Computer Science Department

University of Crete

FPGA implementation of a Cache Controller with Configurable Scratchpad Space

Master's Thesis

Giorgos Nikiforos

January 2009

Heraklion, Greece

University of Crete

Computer Science Department

FPGA implementation of a Cache Controller with Configurable Scratchpad Space

Thesis submitted by

Giorgos Nikiforos

in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

Heraklion, January 2009

Computer Science Department

University of Crete

FPGA implementation of a Cache Controller with Configurable Scratchpad Space

Master's Thesis

Giorgos Nikiforos

Abstract

Chip Multiprocessors (CMP) are the dominant architectural approach since the middle of this decade. They integrate multiple processing cores on a single chip. It is desirable for future CMP's to support both implicit and explicit communication. Implicit communication occurs when we do not know in advance which input data will be needed, or who last modified them; caches and coherence work well for such communication. Explicit communication is when the producer knows who the consumers will be, or when the consumer knows its input data set ahead of time; scratchpad memories and remote DMA work best, then.

This thesis designed and implemented a cache controller that allows a (run-time) configurable part of its cache to behave as directly addressable, local, scratchpad memory. We merged this cache controller with the virtualized, user-level RDMA controller and network interface (NI) that was designed by other members of our group. An FPGA prototype is implemented, around the MicroBlaze softcore processor. We implemented an external 1-clock-cycle L1 data cache, and a pipelined L2 cache with interleaved banks, operating with a 4-clock-cycle latency, and offering an aggregate access rate (to the processor and to the NI) of two words per clock cycle;

way-prediction is used to reduce power consumption. The scratchpad capability is integrated at the level of the L2 cache. We do not yet provide for coherence with corresponding caches of other processors.

The design consumes one fifth of the resources in a medium-size FPGA. The merged cache controller and NI is about 8 percent more area-efficient than the sum of a distinct cache controller and a separate NI. We evaluate the performance of the system, using simulations and micro-benchmarks running on the MicroBlaze processor.

Supervisor professor: Manolis Katevenis

Τμήμα Επιστήμης Υπολογιστών

Υλοποίηση σε FPGA ενός ελεγκτή κρυφής μνήμης με ρυθμιζόμενο χώρο μνήμης SRAM

Μεταπτυχιακή Εργασία

Γιώργος Νικηφόρος

Περίληψη

Τα (**chips**) Πολυεπεξεργαστικών Συστημάτων αποτελούν την βασική αρχιτεκτονική τάση από τα μέσα της δεκαετίας. Ενσωματώνουν πολλαπλούς πυρήνες επεξεργασίας στο ίδιο **chip**. Το επιθυμιτό στα μελλοντικά συστήματα είναι να υποστηρίζουν και έμμεσους (**implicit**) και άμεσους (**explicit**) τρόπους επικοινωνίας. Έμμεσος τροπος επικοινωνία συμβαίνει όταν δεν είναι γνωστό εξαρχής που βρίσκοναι τα δεδομένα· είδος επικοινωνίας που παρατηρείται σε συστήματα κρυφής μνήμης. Άμεσος τρόπος επικοινωνίας παρατηρείται όταν ο παραγωγός γνωρίζει ποιος είναι ο καταναλωτής· εφαρμόζεται σε συστήματα με μνήμες **scratchpad** και **Remote DMA (RDMA)** .

Σ' αυτή την εργασία σχεδιάστηκε και υλοποιήθηκε ένας ελεκτής (**controller**) κρύφης μνήμης ο οποίος επιτρέπει δυναμικά ένα μέρος της κρυφής μνήμης να συμπεριφέρεται ως ευθέως διευθυνσιοδοτούμενη τοπική μνήμη (**scrachpad**). Ενοποίησαμε αυτόν το ελεκτή κρυφής μνήμης με ένα εικονικό σε επίπεδο χρήστη, ελεκτή **RDMA** και μια διεπαφή δικτύου η οποία σχεδιάστηκε από άλλα μέλη του εργαστηρίου.

Συγκεκριμένα, υλοποιούμε ένα πρωτότυπο βασισμένο σε **FPGA** και τον επεργαστή **MicroBlaze**. Υλοποιήσαμε μια εξωτερική κρυφή μνήμη πρώτου επιπέδου με ένα κύκλο καθυστέρηση και μια κρυφή μνήμη δευτέρου επιπέδου που είναι **pipelined** και **bank interleaved** με 4 κύκλους καθυστέρηση προσφέροντας συνολικό ρυθμό προσπελάσεων 2 λέξεις ανά κύκλο ρολογιού (μια στον επεξεργαστή και μια στο δίκτυο)· πρόβλεψη των ενδεχόμενων δρόμων της μνήμης των δεδομένων περιορίζει την κατανάλωση ισχύος. Ο χώρος **scratch-pad** βρίσκεται στην **(L2)** μνήμη. Δεν περιλαμβάνεται ακόμα **coherence** με τις άλλες κρυφές μνήμες των άλλων επεξεργαστών.

