>
<td></td>
<td>Power (mW)</td>
<td>8.7</td>
<td>8.0</td>
</tr>
<tr>
<td>4-BANK</td>
<td>Miss Rate (%)</td>
<td>15.40%</td>
<td>3.10%</td>
</tr>
<tr>
<td></td>
<td>Delay (cycle)</td>
<td>58.97</td>
<td>55.83</td>
</tr>
<tr>
<td></td>
<td>Power (mW)</td>
<td>9.2</td>
<td>8.5</td>
</tr>
<tr>
<td>2-BANK</td>
<td>Miss Rate (%)</td>
<td>18.40%</td>
<td>9.40%</td>
</tr>
<tr>
<td></td>
<td>Delay (cycle)</td>
<td>61.19</td>
<td>58.98</td>
</tr>
<tr>
<td></td>
<td>Power (mW)</td>
<td>10.0</td>
<td>9.1</td>
</tr>
</tbody>
</table>
results demonstrate the benefits of the page remapping algorithm. It can reduce the memory page miss rate by 70-80% on average over the original program mapping. The effect of reduced page miss rate translates to increased performance and reduced power consumption.
Chapter 9

Summary and Future Work

9.1 Contributions

System-on-a-chip (SOC) microprocessor system design continues to make progress in terms of performance and power. With the increasing complexity in embedded systems and requirements of shorter time-to-market of embedded products, it adds tremendous design challenges to meet the performance and power targets and maintain the design complexity to a controllable scale. In this dissertation the main focus is on developing architectural level estimation techniques to estimate system-level power in high confidence and optimization techniques to achieve higher power efficiency. The key contributions of the dissertation are summarized in the following sections.

9.1.1 System Bus Power Estimation

The power consumption associated with the system-level bus represents one of the major contributions to the overall power budget. In the dissertation, the background
and research status on the external bus power in a system-on-a-chip (SOC) microprocessor system is reviewed. Then, the methodology to estimate dynamic and status power consumption on the system-level bus and memory is defined. Finally, the design trade-offs of power-aware system design in terms of power, performance and complexity are summarized.

The targeted applications used in the dissertation are multimedia benchmarks, because they have much higher data access frequency and much higher power consumption than other type of benchmarks. To evaluate their power and performance, a simulation framework is developed to achieve accurate and efficient estimation of the power consumption associated with the system-on-a-chip (SOC) systems. Analog Devices Blackfin processor Blackfin EZ-Kit is used as the primary testbench to generate all the metrics used in the model. The power model also uses SDRAM as an implementation entity. The model is generic enough to be suitable for many memory systems of different applications. Part of related work was published in Workshop on Power-Aware Computer Systems (PACS) in conjunction with Micro-37[63].

### 9.1.2 Bus Arbitration

Power aware bus arbitration is the first approach proposed to optimize system-level bus performance. An arbitration algorithm is designed for the external bus interface unit (EBIU) to reschedule external bus requests based on the criteria of minimum
power-delay product. The results show that significant power reductions and performance gains can be achieved using power-aware bus arbitration schemes compared to traditional arbitration schemes, first-come-first-serve (FCFS) and fixed-priority. Two hardware implementations are provided and the trade-offs between them are also explored. The publication of bus arbitration work was accepted in Proceedings of International Conference on High Performance Embedded Architectures and Compilers (HiPEAC) 2005[64] and later was published in the Transaction of HiPEAC.

9.1.3 Memory Page Remapping

External memory page misses are the overhead of system level bus accesses. Page misses can directly lower the bus bandwidth and increase memory power consumption. Based on the founding of a variety of access patterns in multimedia applications, such as fixed stride accesses, 2-way stream accesses and 2-D stride accesses, we proposed a page remapping algorithm that can benefit external memory accesses. A page coloring algorithm is designed to optimize the external memory data layout and the benchmark page miss ratio is significantly reduced based on the simulation results. Due to the reduced overhead in the memory access, the multimedia benchmarks studied in the dissertation demonstrate increased memory bandwidth and reduced power consumption. The article about the memory page remapping algorithm was accepted in Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES’07)[65].
9.2 Future Work

During the dissertation study, our focus is on system-level bus and memory behaviors in multimedia benchmarks. Such work can be extended to multi-core embedded architectures and can be used to develop techniques for memory and bus management in shared memory space. There are areas where one could focus on developing algorithms to consider data and code memory accesses separately and could utilize access patterns in shared or dedicated memory spaces. Several other aspects of external memory such as multiple bank power states, page refresh rates and heterogeneous memory structure can be further exploited.
Bibliography


[32] M. S. Hrishikesh, D. Burger, N. P. Jouppi, S. W. Keckler, K. I. Farkas, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter


