MASTER

Application-driven energy-aware memory partitioning for ultra low power DSPs

Walimbe, S.N.

Award date:
2012

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
Application-driven Energy-aware Memory Partitioning for Ultra Low Power DSPs

Master Thesis Report

Tutor: Dr. Christian A. Bachmann (Holst Centre/imec-NL)

Supervisor: Prof. Dr. H. Corporaal (Electronic Systems Group, Eindhoven University of Technology)

Committee Member: Prof. Dr. C.H. van Berkel (Eindhoven University of Technology)
Lord Kṛṣṇa said

"Your right is only to perform your duty, but never to its results (fruits). Let not the results be your motive, nor you be indolent."

Śrīmad Bhagavad-Gītā, Chapter Two, Verse Forty-Seven
Abstract

Memory subsystems account for a substantially large fraction of the energy consumption of any embedded system. Especially the systems which work on a battery suffer from this in the race to reduce the energy consumption of the whole system. Partitioning the memory according to the software application requirements, has been a popular method to reduce the power consumption in SRAM based memories in these systems. Most of the partitioning techniques presented so far in the literature concentrate on the dynamic energy consumption in memory.

In recent years, the advanced CMOS technology nodes have attracted the attention of the researchers towards the static energy consumption of the SoC. The time duration for which a memory is idle, can be exploited to reduce the static energy consumption in the memory subsystem. By making use of the concept of Low Power Operating Modes (LPOMs), this thesis presents partitioning based methods to reduce the energy consumption of a memory, from both dynamic and static energy viewpoint. To achieve this, it makes use of the memory partitioning with efficient data placement in each of the partitions. The presented problem is a multivariate multidimensional combinatorial optimization problem with large non-convex search space. This work presents two heuristic approaches to reach an energy efficient solution within an acceptable amount of time. As will be evident from the results, careful analysis of the software application and accurate modelling of the memories can help build an automated framework to compute a memory configuration tailored to a specific application. Energy savings close to 25% of the original memory architecture were achieved for the test application considered in this work.

Keywords: Power optimization, leakage power, embedded design, memory hierarchy, scratch-pad memory, partitioning algorithm, memory management, simulated annealing, set theory, SRAM chips
Acknowledgements

Throughout the time spent working on this thesis, I interacted with many people. I feel extremely fortunate that each and every one of them was so kind to interact with me to solve all the questions I had.

A huge thank you to,

Chris, for being more friend than supervisor. Along with the the technical discussions, I have learned a lot from you during this time period. I would not have been motivated to perform any of the work presented in this thesis if not for your constant encouragement and support. You made me aim high, but at the same time helped me keep my feet on the ground.

Prof. Corporaal, for having those long and fruitful discussions with me, even if it used to get late in the evening. Your expertise and the approach of evaluating from all the aspects has helped me a lot.

Mario, for all the help you provided by going out of your way. Special thanks for the test platforms which you created, by taking out the important time from your schedule. Your knowledge, patience and kindness is truly commendable.

imec-NL DSP team: Alex, Ben, Jan, Jos Huisken, Jos Hulzink, Maryam, Tobias and Vibhu for healthy discussions, help and for providing me with the best possible work environment.

Prof. Francky Catthoor, for insightful discussions which helped me to be on track.

Prof. Kees Van Berkel, for taking out time to review my work.

Prof. Hans Zantema and Sander Stuijk, for extremely helpful discussions regarding computer assisted optimization problem solving techniques.

Also thanks to many more friends from imec-NL and TU/e who have helped me directly or indirectly. You know who you are.

Above all, I owe everything I have to my parents. I have been able to sail through till now, only because of their support and the values they imbibed in me. This thesis is dedicated to them.
## CONTENTS

4.1.2 Definitions of Concepts Used in Formulation ................................................. 22
4.2 Problem Formulation ......................................................................................... 24
4.3 Analysis of Search Space .................................................................................. 27
4.4 Data Object Clustering based Heuristic Algorithm ............................................. 28
   4.4.1 Step 1 - Data Object Clustering ................................................................. 29
   4.4.2 Step 2 - Search for Efficient Partitioning .................................................. 30
4.5 Simulated Annealing based Partitioning Algorithm ............................................. 31

5 Implementation, Results and Evaluation ................................................................. 33
   5.1 Hardware Analysis of Test System .................................................................. 33
   5.2 UWB Frame Decoding ..................................................................................... 34
   5.3 Software Analysis of Test System .................................................................... 34
   5.4 Developed Tools and Models .......................................................................... 37
      5.4.1 Memory Access Profiler ............................................................................ 38
      5.4.2 Memory Energy Simulator ......................................................................... 38
      5.4.3 Cycle Accurate Energy Simulator ................................................................. 38
   5.5 Results ............................................................................................................. 39
      5.5.1 Data Object Clustering based Heuristic Algorithm .................................... 39
      5.5.2 Simulated Annealing Approach ................................................................. 41
      5.5.3 Comparison of Results .............................................................................. 41
   5.6 Standard Benchmarks Suits ............................................................................ 42

6 Conclusions and Future Work ................................................................................. 43
   6.1 Conclusions ..................................................................................................... 43
      6.1.1 Hardware Perspective ............................................................................... 43
      6.1.2 Software Perspective ................................................................................. 44
      6.1.3 Miscellaneous ......................................................................................... 44
   6.2 Possible Future Work ....................................................................................... 44

A Object Lifetime Analysis ......................................................................................... 47

B Decoder Overhead Characterisation ....................................................................... 51

C Behavior of Break-even Values ............................................................................... 53

D Results with Program Memory .............................................................................. 55
List of Tables

3.1 Details of Low Power Operating Modes (LPOMs) .................. 15
5.1 Comparison of Results ............................................. 42
A.1 Results with Shut Down (SD) Opportunities Included .......... 49
# List of Figures

1.1 Energy Distribution for (a) Uni-Processor Advanced RISC Machine (ARM) (b) Multi-Processor ARM Based Setups [1] .................................................. 2  
1.2 Expected Embedded Memory Static Power Requirements [2] .......... 3  
2.1 Simplified View of Typical Memory Structure [3] .......................... 6  
2.2 Example of Partitioned Memory [4] .............................................. 7  
2.3 Partitioning based on Memory Access Profile [3] ............................ 8  
2.4 (a) Monolithic and (b) Bit-width Aware Partitioned Memory [5] ....... 9  
2.5 Proposed Dual Region Memory System [6, 7] ................................. 9  
2.6 Memory Organization Optimized to Memory Footprints [8] .......... 10  
2.7 Memory Power States and Transition Costs [9, 10] ......................... 11  
3.1 Example Lifetime of Leakage-aware Memory Partition .................. 14  
3.2 Energy State Machine of Low Power Operating Modes (LPOMs) used in this Work ................................................................. 16  
3.3 (a) Dynamic Energy as Function of Memory Size (b) Static Power as Function of Memory Size .................................................. 17  
4.1 Inter-access Times of Data Objects ............................................. 20  
4.2 Illustration of $\tau_{ij}(d)\tau_{mn}$ ..................................................... 21  
4.3 Illustration of $\tau_{ij}(0)\tau_{mn}$ ...................................................... 21  
4.4 Energy Function with respect to Bi-partition Boundary for Different Data Object Mappings .................................................. 27  
5.1 (a) Ultra-wideband (UWB)-Receiver (RX) Block Diagram (b) UWB-RX Application-Specific Instruction-set Processor (ASIP) Energy Distribution ................................................. 34  
5.2 UWB-RX Frame Decoding on ASIP ............................................ 35  
5.3 Data Memory Accesses During Various Application Phases .......... 35  
5.4 Temporal Memory Access Profile of UWB-RX with Application Execution Phases Overlayed .................................................. 36
LIST OF FIGURES

5.5  (a) Spatial Memory Access Profile of UWB-RX (b) Size of Data Objects in Words 37
5.6  Execution Flow of Implemented Framework ............................. 37
5.7  Results of Heuristic Algorithm ............................................ 39
5.8  Partition Lifetimes for Best Result of Heuristic Algorithm .......... 40
5.9  Partition Lifetimes for Result of Simulated Annealing (SA) ........ 41

6.1  Scope for Energy Reduction with Bit-width Aware Architecture .... 45

A.1  Lifetimes of Data Objects over One UWB Frame ...................... 47
A.2  Partition Lifetimes for Result of SA Approach ......................... 48

B.1  Area Overhead of Decoder as a function of Number of Partitions 51

C.1  (a) Energy Consumed in Transfer to LPOMs (b) Energy Consumed by Being in LPOMs during Transfer Time (c) Static Power Difference as Function of Memory Size (d) Break-even Time Values as Function of Memory Size 54

D.1  Results of Heuristic Algorithm Applied to Program Memory ........ 56
D.2  Partition Lifetimes for Result of Heuristic Algorithm with Program Memory 56
Chapter 1

Introduction

Digital devices containing a processor and a memory have become an integral part of the day-to-day life. Especially the growth of mobile devices has been substantial in the last few years. Ability to process a piece of information to control or augment the functionality of these mobile systems make them ‘Embedded Systems’.

Apart from the cost perspective, these embedded systems are typically designed under three constraints [1]:

- Performance

- Power (energy) efficiency and

- Predictability (real-time responsiveness)

Especially the mobile devices have tight constraints on power consumption, as they are operated on a battery. Researchers have been trying to extend the state of the art so much, that efforts to make the mobile devices work without a battery have grown tremendously in the recent years.

The work presented in this thesis aims at improving the energy efficiency of an embedded system, specifically concentrating on the memory subsystem. It has been shown that the memory subsystem is a bottleneck in the design of any embedded system. Apart from the performance perspective, it is typically the second most power consuming part, after the processor itself [1]. Many studies have also shown that the memories in the data dominated applications consume even more than 50% of the total energy. Hence, power optimizations in the memory has been a popular research area since the 1990s. To get a more clear idea about the importance of the memory system as a candidate for power reduction, the next section describes a few experiments performed by the researchers. Section [1.2] presents the memory energy reduction problem statement in brief along with the major contributions of this work. Finally, this chapter concludes with the structure of the whole report explained in the Section [1.3].
CHAPTER 1. INTRODUCTION

1.1 Motivation and Future Trends

Figure 1.1 shows the results of the experiments performed on two test systems, to compute the energy distribution [1].

![Energy Distribution for (a) Uni-Processor ARM (b) Multi-Processor ARM Based Setups](image)

**Figure 1.1:** Energy Distribution for (a) Uni-Processor ARM (b) Multi-Processor ARM Based Setups [1]

Advanced RISC Machine (ARM) based processors were the focus of these experiments since they are quite common in mobile devices. The result shown is an average of more than 150 experiments. Experiments on the uni-processor ARM were carried out by varying the size and the latency of the main (off-chip) memory, for different benchmarks. Memory energy in the left pie-graph includes the combined energy of caches and main memory. It should be noted that, unlike uni-processor ARM, multi-processor ARM system had main memory residing on the chip. In case of the multi-processor ARM, number of processors were also varied. It can be clearly observed that memory subsystem consumes significant amount of power as compared to the processor itself. With the advancement in the Complementary Metal-Oxide-Semiconductor (CMOS) technology nodes, this trend is expected to continue in the future as well. Specifically the static power consumption in the memory is going to be a dominating factor of the future System on Chips (SoCs).

The 2011 edition of International Technology Roadmap for Semiconductors (ITRS) [2] presents the embedded memory power requirements for CMOS based Static Random-Access Memory (SRAM). Figure 1.2 shows the expected memory static power dissipation trend in future.

The steep increase implies that the memory is in general expected to play a significant role from power consumption perspective in the future. Note that since the power consumption in the memory heavily depends on the software application, the power optimization in memory involves a hardware-software co-design approach. Details about this are presented in subsequent chapters.

1.2 Brief Problem Description and Contributions

Thanks to the predeterminable nature of a typical embedded software application, power optimizations in the memory are feasible at the design time. This thesis work concentrates on improving the energy efficiency of an embedded system, more specifically concerning the
1.2. BRIEF PROBLEM DESCRIPTION AND CONTRIBUTIONS

Figure 1.2: Expected Embedded Memory Static Power Requirements [2]

energy consumed by the data memory. Since the use of SRAMs as a scratch-pad or even main memory has recently increased in energy efficient embedded systems, SRAM architecture\footnote{Cache memories are not considered in this work.} is the basis of the power optimizations presented in this work. Accredited to the increase in the static power consumption in memories, this work also makes use of Low Power Operating Modes (LPOMs) of SRAM. These are the low power standby states of the memories, aimed at reducing the power consumed in the idle condition. Memory partitioning techniques are also applied to reach an energy efficient memory configuration. In general, the problem dealt within this work can be stated as: given a hardware platform and SRAM memory hardware along with the software application, compute an energy efficient memory configuration considering both, dynamic and static energy consumption.

The main contributions of this work are:

- SRAM power consumption model of advanced 40 nm industrial memories supporting LPOMs.
- Fast and accurate memory energy simulator for industrial SRAMs.
- Memory access profiler with cycle accurate time-stamps.
- Automated framework to reach an energy efficient memory configuration which involves determining the:
  - Data placements
– Number of partitions and their sizes in words
– Time instances for transitions to LPOMs with the corresponding LPOM
– Memory partitioning overheads and their detailed analysis

1.3 Structure of Report

This thesis presents the work carried out as a masters project. Chapter 2 provides an overview of interesting and relevant scientific literature. With the chosen work areas, Chapter 3 introduces the concept of LPOMs to reduce the static energy consumption in the memory, in detail. A formal description of the power optimization problem along with the two methods to reach a solution are presented in the Chapter 4. The test system considered for all the experiments along with the results is presented in the Chapter 5. Finally, the Chapter 6 presents extensive conclusions of this work and motivates the possible directions for future work.
Chapter 2

Relevant Literature

Energy consumed by a memory can generally be expressed as,

\[ E_{\text{Memory}} = E_{\text{Dynamic}} + E_{\text{Leakage}} \]  \hspace{1cm} (2.1)

If detailed further as in Equation 2.2, it can be observed that the energy consumption in memory can be attributed to the four main terms/entities:

- Number of accesses to the memory \((N_{\text{Accesses}}\), dependent on the software application\)
- Energy consumed in one access to the memory \((E_{\text{Per\_Access}}\), dependent on the memory size\)
- Leakage power consumed by the memory \((P_{\text{Leakage}}\), dependent on the memory size\)
- Execution time of the software application \((T_{\text{Execution}})\)

\[ E_{\text{Memory}} = \frac{N_{\text{Accesses}} \times E_{\text{Per\_Access}}}{E_{\text{Dynamic}}} + \frac{P_{\text{Leakage}} \times T_{\text{Execution}}}{E_{\text{Leakage}}} \] \hspace{1cm} (2.2)

Note that the Equation 2.1 and 2.2 abstract away from the type of access \((\text{read or write})\). In practise this difference can be taken into account as well. Assuming that the \(N_{\text{Accesses}}\) and \(T_{\text{Execution}}\) are constant, one of the ways to reduce the power consumption in a memory system \(\text{architecture}\) is to partition the monolithic memory into different blocks, tailored to the specific software application. Partitioning aims at reducing the remaining two terms in the Equation 2.2. This chapter first introduces the concept of memory partitioning in brief and then enlists some of the important memory partitioning techniques presented in the literature. Details about the memory partitioning corresponding to the focus of this work are presented in the next chapter.
2.1 Concept of Memory Partitioning

The partitioned memory architecture in essence exploits the fact that the (dynamic) energy consumed in accessing a memory partition \( (E_{\text{Per Access}}) \) increases monotonically with the size of the memory. To get a better picture, Figure 2.1 shows the structure of a typical memory.

![Figure 2.1: Simplified View of Typical Memory Structure][3]

The power dissipation occurs in three main components/peripherals of this memory structure [11]:

1. Address decoders and word lines;
2. Data array, sense amplifiers, and the bit-lines; and
3. The data and address buses leading to the memory

The first two components are directly dependent on the size of the memory. One access to the memory block (partially) activates all of these components. Hence, reducing the size of the memory (by partitioning), clearly reduces the amount of energy consumed in one access; one of the terms in Equation 2.2. Basic principle behind the memory partitioning is to divide an address space into blocks and map these blocks to the distinct physical address spaces. Since these smaller blocks can now be accessed independently, energy consumed in accessing a block is reduced. Consequently, this helps in bringing down the overall energy consumption in the memory system. Figure 2.2 [4] explains this in a visual form.

Left-hand side of the figure shows a monolithic memory, which is split up into two blocks of equal size on the right-hand side. A decoder is necessary now to split the CE (Chip-Enable) signal for two banks. Address lines \( \text{Addr}[0\text{--}6] \) are buffered to ensure that delay in the buffer is compensated. Note that the overhead due to control logic, extra wiring and additional peripheral circuitry in general restricts the arbitrarily small partitioning. The next section presents a few important partitioning techniques presented in the scientific literature.
2.2 Memory Partitioning Techniques

Partitioning the monolithic memory has various implications. Although the reduction in the dynamic energy consumption was one of the initial motivations of researchers behind partitioning, the focus has changed in recent years. With advancement in the technology nodes, leakage energy consumption has become a dominant part of the whole SoC. The case with memories is not different either. Energy consumed in retaining the data while it is not being accessed, now consumes a large fraction of total energy consumption as well [12]. The techniques presented in the subsequent sections focus on reducing the dynamic and/or leakage energy consumption in memory.

2.2.1 Partitioning based on Memory Access Profile

A (micro-)architecture-level partitioning technique is presented in [13, 14, 15]. Reference [13] presents an algorithm which recursively bi-partitions the memory, to get an optimal partitioning based on the memory access profile of the application. Maximum number of partitions is fed as an input to the algorithm. The number of reads and writes per address are utilized to come up with a configuration which will reduce the amount of dynamic energy consumed, while still taking the decoder overhead into account. Figure 2.3 presents this graphically.
The original architecture is transformed to the partitioned architecture by utilizing the memory access profile of an application. Note that typically the embedded applications contain a small address range with a high number of accesses. Based on this access profile, the partitioned memory configuration will be different for different applications. Authors extend this work further in [14] by back annotating the layout information to drive the partitioning process. The results of the presented algorithm are claimed to be more realistic than publication [13] since it also accounts for the physical placement of the memory partitions on the chip. Finally, the same work is further extended in [15] by applying the concept of divided word-line [16] and divided bit-line [17]. Basic idea behind this division is to enable only a block of the memory macro, one at a time, thereby reducing the energy consumption. An efficient version of the algorithm using a dynamic programming approach is presented by the same authors in [18]. Also, the concept of address clustering is presented in [19], which is used to improve the efficiency of partitioning algorithm presented in [13]. Address clustering aims at the re-allocation of an address cluster (or range) to improve the results of previously presented partitioning algorithms.

2.2.2 Bit-width Aware Partitioning

Reference [5] presents a ‘bit-width aware’ memory partitioning approach. As shown in the Figure 2.4, authors make use of the notion of an effective bit-width (the smallest size which can hold both maximum and minimum values of a variable).
2.2. MEMORY PARTITIONING TECHNIQUES

The variables with smaller effective bit-width are placed in the narrower partitions. Authors exploit the fact that the total size of the memory is determined by the number of words and bit-width of the memory as well. Reducing either or both of these entities (by partitioning) is expected to yield a reduction in the energy consumption. An algorithm to achieve an efficient non-uniform banking (both, size and width) is presented in this article. Note that the effective bit-width in general can be heavily dependent on the application under consideration, as well as the input data.

2.2.3 Partitions with Different Threshold Voltages

As one of the first works to explicitly consider the leakage energy consumption in a memory, references [6, 7] present a circuit-level technique to reduce the amount of energy consumed in the memory. The idea is to split the monolithic memory into two regions, a dynamic energy aware (low \(V_{th}\), low \(V_{dd}\)) and static energy aware (high \(V_{th}\), high \(V_{dd}\)). To achieve this while keeping the reliability of the SRAM (variations in Static Noise Margin (SNM) and Write Margin (WM)), the supply voltage (\(V_{dd}\)) and SRAM cell size (\(\beta\) ratio) are altered. Figure 2.5 illustrates this graphically.
Two different voltage islands imply the necessity of level shifters in between. The restrictions on lowering the $V_{th}$ and $V_{dd}$ for the reliability of an access, are also considered in this work. The dynamic energy aware region is expected to be small in size, consuming little dynamic power and large static power. Since the frequently accessed data is placed in this region, the overall energy consumption of the system is expected to reduce. The static energy aware region is expected to be mostly idle, with low static power consumption. Hence it does not contribute much to the increase in overall energy consumption.

### 2.2.4 Cell Size vs. Error Rate Trade-off

Another interesting device-level technique for memory partitioning is presented in [8]. Authors tradeoff the accuracy of storage with area and power consumption of the memory. This is achieved by changing the memory cell size, as shown in the Figure 2.6.

![Figure 2.6: Memory Organization Optimized to Memory Footprints](8)

Accuracy and reliability of an SRAM cell are determined by the SNM and WM. While trading off these error rates, memory footprints of the software application are utilized to devise an optimized memory organisation. Higher order bits of the data, which are more important from the reliability perspective, are placed into wider partitions and vice versa. Pre-decoder takes this into account for address decoding. Such a partitioning has a risk of losing the reliability of the lower order bits, but with the gain of reduced energy consumption and chip area.

### 2.2.5 Leakage Aware Partitioning

Work presented in [9] first time explores the low power state or LPOM of the scratchpad (SRAM) memory. Authors present the Equation 2.3 as an analytical SRAM model considering a single low power (‘Sleep’) state.

\[
E = N_{acc} \cdot E_d + (T - T_S) \cdot P_{leak_s} + T_S \cdot P_{leak_s} + N_{sw} \cdot E_{SA}
\]

---

10
where, \( N_{\text{acc}} \) is the number of accesses to the memory block, \( E_d \) is the dynamic energy spent for each access, \( T \) is the total execution time in cycles, \( T_S \) is the sum of the cycles in which the memory is kept in sleep state, \( P_{\text{leak,A}} \) and \( P_{\text{leak,S}} \) are the amounts of leakage power spent in Active and in Sleep state, respectively, \( N_{\text{sw}} \) is the number of times the transitions from Sleep to Active state occur for a particular memory block and \( E_{SA} \) is the energy spent in one Sleep to Active transition. It is assumed that the value of \( E_{AS} \) (Active to Sleep transition) is negligibly small. All these components are function of the size of the memory block.

The idea here is to transit the memory partition into a Sleep state whenever the block is expected to be idle (not accessed) for a large amount of time. The partitioning is carried out in such a way that the opportunities for Sleep state transitions are increased. An exhaustive search is performed to determined the best partitioning scheme. It is feasible in their case, since the data reallocation is not considered in this work. It is also important to note that the Sleep state transition strategies should be designed in such a way that the gain obtained by transiting to the Sleep state does not backfire. It is achieved by making sure that the time for which a partition will be in the Sleep state, is large enough to overcome the cost incurred by the time and energy required to transit to and/or from this state. Authors introduce a concept of a ‘break-even’ (time) value. This is the amount of time for which the partition should be idle to make it worth to transit to the Sleep state.

In this article, Sleep state of the SRAM is characterized by \( V_{ddL} \) (lowered supply voltage). Active state is the state in which read or write access can take place. Reliable read or write access cannot be guaranteed in Sleep state and hence the transition back to Active state is essential before any access. The finite state machine with Active and Sleep energy states is shown in the Figure 2.7.

\[ T_{SA}(V_{ddL}, W) \]
\[ E_{SA}(V_{ddL}, W) \]
\[ P_{\text{leak,S}}(V_{ddL}, W) \]
\[ P_{\text{leak,A}}(V_{dd}, W) \]

**Figure 2.7: Memory Power States and Transition Costs [9][10]**

\( T_{SA} \) and \( T_{AS} \) are the time durations necessary for state transitions. Note that \( P_{\text{leak,A}}, P_{\text{leak,S}}, E_{SA}, T_{SA}, E_{AS} \) and \( T_{AS} \) are functions of \( V_{ddL} \) and \( W \) (size of the memory array).

### 2.3 DRAM Partitioning

It should be noted that partitioning a Dynamic Random-Access Memory (DRAM) and SRAM differs in many ways.
1. Micro-architectural implementation of DRAM and SRAM is completely different. DRAM uses only one transistor, whereas a typical implementation of SRAM cell contains six transistors. The leaking element in DRAM is the capacitor, whereas the complete set of transistors leak in the SRAM.

2. Unlike SRAM, a typical DRAM instance has various modes of operation, e.g. page mode, burst mode etc. Energy characterizations in these modes largely differ.

3. LPOMs of DRAM concentrate on refresh rate. LPOMs in SRAM can be implemented in many different ways; the next chapter explains this in detail.

4. Attributed to the comparatively larger sizes of the DRAMs, energy functions largely do not depend on the size of the memory. Whereas for the SRAMs, small change in the size (or width) or the memory is reflected immediately in all the components of the Equation 2.3.

The techniques presented in this chapter are applicable only to the SRAM. DRAM partitioning, although a matured field of research, falls outside the scope of this work.

2.4 Chosen Work Areas

Taking into account the fact that static power will contribute to the large amount of power consumption for future technology nodes, static power reduction is expected to be an interesting area of research in near future. Constrained by the available test platforms and tools, this thesis work focused on:

- Memory partitioning with non-uniform sized partitions (number of words) for dynamic and static energy reduction,
- Low Power Operating Mode (LPOM) transition strategies for reduction in static power consumption, and
- Energy efficient data placement considering both, dynamic and static energy

The partitioning techniques presented in Section 2.2 lack one or more of these focus areas. Note that the maximum number of partitions is not constrained and the size of each partition is determined by the amount of data placed into it. On the other hand, this work is constrained to the compile time static data allocation strategies in a single ported SRAM based memories in the uni-processor systems.

Before going for the actual description of the problem, the next chapter introduces the the concept of LPOMs in detail, along with the details about the costs of partitioning.
Chapter 3

Leakage Aware Memory Partitioning

As described in the previous chapters, partitioning of the memory aims at reducing the dynamic as well as static energy consumed in memory system. Partitioning for reducing the dynamic energy has been discussed in sufficient detail in the previous chapter. This chapter first introduces in detail the concept of Low Power Operating Modes (LPOMs) to reduce the static energy consumption. Then the implementation of LPOMs within the industrial memories used in this work is explored in Section 3.2. Finally, the overheads of memory partitioning which restrict the arbitrarily small partitioning of memory systems are discussed in the Section 3.3.

3.1 Concept of LPOMs

The Low Power Operating Modes (LPOMs) of SRAM are essentially useful in the advanced technology nodes where static power has become a dominant factor in the total energy consumption. This work uses the industrial 40 nm (typical corner) memories with already incorporated LPOMs. Main concept behind the usage of LPOMs is to reduce the energy consumed in memory when it is idle. An exemplary lifetime of the memory partition with LPOMs is shown in the Figure 3.1.

As indicated, the memory is in the ‘Normal’ mode of operation during \( t_0 \) to \( t_1 \). During this phase, it can be accessed (read or written) reliably. The energy consumed in accesses is shown as an addition to the static power consumed in this mode. Exploiting the idle time duration between \( t_1 \) and \( t_4 \), the memory (partition) can be transitioned into an LPOM. Notice that there is a definite amount of time and energy consumed in transition to/from this LPOM. To ensure that reduced static power consumption really brings in the energy savings, it is necessary to check whether it is worth to transit to an LPOM, considering the transfer energy and time. Since there can be more than one LPOM, with different static power, transition energy and time values attached, it is necessary to think about which LPOM to transit to as well. To explain this clearly, the model presented in the Equation 2.3 is extended with more number of LPOMs as shown below.

\[ \text{Equation 3.3} \]
CHAPTER 3. LEAKAGE AWARE MEMORY PARTITIONING

Figure 3.1: Example Lifetime of Leakage-aware Memory Partition

\[ E_{Memory} = N_{acc} \cdot E_d + \sum_{mode \in \text{LPOMs}} \left( (T - T_{mode}) \cdot P_{\text{StaticNorm}} + T_{mode} \cdot P_{\text{Static mode}} + N_{TF_{mode}} \cdot E_{TF_{mode}} \right) \]

(3.1)

where,

- \( E_{Memory} \) is the total energy consumption of a memory
- \( N_{acc} \) is the number of accesses to the memory
- \( E_d \) is the dynamic energy consumed in one access
- \( \text{LPOMs} \) is a set of low power modes
- \( T \) is the total execution time
- \( T_{mode} \) is the total amount of time for which memory is in the corresponding \( mode \)
- \( P_{\text{StaticNorm}} \) is the amount of static power consumed in the normal mode of operation
- \( P_{\text{Static mode}} \) is the amount of static power consumed in the corresponding \( mode \)
- \( N_{TF_{mode}} \) is the total number of transitions to and from the \( mode \)
- \( E_{TF_{mode}} \) is the total amount of energy consumed in one transfer to and from the \( mode \) combined

Based on this model, to ensure that the total energy is indeed reduced if the memory is transited to an LPOM, a break-even time value \( B_{mode} \) for a particular \( mode \) can be computed as,

\[ B_{mode} = \frac{E_{TF_{mode}} - (P_{\text{Static mode}} \times K_{mode})}{P_{\text{StaticNorm}} - P_{\text{Static mode}}} \]

(3.2)
where, $K_{mode}$ is the total time necessary to transit to and from the mode. So, for the exemplary situation depicted in the Figure 3.1 $K_{mode} = (t_2 - t_1) + (t_4 - t_3)$. Transition to the mode makes sense only if the idle time duration is greater than $B_{mode}$. What makes these calculations less straightforward is that all the power and energy entities in the Equation 3.1 and 3.2 are dependent on the size of the memory (partition). These sizes of the memory partitions cannot be predetermined since they depend on the amount of data placed into the partitions.

### 3.2 Implementation of LPOMs

Industrial SRAMs implement the LPOMs in various ways. Two of them are:

- By reducing the power supply voltage ($V_{dd}$) of the memory cells
- By applying a source bias voltage ($V_{ss}$) to the memory cells

Either of these techniques can be employed with (partial) shutdown of peripheral circuitry of memory. Memories used in this work apply the source bias voltage $V_{ss}$ along with periphery shutdown strategies. The static power consumed in these LPOMs is between 45% to 85% of the normal operating mode. The actual amount of savings depend on the LPOM under consideration and the size of the memory itself.

As stated before, LPOM transfer times and energies are dependent on the size (‘W’) as well as $V_{ss}$ in this particular instance of industrial memories. So for the set of memories used in this work, energy state machine is illustrated in the Figure 3.2.

The transition between the LPOMs without going to the Normal mode are not considered, accrediting to the compile time analysis of the software application. Modes Light Sleep (LS) and Deep Sleep (DS) are used in this work as both of them retain the data stored in the memory array. Nonetheless, results with the use of Shut Down (SD) mode by considering the data lifetimes, is presented in the Appendix A. The details about the LPOMs are presented in the Table 3.1. Note that these are limited due to the proprietary nature of third party industrial memories utilized.

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Normal/active mode</td>
</tr>
<tr>
<td>LS</td>
<td>Address decoder power gated (partial periphery shutdown), memory array source biased</td>
</tr>
<tr>
<td>DS</td>
<td>Complete periphery shutdown, memory array source biased</td>
</tr>
<tr>
<td>SD</td>
<td>Complete periphery shutdown, memory array shutdown</td>
</tr>
</tbody>
</table>

**Table 3.1:** Details of LPOMs

The behavior of all the variables in the Figure 3.2 was observed and modelled by generating various memory instances using the industrial memory compiler. Most of the variables in the

---

3Applicable particularly to the industrial memories used in this work.
state machine are linear functions of the memory size (in words). Example of the behavior of $E_{RD}$, $E_{WR}$, $P_{Static, Norm}$, $P_{Static, LS}$, $P_{Static, DS}$ and $P_{Static, SD}$ is presented in Figure 3.3.

The created model provides the value of all these variables for memory instances of all sizes and for all the LPOMs, based on interpolation. The range of the sizes of the memory shown in Figure 3.3 is restricted to two (internally) banked memory structures.

### 3.3 Partitioning Overheads

Arbitrarily small partitioning of the memories is restricted by the partitioning overheads. This section highlights the important overheads which are later on characterized for the solutions presented in this work.

- **Decoder/wrapper energy:** Although the energy overhead due to the peripheral circuitry of additional memory partition(s) is taken into account by the model presented in the previous section, decoder and wrapper (see Figure 2.2) energy characterization is necessary as well. This also includes the overhead due to additional wiring. As indicated in the available literature already [14], the energy consumed in the decoder is heavily dependent on the number of memory partitions. This is because the size of the decoder is determined by the number of outputs which it needs to provide, which
3.3. PARTITIONING OVERHEADS

Figure 3.3: (a) Dynamic Energy as Function of Memory Size (b) Static Power as Function of Memory Size

in turn determines the energy consumption in the decoder. Note that the value of this energy completely depends on the activity happening inside the decoder. The amount of activity being determined by various factors including the software application and data allocation, the total energy overhead due to decoder cannot be predetermined. This energy overhead value needs to be calculated every time a new memory configuration is considered. A bit more detailed discussion about the decoder overhead characterization is also included in the Appendix B.

• **Area:** The memory area model is also created along with the energy model presented in the previous section. It computes the additional area overhead incurred by the peripheral circuitry of additional memory partition(s). This area of peripheral circuitry is a part of the whole memory partition itself. But the decoder area overhead still needs to be taken into account separately. Even though this decoder area overhead is expected to be comparatively small as compared to that of peripheral circuitry, it is necessary to ensure that it remains within an acceptable bound.

• **Performance:** Transitions to and from the LPOMs require certain amount of time. Although it can be accounted for in the break-even value formula presented in Section 3.1, the processor needs to execute some instructions to issue commands for the LPOM transitions (e.g. enabling of the transition-enable pin of the memory). It is necessary to ensure that this performance overhead due to execution of extra instructions on the processor does not exceed the slack time duration of the software application execution.

• **Design time:** To reach an energy efficient solution regarding an energy efficient data placement and partitioning, the execution time of the algorithm needs to be considered. As will be illustrated in the next chapter, the execution time can vary from a few seconds to many hours. Even though finding an energy efficient partitioning configuration is a
one time task and needs to be computed only during the design time, unreasonably large amount of time necessary to reach a solution might not be acceptable either.

As a conclusion to this chapter, LPOMs of SRAM are expected to be useful in reducing the static energy consumption. But at the same time, they need to be utilized carefully to avoid the LPOMs to backfire in terms of the total energy consumption. Also, the data needs to be placed in different partitions such that the dynamic as well as the static energy consumption is reduced. The next chapter formally presents the optimization problem of computing an energy efficient memory configuration.
Chapter 4

Problem Description and Solutions

Previous chapters discussed various aspects of the power optimization problem dealt within this work. Solutions to the optimization problems concerning memory partitioning for reduction in the dynamic energy consumption have been discussed as well. Unlike these approaches, introduction of the time dimension to consider the static energy along with the dynamic energy makes the problem considered in this work much more complex. This chapter first presents the foundation for a concrete description of the problem in a more formal way. Then the search space of this optimization problem is analyzed in detail in Section 4.3. Finally, the chapter ends with a description of two methods to reach an energy efficient solution. The first method is a heuristic algorithm based on the efficient clustering of data objects, whereas the second method is based on the Simulated Annealing (SA) technique.

4.1 Basis of Formulation

Throughout this chapter, any variable or array used by a software application is termed as a ‘data object’. Also, it is assumed that a memory partition is characterized by the address range which it represents. Using these two concepts, Section 4.1.2 first presents a few definitions. Their use will become clear while formulating the problem in the subsequent sections. Relations between time intervals from Allen’s Interval Algebra [21] is used to formulate the core of this optimization problem. Important concepts of this algebra are first introduced in the next section.

4.1.1 Interval Relations from Allen’s Interval Algebra

Section 4.1.2 presents a few definitions necessary for the problem formulation. To formulate one of the definitions in this section, it is necessary to introduce a few properties of time intervals as presented in reference [21]. It is important to understand that the essence of static energy reduction lies in the idle time duration of a memory partition, because these are the time durations which will allow the transitions to the Low Power Operating Modes (LPOMs) as presented in the previous chapter. Idle time duration of memory partition is determined by the idle time durations of the data objects placed into it. Figure 4.1 presents
the access times and idle (or inter-access) times of data objects in a graphical manner.

**Figure 4.1: Inter-access Times of Data Objects**

$DOx$ and $DOy$ represent the data objects $x$ and $y$ respectively. As shown, $DOx$ is accessed at times $t_i$, $t_j$, and $t_k$. Hence the ‘idle time set’ $T_{DOx}$ of $DOx$ contains two idle intervals, $\tau_{ij}$ and $\tau_{jk}$. The exemplary height of the time intervals represented correspond to the size of the data object. Significance of this size of the data object will become clear in subsequent sections. It is assumed that only one address location belonging to a single data object can be accessed at a single time instance. After the information regarding access times of all the data objects is obtained, it is necessary to take into account all possible situations corresponding to the idle intervals of two data objects which are to be placed into same memory partition. For this, two distinguishing base relations mentioned in the Reference [21] between any set of intervals, are useful. As discussed below, these are the two situations which can occur with respect to the idle time sets of $DOx$ and $DOy$, to be placed in the same partition.

- Figure 4.2 presents a case when $\tau_{ij}$ occurs during $\tau_{mn}$. It is denoted by $\tau_{ij}(d)\tau_{mn}$.

  In this situation, the placement of $DOx$ and $DOy$ in the same partition would change the combined idle interval set to $\{\tau_{mi}, \tau_{ij}, \tau_{jn}\}$. This way the duration of the idle time interval $\tau_{mn}$ is reduced. Since this reduction can possibly cost an opportunity to transit to an LPOM, the data object placements need to be made carefully.

- Similarly, Figure 4.3 presents the case when $\tau_{ij}$ overlaps with $\tau_{mn}$. It is denoted by $\tau_{ij}(o)\tau_{mn}$.

  In this case, the combination of idle time sets due to placement of the two data objects in the same partition, adds interval set $\{\tau_{mi}, \tau_{mj}, \tau_{jn}\}$ to the combined idle time set. Since the duration of both $\tau_{ij}$ and $\tau_{mn}$ is reduced, this placement would possibly imply the loss of two LPOM transition opportunities.

In other words, both the relations presented above imply a sort of a lateral merging of access times due to the placement of two (or more) data objects in the same partition. This way, the opportunities for LPOM transitions are determined by the data object placements, and the break-even values are determined by the size of the partitions which can only be computed...
after fixing a particular data object placement. Superficially, the data objects with similar access patterns are better to be placed in the same partition. But even though the problem might look like an instance the (access) pattern matching paradigm, the interdependence of variables with the importance of partition size as explained above makes it a combinatorial optimization problem. Also, the placement of data objects with similar access patterns in the same partition from the static energy perspective, but that does not take into account the number of accesses (and hence the dynamic energy consumption) to the corresponding partition. Dependence of the access energies on the size of the memory and necessity of simultaneous optimization for dynamic and static energy consumption, makes this problem even more complex.
CHAPTER 4. PROBLEM DESCRIPTION AND SOLUTIONS

4.1.2 Definitions of Concepts Used in Formulation

With the concept of access intervals presented in the previous section, it is also necessary to introduce some more terminology required by the problem formulation. The definitions which are useful for the formulation in the Section 4.2 are presented below. In all these definitions, the bold identifiers denote sets whereas calligraphic symbols denote address ranges.

Definition 1. Address range \( R = [A, B] \) indicates a range of addresses corresponding to a single data object or a set of data objects placed in a single partition. \( A \) and \( B - 1 \) are the first and last addresses in \( R \). \( |R| \) indicates the number of words within the address range \( R \). In other words, this is the size of the partition represented by the address range \( R \).

Definition 2. \( \text{Reads}(DOx) \) and \( \text{Writes}(DOx) \) indicating the number of reads and writes to data object \( DOx \) are obtained from memory access profiling of an application.

Definition 3. \( \text{Reads}(R) \) and \( \text{Writes}(R) \) respectively denote the number of read and write accesses to the address range \( R \).

Definition 4. \( P \) denotes a set of memory partitions, each having an address range characterized by the data object(s) placed into it. \( n_R(P) \) is the cardinality of set \( P \).

Definition 5. An idle time interval is a time interval \( \tau_{ij} = [t_i, t_j] \) in which certain address range \( R \) is not accessed. \( |\tau_{ij}| \) denotes the length of an idle time interval \( \tau_{ij} \).

Definition 6. \( T_{DOx} \) is a set of idle time intervals corresponding to the data object \( DOx \).

Definition 7. The idle count \( n_I(T) \) of a set of idle time intervals \( T \) is the cardinality of the set \( T \). This count is later required to calculate the number of LPOM transitions.

Definition 8. The idle duration \( T_I(T) \) of a set of idle time intervals \( T = \{\tau_1, \ldots, \tau_{n_I(T)}\} \) is defined as \( \sum_{i=1}^{n_I(T)} |\tau_i| \). This value is useful to calculate the amount of time spent in a specific power mode.

Definition 9. The composition of the idle time intervals of two data objects, \( T_{DOx} \) and \( T_{DOy} \), denoted by \( T_{DOx} \oplus T_{DOy} \), is defined as (using Allen’s interval algebra [21])

\[
T_{DOx} \oplus T_{DOy} = \left\{ T_{DOx} \cup T_{DOy} \mid \forall \tau_{ij} \in T_{DOx}, \tau_{mn} \in T_{DOy}, \right. \\
(\tau_{ij}[d]\tau_{mn} \rightarrow T_{DOx} := ((T_{DOx} \setminus \tau_{ij}) \cup \tau_{im} \cup \tau_{mj})) \land \\
(\tau_{ij}[o]\tau_{mn} \rightarrow T_{DOy} := ((T_{DOy} \setminus \tau_{mn}) \cup \tau_{in})) \\
\left. \land (\tau_{ij}[d]\tau_{mn} \rightarrow T_{DOy} := ((T_{DOy} \setminus \tau_{mn}) \cup \tau_{mi} \cup \tau_{jn})) \right\}
\]

Definition 10. Operation of composition obeys the law of associativity. In other words, the order in which the idle interval sets of two or more data objects are composed together, is not important.

\[
T_{DOx} \oplus T_{DOy} \oplus T_{DOz} = (T_{DOx} \oplus T_{DOy}) \oplus T_{DOz} = T_{DOz} \oplus (T_{DOy} \oplus T_{DOz})
\]
4.1. BASIS OF FORMULATION

Definition 11. $T_R$ is a set of idle time intervals corresponding to the address range $R$.

Definition 12. Deep Sleep Interval set $\text{DSI}(R)$, is the set of idle intervals for which the partition represented by $R$ is in DS mode.

Definition 13. Light Sleep Interval set $\text{LSI}(R)$, is the set of idle intervals for which the partition represented by $R$ is in LS mode.

Definition 14. The time duration for which a partition represented by an address range $R$ is in DS mode is denoted by $T_I(\text{DSI}(R))$.

Definition 15. The time duration for which a partition represented by an address range $R$ is in LS mode is denoted by $T_I(\text{LSI}(R))$.

Definition 16. The time duration for which a partition represented by an address range $R$ is in Normal mode is denoted by $T_I(T_R \setminus \text{LSI}(R) \cup \text{DSI}(R))$.

Definition 17. $P_{\text{Static L P O M}}(|R|)$ is the static energy consumed by a partition characterized by the address range $R$, while being in the corresponding L P O M.

Definition 18. $E_{TF,\text{L P O M}}(|R|)$ is the (combined) energy necessary to transfer the partition represented by $R$ to and from the LPOM. It can be computed as $E_{\text{entry L P O M}}(|R|) + E_{\text{exit L P O M}}(|R|)$. Here $E_{\text{entry L P O M}}(|R|)$ and $E_{\text{exit L P O M}}(|R|)$ are the energies consumed by one transfer to or from the LPOM respectively. $E_{RD}(|R|)$ and $E_{WR}(|R|)$ respectively denotes the energy consumed in one read and write access while accessing a partition represented by $R$. Values of all these functions are computed using the model presented in the Chapter 3.

Definition 19. $K_{\text{L P O M}}(|R|)$ is the time necessary to transform the mode of operation of a memory partition. It is defined as $t_{\text{entry L P O M}}(|R|) + t_{\text{exit L P O M}}(|R|)$.

Definition 20. OV($N$) indicates the energy overhead of decoding circuitry for $N$ partitions. This function can be obtained by performing power simulations for the application, with different number of memory partitions. The average of the experiments with different data placements can be used to model the behavior of this function.

With these definitions taken into account, the formulation presented in the next section assumes the following.

- The granularity of the time intervals is one clock cycle, denoted by $T_{\text{cycle}}$. Note that it is easily possible to lift this restriction. The time intervals can be made to coincide with the application execution phase boundaries. But it should be ensured that application execution phase boundaries are fine tuned to exploit high inter-access times across the data objects.

- The boundaries of an address range always coincide with the boundaries of the data object(s) placed into it. In other words, partial allocation of a data object to a certain address range (or a partition) is not considered. This assumption can also be easily modified. But it might not be reasonable since behavior of complete data object is
expected to remain same throughout the application execution. Currently it is assumed that the energy efficient data object split-up has already been performed.

- The access scheduling [22, 23] of the application has already been carried out for better results. The techniques like loop split-up [1] might be specifically useful in this case to increase the idle time durations of data objects, and hence the partitions. It will also become apparent from the results presented in the Chapter 5 that the analysis performed in this work would create a valuable feedback to the software designer.

- The number of partitions in which a memory is to be partitioned, is constant. An energy efficient solution must be found for different number of partitions separately. The simulated annealing technique presented in the Section 4.5 does not have this restriction.

4.2 Problem Formulation

Considering the focus areas presented in Section 2.4 this work tries to solve two tightly coupled problems.

1. Efficient data object placements to reduce dynamic and static energy consumption in memory

2. Determining the time instances suitable for energy savings by transitions to the LPOMs

For a particular data object placements, this section describes the invariants in the optimization problem of determining LPOM opportunities, with a cost function to be minimized. All the computations presented in this section are necessary to create a context corresponding to the different data object mappings to the partitions. The data object mappings and the LPOM transition opportunities together decide the total amount of energy savings obtained.

1. Extension of a single address range over all the data objects placed in a partition would mean that the idle interval sets of all the data objects are composed together as expressed below.

\[
\forall R \in P \ T_R = \bigoplus_{DOx \in R} T_{DOx}
\]

2. Break-even time value for a partition to be in the LS mode is computed as follows.

\[
\forall R \in P \ B_{LS}(|R|) = \left( \frac{E_{TF_{LS}}(|R|)}{P_{StaticNorm}(|R|)} - P_{Static_{LS}}(|R|) + K_{LS}(|R|) \right)
\]

3. Break-even time value for a partition to be in the DS mode is computed as follows.

\[
\forall R \in P \ B_{DS}(|R|) = \left( \frac{E_{TF_{DS}}(|R|)}{P_{StaticNorm}(|R|)} - P_{Static_{DS}}(|R|) + K_{DS}(|R|) \right)
\]
4. Deep Sleep Interval set $\text{DSI}(R)$ of a partition, is composed of the idle time intervals greater than the break-even threshold of DS mode ($B_{DS}(|R|)$).

$$\forall R \in P \forall \tau_{ij} \in T_R \left( |\tau_{ij}| > B_{DS}(|R|) \right) \rightarrow \text{DSI}(R) := \text{DSI}(R) \cup \tau_{ij}$$

5. Light Sleep Interval set $\text{LSI}(R)$ of a partition, is composed of the idle time intervals greater than break-even threshold of LS mode ($B_{LS}(|R|)$) and less than or equal to $B_{DS}(|R|)$.

$$\forall R \in P \forall \tau_{ij} \in T_R \left( |\tau_{ij}| > B_{LS}(|R|) \land |\tau_{ij}| \leq B_{DS}(|R|) \right) \rightarrow \text{LSI}(R) := \text{LSI}(R) \cup \tau_{ij}$$

6. Static energy consumed by a partition while in the Normal mode of operation, is a product of the static power consumed in the Normal mode and time duration for which the partition is in the Normal mode.

$$\forall R \in P E_{\text{StaticNorm}}(|R|) = T_I(T \setminus (\text{LSI}(R) \cup \text{DSI}(R))) \times P_{\text{StaticNorm}}(|R|)$$

7. Static energy consumed by a partition while in the LS mode of operation, is a product of the static power consumed in the LS mode and time duration for which the partition is in the LS mode.

$$\forall R \in P E_{\text{StaticLS}}(|R|) = T_I(\text{LSI}(R)) \times P_{\text{StaticLS}}(|R|)$$

8. Static energy consumed by a partition while in the DS mode of operation, is a product of the static power consumed in the DS mode and time duration for which the partition is in the DS mode.

$$\forall R \in P E_{\text{StaticDS}}(|R|) = T_I(\text{DSI}(R)) \times P_{\text{StaticDS}}(|R|)$$

9. Energy consumed in transferring the mode of operation of a partition to LS, is the product of the number of transfers and the energy per transfer.

$$\forall R \in P E_{\text{TransferLS}}(|R|) = (n_I(\text{LSI}(R))) \times E_{TF_{LS}}(|R|)$$

10. Energy consumed in transferring the mode of operation of a partition to DS, is the product of the number of transfers and the energy per transfer.

$$\forall R \in P E_{\text{TransferDS}}(|R|) = (n_I(\text{DSI}(R))) \times E_{TF_{DS}}(|R|)$$

11. The number of read accesses to a partition is the sum of the read accesses to the data object(s) placed into it.

$$\forall R \in P \text{Reads}(R) = \sum_{DO_x \in R} \text{Reads}(DO_x)$$

---

4. Static power consumption in DS mode is always less than that of LS mode irrespective of the size of the memory.
12. The number of write accesses to a partition is the sum of the wrote accesses to the
data object(s) placed into it.

\[ \forall R \in P \quad \text{Writes}(R) = \sum_{DOx \in R} \text{Writes}(DOx) \]

13. Dynamic energy consumed in the read access to a partition is the product of the num-
ber of reads and energy consumed in a single read access.

\[ \forall R \in P \quad E_{\text{DynamicRead}}(|R|) = \text{Reads}(R) \times E_{RD}(|R|) \]

14. Dynamic energy consumed in the write access to a partition is the product of the num-
ber of writes and energy consumed in a single write access.

\[ \forall R \in P \quad E_{\text{DynamicWrite}}(|R|) = \text{Writes}(R) \times E_{WR}(|R|) \]

15. Overhead of decoding circuitry necessary for a particular partitioned memory configu-
ration is represented as below.

\[ \text{OV}(n_R(P)) \]

16. Finally, the cost function to be minimized is the total energy consumed by the corre-
sponding memory configuration.

\[ E_{\text{Total}}(P) = \text{OV}(n_R(P)) + \sum_{R \in P} E_{\text{StaticNorm}}(|R|) + E_{\text{StaticLs}}(|R|) + E_{\text{StaticDS}}(|R|) + E_{\text{TransferLs}}(|R|) + E_{\text{TransferDS}}(|R|) + E_{\text{DynamicRead}}(|R|) + E_{\text{DynamicWrite}}(|R|) \]

Notice the similarity of this equation with Equation 3.1 presented in the previous chap-
ter.

The formulation presented in this section is necessary to compute the energy consumption
of a particular memory configuration and corresponding data object placements. To find the
optimal solution to this problem means to find the data object placements, number of parti-
tions, size of partitions and the LPOM transition instances, which will have the lowest energy
consumption. Remember that it is necessary to reduce the static as well as dynamic energy
consumption simultaneously. Next section analyses the search space of this optimization
problem, to help build a method to reach an energy efficient solution.
4.3 Analysis of Search Space

To get an idea about (ir-)regularities in the search space, a few experiments were performed based on Reference [10]. Although the authors in [10] use only one LPOM, and the micro-architectural implementation of LPOM is different in their case, similar search space analysis is still applicable to this work. As shown in the Figure 5.1 the energy function is quite irregular, with several local minima.

Figure 4.4: Energy Function with respect to Bi-partition Boundary for Different Data Object Mappings

The plots of the energy function were obtained by varying the bi-partition boundary in random steps, for random mappings of the data objects. Bi-partition boundary here implies the address location at which the monolithic memory is split-up to make a memory configuration with two partitions. The irregularity of this function increases even more with increase in the partition cardinality and inclusion of lifetime analysis as presented in the Appendix A. To get a better idea about why there is no regularity in the search space, Appendix C presents the behavior of break-even values with respect to the size of the memory. Also, for different data object sequence/mapping, the energy function behaves in unpredictable way. In general, the search space of this combinatorial optimization problem is non-convex. Typically this means
that it is necessary to evaluate each and every candidate solution to reach a global-optimum. To get an idea about the size of the solution space for an exhaustive search, Equation 4.1 computes the time complexity of the brute-force implementation. Assume that \( N \) is the number of data objects in a particular software application, and \( P \) is the number of partitions in which a monolithic memory is supposed to be partitioned. Any memory partition in a configuration with two or more partitions can have different combinations of data objects placed into it. This is denoted by \( \binom{N}{k} \). Also, the number of data objects in a partition can vary between 1 to \( N - P - 1 \). This means,

\[
\text{Solution space size} = \sum_{k=1}^{N-P-1} \binom{N}{k}
\]

For \( P = 2 \),

\[
= \sum_{k=0}^{N} \binom{N}{k} - \binom{N}{0} - \binom{N}{N}
\]

\[
\text{Solution space size} = 2^N - 2 \quad \text{(4.1)}
\]

Hence for a two partition memory configuration the size of the solution space, being exponential in the number of data objects, restricts the feasibility of the exhaustive search. This is because the actual number of computations for a general case of an embedded application with even 50 data objects, becomes unacceptably large. A non-convex search space with an exponential complexity generally implies that it is necessary to reside to a heuristic algorithm which will provide a near-optimal solution. This work presents two approaches/heuristics to solve this optimization problem, in the following sections. The next section describes a heuristic algorithm based on the clustering of data objects and Section 4.5 provides a method based on Simulated Annealing (SA).

### 4.4 Data Object Clustering based Heuristic Algorithm

As explained in the previous section, an exhaustive search is necessary to determine the optimal partitioning solution. Since this is typically not feasible, it is necessary to reduce the size of the search space in a smarter way. To reduce the time complexity of the problem, a two step approach is presented below.

Step 1: Obtain a data object order/sequence which considers number and pattern of accesses to the data objects. This mapping also tries to maximize the opportunities to transit to an LPOM, by placing the data objects with similar access patterns close to each other.

Step 2: Exhaustively search for the energy efficient partition boundary within this data object order. The number of partition boundaries depend on the number of partitions in which a monolithic memory is to be partitioned.
4.4. DATA OBJECT CLUSTERING BASED HEURISTIC ALGORITHM

4.4.1 Step 1 - Data Object Clustering

To reach an energy efficient data object order/sequence considering both the dynamic and static energy consumption, this section presents a set of heuristics. This is the first step of the two step approach presented above. The assumptions and inputs to the algorithm are described below.

- The set of time instances at which an accesses to the memory occurs is denoted by \( T \).
  \[ T = \{t_1, t_2, \ldots, t_N\} \]

- The access trace of a monolithic memory, corresponding to the data objects is described by \( DO_T \).
  \[ DO_T = \{DO_{t_1}, DO_{t_2}, \ldots, DO_{t_N}\} \]
  Here \( DO_{t_i} \) indicates the data object which is accessed at time instance \( t_i \).

- Set \( S \) is a non-repetitive, exclusive and exhaustive set consisting of all possible 2-tuples of data objects. Here non-repetitive means that the set does not contain two different tuples with same data objects. This set contains \( \frac{N \times (N-1)}{2} \) number of tuples, where \( N \) is the number of data objects. Also, no tuple contains two instances of the same data object.
  \[ S = \{(DO_i, DO_j) \mid DO_i \neq DO_j, \forall DO_i, DO_j \in DO_T\} \]

- Each tuple in the set \( S \) has a value attached with it, to indicate how useful it is to place the corresponding data objects together. \( S_{x,y} \) indicates the value of the tuple in the set \( S \), made up of data objects \( DO_x \) and \( DO_y \).

- Function \( f(t_i, t_j) \) returns a value which is inversely proportional to the difference between \( t_i \) and \( t_j \).
  \[ f(t_i, t_j) = \frac{t_N}{t_j - t_i} \]

- Function \( g(N_{accDO_x}, N_{accDO_y}) \) returns a value which is directly proportional to the number of read and write accesses to the data objects \( DO_x \) and \( DO_y \).
  \[ g(N_{accDO_x}, N_{accDO_y}) = \frac{(N_{accDO_x} + N_{accDO_y}) \times T_{cycle}}{t_N} \]

- Function \( sort(S) \) returns a set of tuples, sorted in descending order, based on the value attached with each tuple.

- Function \( merge(S) \) returns a list of data objects, formed using 2-tuples in set \( S \). This function also performs the repeat removal, while prioritizing the data objects in the order obtained after sorting. Repeat removal is necessary to avoid a placement of the same data object at two distinct location at the same time.
CHAPTER 4. PROBLEM DESCRIPTION AND SOLUTIONS

First the information regarding inter-access times of consecutively accessed data objects is extracted. This is useful to exploit the LPOMs of the memories by matching the temporal memory access pattern of data objects. In other words, it accounts for a fraction of total application execution time for which the memory is not accessed. Then the number of accesses to each data object are taken into account. Assuming that one cycle is necessary to access the SRAM based memory, this accounts for a fraction of total execution time for which memory gets accessed. Finally, a cost function created in this way is used to come up with a data object sequence.

Algorithm 1 compute an efficient data object sequence

Require: application memory access profile
Ensure: \( \forall t_i,t_j \in T \), \( S_{t_i,t_j} = 0 \)

\( DT = \{DO_1, DO_2, \ldots, DO_N\} \)

for \( t_i = 1 \rightarrow N - 1 \) do
  if \( DO_i = DO_x \) and \( DO_{i+1} = DO_y \) and \( DO_i \neq DO_{i+1} \) then
    \( S_{x,y} \leftarrow S_{x,y} + f(t_i, t_{i+1}) \)
  end if
end for

for all \( S_{x,y} \in S \) do
  \( S_{x,y} \leftarrow S_{x,y} + g(N_{accDO_x}, N_{accDO_y}) \)
end for

\( S \leftarrow \text{sort}(S) \) \( \triangleright \) sorting based on cost function value

data object sequence \( \leftarrow \text{merge}(S) \) \( \triangleright \) merging and repeat removal

4.4.2 Step 2 - Search for Efficient Partitioning

In the second step of the approach, all possible partition boundaries are evaluated to compute the most energy efficient memory configuration. By employing this two step approach, the size of the solution space is drastically reduced. Step 2 being the computationally intensive step of the approach dominates the time complexity of this heuristic algorithm.

\[
\text{Solution space size} = \left( \begin{array}{c} N \\ P - 1 \end{array} \right) \\
\text{Time complexity} = \Theta(N^{P-1})
\]

Here \( N \) is the number of data objects and \( P \) is the number of partitions. So the number of partitioning combinations is equal to \( \binom{N}{P-1} \). Note that typically the number of partitions \( P \ll N \). Hence it holds that \( N^{P-1} \ll 2^N \) which is the complexity of exhaustive search (Equation 4.1). In essence, the heuristic algorithm based on data object clustering reduces
the time complexity to an acceptable value. As will be shown with the results presented in the next chapter, this algorithm also makes it feasible to reach an energy efficient solution within substantially small amount of time.

### 4.5 Simulated Annealing based Partitioning Algorithm

One of the well established methods to find a near-optimal solution in a large non-convex search space is the Simulated Annealing (SA) [20] approach. This work implements a variant of random restart SA. Simply stated, if a candidate solution sustains 1000 restarts with random initial guesses, it is finalized as the best candidate solution. Parameters to the SA algorithm are determined empirically to reach a better solution. While calculating these, the time to reach a solution is taken into account as well. This algorithm has no optimality claim, but reaches an energy efficient solution within an acceptable amount of time. Also, since this algorithm does not restrict the data object sequence/mapping at any stage of execution, it is expected to bring more energy savings as compared to the heuristics presented in previous section. At the same time, runtime of the SA algorithm can get quite large since it is dependent on the empirically determined parameters.

The next chapter introduces the test platform used for the experiments in this work. The problem formulation presented in the Section 4.2 is utilized in the data object clustering based heuristic algorithm. Also, the results of SA approach are presented thereafter.
Chapter 5

Implementation, Results and Evaluation

This chapter provides the details about the framework implemented to reach an energy efficient memory configuration. First, Section 5.1, 5.2 and 5.3 introduce the test platform used for all the experiments. Then the additional tools developed as a part of the framework are presented in Section 5.4. Finally the results are summarized in the Section 5.5. Note that the goal of this work is the optimization of memory energy consumption and not of the processor itself. Details about the software application execution flow of the test system, which are not interesting for data memory access analysis, are hence omitted from the discussion.

5.1 Hardware Analysis of Test System

At the starting point of this work, a test platform was chosen for experiments, which is explained below in brief. This system as a whole, is a Ultra-wideband (UWB)-Receiver (RX). Low transmission power inherent in the UWB protocol makes it especially attractive for the wireless sensor node applications. UWB-RX is an SoC consisting of an Application-Specific Instruction-set Processor (ASIP) along with various Application-Specific Integrated Circuits (ASICs) to perform fixed tasks, as shown in Figure 5.1(a).

The ASIP works with the program memory, data memory and vector memory. The hardware components related to the memories in the UWB-RX, are highlighted by red color in the Figure 5.1(a). All of these memories are single ported SRAMs. The initial distribution of average power for the memory related components in the system, is shown in the Figure 5.1(b). This is the result of an application execution on the ASIP at 62.4 MHz ($T_{cycle} \approx 16$ ns). As can be seen, the data and program memory together constitute more than half of the total power consumption. Even though the results that will be presented are for the data memory only, the same approach and framework can be applied to the program memory as well. The results with the program memory are described in Appendix D.
5.2 UWB Frame Decoding

UWB-RX receives a UWB frame after a fixed time interval. The frames are decoded and processed by the ASIP repeatedly. To get an idea about the application execution phases which the ASIP encounters, Figure 5.2 shows the UWB frame structure in brief.

Preamble, Start Frame Delimiter (SFD), PHY Header (PHR) and Physical layer Service Data Unit (PSDU) are in essence the chunks of data in the UWB frame, which are received and decoded over various execution phases of the ASIP. Initialization, Synchronization, Tracking and PHR data collection phases are much shorter as compared to the Burst Processing [24]. The correspondence between the UWB frame and the ASIP application execution phases is also shown in the Figure 5.2. The important property of this test application which makes it interesting for leakage aware memory partitioning related work is the streaming execution with sparsely placed accesses in temporal dimension. Data memory experiences the largest number of accesses during the Burst Processing phase, as shown in Figure 5.3. This is the longest executing application phase as well.

Since all the other phases except the Burst Processing phase have small execution time, they have relatively small inter-access (or idle) times for most of the data objects, as depicted in Figure 5.4. During this phase, the ASIP essentially waits for the PSDU burst to arrive, and then decodes it.

The Burst Processing phase is hence expected to introduce a large number of LPOM transition opportunities for the memory partitions.

5.3 Software Analysis of Test System

Figure 5.5(a) shows the number of accesses to various data objects placed into a data memory. Irregular spread of access counts is clearly visible from this graph. It is necessary to

---

5The system used in this work is the next generation of the system presented in [24]
5.3. SOFTWARE ANALYSIS OF TEST SYSTEM

Transmitted frame
PREAMBLE SFD PHR PSDU
Time
ASIP 
Application 
Execution 
Phases
UWB 
Frame 
Structure
Burst Processing and End of Frame
(Burst_Proc, EoF)
PHR Data Collection Phase (PHR)
Synchronization and Tracking Phase (Sync, Track)
Initialization and Synchronization Phase (Init, Trigger_TX, Sync)
/Preamble detected
SFD detected
Preamble detected
SFD detected

Figure 5.2: UWB-RX Frame Decoding on ASIP

Figure 5.3: Data Memory Accesses During Various Application Phases
take into account the sizes of these data objects, as shown in Figure 5.5(b). This is because the placement of data objects in memory partitions determines the size of the partitions. Size of the memory partition in turn determines the total energy consumed by that partition and hence the total memory system. The data objects with size ‘1’ are the flags, each of the size of one word. Some of these objects are not accessed in the current implementation of the system.

As discussed previously, not only the access counts, but also the temporal access patterns of data objects are necessary to be considered from static energy point of view. Figure 5.4 shows the access times of various data objects, for one frame of decoding and execution. Alternating blue and red overlay is used to indicate the different application execution phases. If observed closely, the access patterns of data objects 23, 42 and 62 can be visually classified as matching with each other. Not only these ones, but tuples (51, 52, 53, 54) and (33, 34, 35) have matching patterns, amongst others. These kind of matchings might help increase the LPOM transition opportunities. Also note that only one data object can be accessed at any time instance. This is because the ASIP is a single core processor and the memories used are single ported as well.

---

Figure 5.4: Temporal Memory Access Profile of UWB-RX with Application Execution Phases Overlaid

---

6The second frame of the simulation data is considered to get fair results since the finite amount of wake-up time of the ASIP is counted in the first frame.
5.4. DEVELOPED TOOLS AND MODELS

One of the main contributions of this work is a complete framework for reaching an energy efficient memory configuration, as shown in the Figure 5.6.

Following subsections describe the tools and models which were built as a part of implementing this framework. As a first step, the application code was transformed to enhance the opportunities of LPOM transitions. Main focus was on elementary split-up of the complex data structures. The C-structures were split up into smaller variables as far as possible. From the hardware end, power consumption values of SRAM memory instances with different bit-lines and word-lines were extracted from the data-sheets. Next subsection describes the Memory Access Profiler built to capture and log the accesses to the memory. Following two subsections explain two memory energy simulators developed as part of the framework described in Figure 5.6.

Figure 5.5: (a) Spatial Memory Access Profile of UWB-RX  
(b) Size of Data Objects in Words

Figure 5.6: Execution Flow of Implemented Framework
5.4.1 Memory Access Profiler

To create an access profile of the application, all the accesses to the memory system are first captured using a logger. Then a bi-dimensional matrix is built, whose rows represent the data objects and columns represent the execution cycles. An element in the matrix is 1, only if the corresponding data object is accessed at the respective cycle. This matrix now contains the spatiotemporal information of all the data memory accesses. The spatial information (number of accesses to a data object) can be evaluated as the sum of the elements in the row which represents the data object. Whereas the temporal information (inter-access times) is the difference between cycle times at which the corresponding data object is accessed.

5.4.2 Memory Energy Simulator

Based on the SRAM power consumption values obtained from the data-sheets, an SRAM model based on interpolation was built as described in the Section 3.2. Basic motivation behind this model is to be able to compute the energy consumption of a memory configuration without actually implementing it. The energy calculator returns an energy consumption of an input memory configuration based on the number of partitions and the corresponding data object placements, by using the SRAM model. It uses the transformed memory access profile of an application as an input. This transformation has the information regarding the LPOM transition instances. The energy reduction technique evaluates the possible opportunities for LPOM transitions, and performs this transformation before feeding it to the energy calculator. To check the accuracy of this simulator, various experiments were carried out with different number of memory partitions. For each of the experiment, energy consumption of energy calculator was compared with the value obtained from the industrial gate level power simulator. As compared to the industrial gate level power simulator, this energy calculator has an average accuracy of more than 85%.

5.4.3 Cycle Accurate Energy Simulator

To get a better idea about how exactly the industrial gate level power simulator computes the power consumption, a cycle accurate energy simulator was built as well. The main difference between this simulator and the ‘top-level’ energy calculator presented in the previous subsection is that this simulator considers a toggling of single bit while computing the energy. Unlike the top-level simulator presented in previous subsection, this simulator has a better estimate of read and write access energy because of this. The access energy values used in the top-level simulator are the long-run average power values computed by considering an average case of an active/access cycle. More specifically, the average power is computed for,

- Switching of Memory Enable (ME) and Write Enable (WE) bits,
- Switching of 50% of the address and data bits

Also, it is assumed that alternate cycles are active. The cycle accurate simulator does not have these assumptions and hence its average accuracy is more than 95% as compared to
the industrial gate level simulators. The experiments carried out to check the accuracy are explained in the previous subsection. This accuracy value is also useful to have a better estimate of the closeness of the energy savings with real implementations.

5.5 Results

With the use of the tools presented in the previous section and applying the energy reduction techniques presented in Sections 4.4 and 4.5, this section presents the results of memory partitioning framework. Section and present the results of Data Object Clustering based Heuristic Algorithm and Simulated Annealing based Partitioning respectively. Whereas Section 5.5.3 compares the results of two approaches.

5.5.1 Data Object Clustering based Heuristic Algorithm

The heuristic algorithm presented in the Section 4.4 was executed by fixing the number of partitions beforehand. For each of this execution, the lowest energy consuming partitioning scheme was chosen as a candidate solution. The exploration with increasing number of partitions was halted when the trend of monotonous increase in the energy consumption was observed for two consecutive partition count increments. Results of these runs are shown in the Figure 5.7.

As can be expected, the dynamic energy consumption reduces with increase in the partition cardinality. This is because the accesses are directed towards smaller sized memory. But, as
mentioned already, energy consumed by (now necessary) decoding and peripheral circuitry restricts the arbitrarily small partitioning. Reduction in dynamic energy while going from the monolithic configuration to a single partition configuration with LPOMs is due to the removal of unused data objects. Note the steep increase in the static energy consumption with increase in the partition cardinality. This can be accredited to the high amount of leakage in the advanced technology node. The use of LPOMs helps bring down this by a substantial amount, making a two partition configuration the most energy saving one. The increase in static energy consumption of peripheral circuitry and memory array increases so much that the four partition configuration in fact worsens the energy consumption of the memory.

\[\begin{array}{c}
\text{Partition 1} \\
(4032 \text{ Words, 80 Accesses})
\end{array}\]

\[\begin{array}{c}
\text{Partition 2} \\
(2112 \text{ Words, 2545 Accesses})
\end{array}\]

\text{Start of Burst Processing}

\text{Normal mode}

\text{Light Sleep mode}

\text{Deep Sleep mode}

\[\begin{array}{c}
2.3 \\
2.5 \\
3.0 \\
3.5 \\
4.0
\end{array}\]

\text{Time in ms}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure5.8.png}
\caption{Partition Lifetimes for Best Result of Heuristic Algorithm}
\end{figure}

Figure 5.8 presents the lifetimes of the partitions in this configuration. Black lines represent the boundaries of the application execution phases. Most opportunities to transit to a DS mode are obtained during the Burst Processing phase. As was expected from the presented heuristic algorithm, dynamic energy consumption is also considered while deducing an energy efficient partitioning scheme. Unlike partition 1, partition 2 has highly accessed data with comparatively small size. As a practical aspect of this problem, the presented energy reduction technique also takes into account the practically available size of the memory partitions. This is because the industrial memory compiler cannot generate random sized memory instances.
5.5. RESULTS

5.5.2 Simulated Annealing Approach

Results of the Simulated Annealing (SA) approach are similar to that to heuristic algorithm based on data object clustering. Figure 5.9 presents the lifetimes of the most energy saving configuration. SA algorithm was executed for a constraint of maximum five partitions. The result in return is the two partition configuration, with the largest energy savings. Although the number of partitions is same as that of the heuristic algorithm, the enhanced opportunities to place the data objects in different partitions further helps to bring down the energy consumption by approximately 3%. Next section presents the details about the solution of the SA, along with the comparison between the two results.

![Figure 5.9: Partition Lifetimes for Result of SA](image)

5.5.3 Comparison of Results

This section presents a comparison of the results of two approaches presented in this work. Value of the design time required presented in the Table 5.1 is referring to a 2.2 GHz AMD Opteron computer equipped with 100 GB of RAM.

The energy savings brought in by the SA approach is at an expense of increased area overhead. This increment is again due to a practical consideration. The memory compiler cannot generate an instance with a single (internal) bank above certain value. In other words,
exceeding this size means that we need to have a memory instance with two (internal) banks. This incurs the additional overhead of the peripheral circuitry. Negligible performance and decoder/wrapper energy overheads underlines the usefulness of the presented solution.

5.6 Standard Benchmarks Suits

It was important to get an idea about proximity of the solutions of above two algorithms with a global-optimum solution. For this reason, a few applications from the standard benchmark suits such as MiBench [25] and Powerstone [26] were ported on the UWB-RX ASIP. It turns out that the optimal solution in case of these small synthetic applications is always a single partition. Also, the number of opportunities to transit to an LPOM are negligible, or even none in many cases. A few possible reasons behind this are presented below.

1. The size of the data objects (and the total required memory size) is small in case of these synthetic applications. Overhead of static energy consumption incurred by adding a memory partition is too large as compared to the savings brought in by reduction in dynamic energy consumption.

2. Break even-values for the monolithic memory configuration are too large to get any opportunity for an LPOM transition.

How much energy savings can this framework bring for other streaming applications remains an interesting open question.
Chapter 6

Conclusions and Future Work

As final remarks, this chapter concludes the work done while highlighting the important findings and insights. The next section presents the conclusions, and Section 6.2 provides some directions towards a possible future work.

6.1 Conclusions

The power optimization problem dealt within this work has a hardware-software co-design paradigm. Consequently, this section is divided into three parts, the conclusions based on hardware perspective, software perspective and the miscellaneous conclusions.

6.1.1 Hardware Perspective

1. The most important conclusion based on the observations of the results is that the static energy has a steep increase with the increase in the number of partitions as seen in Figure 5.7. This trend is expected to continue with the advancement in technology nodes (28 nm and above).

2. The LPOMs of the memory are extremely useful in bringing down this static energy by transiting a memory block into a low leakage state whenever feasible. This can be seen from the Figure 5.7 as well.

3. Arbitrary transitions to the LPOMs are not energy efficient due to a finite energy and time cost attached with each transfer. A careful analysis of the application is necessary to exploit the time intervals with potential to save the static energy.

4. Feasibility of generating a specific sized memory instance from a memory compiler plays an important role in computing a realistic value of energy savings.
6.1.2 Software Perspective

1. The data object placements play a crucial role in lowering the energy consumption. Placement considering both the dynamic and static energy are essential in this case. Placing the highly accessed data objects in a smallest sized partition possible aims at reducing the dynamic energy consumption. Whereas placing the data objects with similar access patterns in the same partition aims at increasing opportunities for LPOM transitions. The optimization problem needs to be solved by having both these considerations simultaneously.

2. Frequency of application execution on the processor is important as well; faster completion of the computation would mean more idle time and enhanced opportunity for LPOM transition.

3. Streaming applications with large idle times are attractive from the memory energy saving perspective. The processor in this case is active during a relatively small period, which implies large amount of time where memories need to remain idle (un-accessed) while still retaining the data.

6.1.3 Miscellaneous

1. The presented problem is a combinatorial optimization problem with a large non-convex search space. The techniques presented are able to yield an energy efficient solution within a satisfactory amount of time, as shown in the Table 5.1. The energy savings up to 25% are obtained using the developed framework.

2. The current amount of variables in this multidimensional multivariate search space are large enough to blow up the size of the search space beyond acceptable bounds. Fine tuning of the current variables as well as addition of any new variable needs a careful consideration to ensure a possibility to reach a reasonable solution.

3. The granularity of the address space and time-line split up is extremely important. This work used the granularity of the data object sizes as a split up of the address space, while the finest possible granularity in temporal dimension (one cycle) was utilized. Finest granularity in both dimensions is the best case possibility, but this might explode the complexity of the optimization problem.

6.2 Possible Future Work

Motivated by techniques presented in the currently available scientific literature and DRAM partitioning techniques, this section provides a set of possible future works.

1. Bit-width aware SRAM partitioning has been presented in the Chapter 2 already. To get an idea about the scope of this technique to reduce the energy consumption, for the test system used in this work, 6.1 shows the corresponding size and number of accessed with respect to a specific bit-width.
As can be easily observed, the majority of the data objects and accesses are inclined towards larger bit-width values. Putting in the fact that the memory compiler can only generate the memory instances with bit-width of 8 or more, this specific test application does not seem have large scope for energy improvements in this direction. Analyzing this and possible inclusion of bit-width dimension for different applications still remains as a future work.

2. Current implementation of the framework determines the data object placements at the compile/design time. Based on the application execution phases, runtime data/memory management can be an interesting direction of research. Though authors in Reference [27, 28] provide a compiler based framework for runtime data management of partitioned SRAM architecture, the size and the number of partitions is fixed independent of the software application in their case.

3. Along with the runtime data migration, data compression can as well bring in the energy savings. Recomputing the data whenever the fetching cost far exceeds the computation cost is a possible approach as well. Note that even though these techniques have been exploited and proven useful in the literature for DRAM partitioning, the tight coupling of energy numbers with the memory size in case of SRAM makes the problem extremely complicated.

4. This work assumes the correct data object split-up. In practise, a precise data object split-up has to be made in such a way that the whole address range represented by the data object has a similar access pattern over the execution time period.

5. Overlaying the data objects which have exclusive lifetimes will help reduce the total size of the memory and hence will bring in the energy savings. Obtaining an optimal overlay is far from trivial though. This is because even though overlaying the data objects will reduce the size of the memory, it will also reduce the number of LPOM transition opportunities. Which overlay will give the best solution needs an extremely careful problem formulation.
6. The results can be improved from the hardware perspective as well. As can be seen from the Figure 5.7, a large increase in the static energy restricts the number of partitions. More effective implementation of the power switches to reduce the leakage in the memory partitions in LPOMs, would help reduce the overall energy consumption as well.
Appendix A

Object Lifetime Analysis

The results presented in this work make use of only two out of three available LPOMs of the industrial memories. The reason behind not using the SD mode is that the contents of the memory are lost after transiting to this mode. Careful analysis of the ‘lifetimes’ of the objects (time duration from first write to last read) helps to exploit the opportunities to shut down a memory partition, since it is not necessary to retain the data outside its lifetime. Figure A.1 presents the lifetimes of the data objects in UWB-RX application, over one frame of execution. Grey and white lines are only used to distinguish alternate data objects.

Figure A.1: Lifetimes of Data Objects over One UWB Frame

Note the large white spaces indicating the time duration for which the corresponding object is dead. This means that the partition in which the object is placed in, can be shut down during that time period, assuming all the other data objects in the same partition are dead.
during that period as well. Considering the amount of SD opportunities that can be obtained by looking at the Figure A.1 and taking into account the fact that memory configurations with large number of partitions are not energy efficient (as is evident from the results presented in the Figure 5.7), a large number of gain in terms of energy with the use of SD mode is not expected to be achieved. Nonetheless, a few things change with inclusion of the lifetime analysis in this optimization problem.

1. First and foremost, the search space becomes more irregular, making the global optimal solution even harder to reach. This is because, now the data object placements will also decide the lifetimes and hence the shut-down opportunities of the partitions.

2. The problem formulation presented in the Section 4.2 changes in a sense that, now the transition to SD mode has higher priority than DS mode, if the lifetime analysis allows to do so. To check this, the lifetimes of the data objects placed in the same partition are merged to check for the opportunities to shut-down this partition.

After allowing for SD opportunities, the result obtained with the Simulated Annealing (SA) approach presented in the Section 4.5 is depicted by Figure A.2. This result is obtained by constraining the maximum number of allowed partitions to five.

![Figure A.2: Partition Lifetimes for Result of SA Approach](image)

Similar to the results without SD mode, the leakage aware partitioning does account for dynamic energy as well. Large number of accesses tend to go into smaller partitions. Detailed
analysis of this result is presented in the Table A.1.

<table>
<thead>
<tr>
<th></th>
<th>Simulated Annealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy savings (% of monolithic memory)</td>
<td>24.5</td>
</tr>
<tr>
<td>Number of partitions</td>
<td>3</td>
</tr>
<tr>
<td>Area overhead (% of monolithic memory)</td>
<td>17.2</td>
</tr>
<tr>
<td>Performance overhead (% of total execution time)</td>
<td>$15.7 \times 10^{-4}$</td>
</tr>
<tr>
<td>Decoder/wrapper overhead (% of total energy)</td>
<td>1.5</td>
</tr>
<tr>
<td>Design time (Hours)</td>
<td>15.7</td>
</tr>
</tbody>
</table>

**Table A.1:** Results with SD Opportunities Included

These results can be compared with the ones presented in the Table 5.1. Due to increase in the number of partitions, the result with SD opportunities included incurs larger area overhead, with only marginal energy savings. The decision of the choice of the solution is at a discretion of the system designer.
Appendix B

Decoder Overhead Characterisation

As explained in the Section 3.3, accurate decoder overhead needs to be characterized separately for each application and particular data mappings. Since this is not feasible, a few experiments were carried out to get an idea about a possible behavior of decoder energy overhead function. Irrespective of the application-specific activity in the decoder, the floor area of the decoder can give an idea about the possible energy consumption in it. As shown in Figure B.1, the decoder area linearly increases with the number of partitions.

Figure B.1: Area Overhead of Decoder as a function of Number of Partitions

This area is an estimation presented in the post-Register-Transfer Level (RTL) compilation report. Various experiments were also carried out to underline the fact the decoder area is largely dependent on the number of partitions, and not the size of the partitions. Assuming that the average amount of activity in the decoder is constant irrespective of the other param-
APPENDIX B. DECODER OVERHEAD CHARACTERISATION

eters, this characterization might be useful for more accurate estimation of decoder energy overheads in future works. Characterization of the wiring energy overhead falls out of the scope of this work though.
Appendix C

Behavior of Break-even Values

Numerous experiments were carried out to get an idea of the search space of the optimization problem. The aim was to find a possible regularity, which might help reaching a better solution in a smarter way. One of the key issues in reducing the static energy is determining the right instances of LPOM transitions. For this, the Equation 3.2 represents the method to compute the so-called ‘break-even value’.

\[ B_{\text{mode}} = \frac{E_{TF_{\text{mode}}} - (P_{\text{Static}_{\text{mode}}} \times K_{\text{mode}})}{P_{\text{Static}_{\text{Norm}}} - P_{\text{Static}_{\text{mode}}}} \]

Since the variables on the right hand side of the equation are functions of the size of the memory, \( B_{\text{mode}} \) depends on memory size as well. To get an idea about the behavior of the break-even values for different LPOMs, Figure C.1(d) shows a normalized break-even value plot with respect to the memory size.

Analysis of this behavior for each of the LPOM is discussed below separately.

- **Shut Down**: As can be expected, the amount of time which is necessary to make it worth shutting down a memory increases with the memory size. The break-even values of the SD mode are always less than those of DS mode. It can be accredited to the lower values of transfer energies of SD mode. The reason behind this is the implementation of the SD mode as explained in the Table 3.1. Simply stated, the SD mode does not need to activate additional circuitry for source biasing as opposed to the DS mode.

- **Deep Sleep**: The break-even values of the DS mode increase with the memory size as well. The reason for these values being higher than those of the SD mode is as explained before.

- **Light Sleep**: The break-even values of LS mode have an unexpected behavior. Reduction in the values with increase in size is accredited to the slopes of functions \( E_{TF_{\text{LS}}} \), \( (P_{\text{Static}_{\text{mode}}} \times K_{\text{mode}}) \) and \( (P_{\text{Static}_{\text{Norm}}} - P_{\text{Static}_{\text{LS}}}) \) respectively. Even though the savings obtained by reduction in the static power in LS mode are increasing with the memory size, transfer cost in terms of the energy is almost constant. A (possible) reason behind this might be that the periphery circuitry included in the partial shutdown is relatively independent of the size of the memory.
Figure C.1: (a) Energy Consumed in Transfer to LPOMs
(b) Energy Consumed by Being in LPOMs during Transfer Time
(c) Static Power Difference as Function of Memory Size
(d) Break-even Time Values as Function of Memory Size

It is important to note that,

1. The details about the internal implementation of the memory is of a proprietary nature to the memory vendor. The limited amount of information presented in the manuals is at times insufficient to reach a concrete conclusion.

2. The range of the memory sizes presented in the Figure is only for a set of memory structures with two (internal) banks. The behavior, albeit similar, is spanned over a different memory size ranges for different number of internally banked structures.
Appendix D

Results with Program Memory

The results presented in Section 5.5 are for the data memory of the UWB-RX. The same framework and energy reduction techniques can be applied to the program memory as well, with the following differences:

1. The data object now becomes a program object. This is a C-function which can be relocated with the help of a linker in the utilized test platform. In other words it can be called as a subroutine in an assembly language code. Since linker allows to replace and takes care of references related to this object, code replacements are possible in program memory as well.

2. There will not be any shut-down opportunities, as explained in the Appendix A. This is because the code needs to be retained for a complete execution of application.

Figure D.1 shows the results of the application of the data clustering based heuristic algorithm mentioned in Chapter 4 to the program memory.

The main difference between this plot and Figure 5.7 is that now the dynamic energy accounts for a larger fraction of the total energy consumption in any memory configuration. This is expected since the program memory is getting accessed much more frequently as compared to the data memory. Nonetheless, similar trends can be observed in this plot as well, but now with more than 28% as the best achievable energy savings. Lifetimes of the partitions over one frame of execution are shown in Figure D.2.

Again, large amount of opportunities for LPOM transitions are obtained during the burst processing phase. Now with program memory, increase in the amount of time spent in the Normal mode of operation is evident as well. This is expected since typically the program memory is more busy as compared to the data memory.

In general, the framework presented in this work can be applied to any SRAM\(^7\) with predetermined set of memory objects to be stored in it.

\(^7\)Not the cache memories.
Figure D.1: Results of Heuristic Algorithm Applied to Program Memory

Figure D.2: Partition Lifetimes for Result of Heuristic Algorithm with Program Memory
Acronyms

ARM  Advanced RISC Machine. xv, 2
ASIC  Application-Specific Integrated Circuit. 33
ASIP  Application-Specific Instruction-set Processor. xv 33–36 42
CMOS  Complementary Metal-Oxide-Semiconductor. 2
DRAM  Dynamic Random-Access Memory. 11 12 44 45
DS  Deep Sleep. 15 23–25 40 48 53
ITRS  International Technology Roadmap for Semiconductors. 2
LPOM  Low Power Operating Mode. vii xiii xv 3 4 10 12 20 22 24 26–28 30 34 36–38 40 42 47 53–55
LS  Light Sleep. 15 23–25 53
ME  Memory Enable. 38
PHR  PHY Header. 34
PSDU  Physical layer Service Data Unit. 34
RTL  Register-Transfer Level. 51
RX  Receiver. xvii xvii 33–37 42 47 55
SA  Simulated Annealing. xvi 19 28 31 41 48
SD  Shut Down. xiii 15 47–49 53
SFD  Start Frame Delimiter. 34
SNM  Static Noise Margin. 9 10
SoC  System on Chip. 2 7 33
SRAM  Static Random-Access Memory. 2 3 9–13 15 18 30 33 37 38 44 45 55

57
Acronyms

**UWB**  Ultra-wideband. xv, xvi, 33–37, 42, 47, 55

**WE**  Write Enable. 38

**WM**  Write Margin. 9, 10
Bibliography