Διαπιστώσαμε ότι το κόστος του συστήματός σε εμβαδόν είναι το ένα πέμπτο μιας μεσαίου μεγέθους **FPGA**. Η υλοποιημένη αρχιτεκτονική είναι 8% αποδοτικότερη μια αρχιτεκτονική με δύο ξεχωριστούς ελεγκτές για κρυφή μνήμη και **DMA**. Αξιολογούμε την απόδοση του συστήματος μέσω εξομοιώσεων, προσομοιώσεων και ειδικών απλών προγραμμάτων που εκτελούνται στον επεξεργαστή. Τα ειδικά αυτά προγράμματα εξομοιώνουν βασικές λειτουργίες που παρατηρούνται σε κανονικά προγράμματα.

Επόπτης καθηγητής: Μανόλης Κατεβαίνης

Acknowledgments

I feel grateful to my supervisor, Prof. Manolis Katevenis as also Prof. Dionysios Pneumatikatos for their valuable assistance and guidance in my academic steps in the field of Computer Science. The extensive discussions about my work and their wide knowledge have been of great value for me.

A big thanks to George Kalokairinos, Vasilis Papaefstathiou anb Stamatis Kavadias for their support and help to a major part of my work. Moreover, I would like to thank my friend and colleague, Dimitris Tsaliagos, for his help.

My warmest appreciation to the following, past and current, members of the CARV Laboratory of the FORTH-ICS, whom I feel to be friends more than just colleagues: Mixalis Ligerakis, Manolis Marazakis, Mixalis Flouris, Markos Foundoulakis, Giorgos Tzenakis, Stavros Passas, Vagelis Mangas Thanos Makatos, Mixalis Alvanos, Giannis Klonatos, and Zoe Zembekou.

Last but not least, I would like to thank my family, my parents Nikos and Eleni and my brother Michalis for the support and encouragement they provided me with.

This work was carried out with the financial and technical support from FORTH - ICS and the Europoean Commission in the context of the SARC integrated project (FP6 #27648), and with the assistance of the HiPEAC Network of Excellence.

Giorgos Nikiforos

Heraklion, January 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer pioneers correctly predicted that programmers would want unlimited amounts of fast memory. The rate of improvement in microprocessor speed exceeds the rate of improvement in DRAM (Dynamic Random Access Memory) speed. Hence computer designers are faced with an increasing Processor-Memory Performance Gap (Figure 1.1) [15], which is the primary obstacle to improved computer system performance. A perfect memory system is one that can supply immediately any datum that the CPU requests. This ideal memory is not practically implementable; because, the four factors of memory speed, area, power and cost are in direct opposition. More than 70% of the chip area is dedicated to memory. As a result the power consumption increases and can reach 80% of the global system consumption. An economical solution, then, is a memory hierarchy organized into several levels, each smaller, faster, and more expensive per byte than the next. The goal is to provide a memory system with cost almost as low as the cheapest level of memory and speed almost as fast as the fastest level. The concept of memory hierarchy takes advantage of the principle of locality. Further, smaller is faster; smaller pieces of hardware will generally be faster than larger pieces. This simple principle is particularly applicable to memories built from similar technologies for two reasons. First, larger memories have

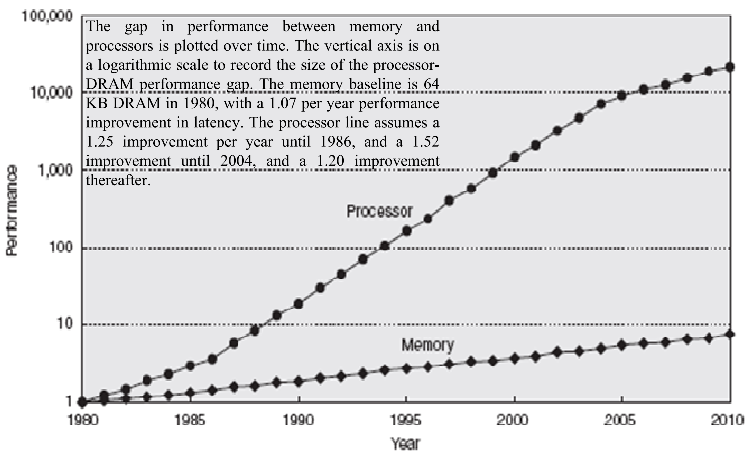

FIGURE 1.1: CPU Memory gap

more signal delay and require more levels to decode addresses to fetch the required datum. Second, in most technologies smaller memories are faster than larger memories. This is primarily because the designer can use more power per memory cell in a smaller design. The fastest memories are generally available in smaller number of bits per chips at any point in time, and they cost substantially more per byte. Memory hierarchies of modern multicore computing systems are based on the two dominant schemes; either multi-level cache (with coherence support), or scratchpads (with DMA functionalities). The case of caches is usually met in general purpose systems due to the transparent (implicit) way of handling data locality and communication. Data are located and then moved not under the direct control of the application software; instead, data copies are placed and moved as a result of cache misses or cache coherence events, which are indirect only results of application software actions.

The benefit is simplicity: the application programmer needs not worry about where data should reside and how and when they should be moved. The disadvantage is inability to optimize for the specific data transport patterns that occur in specific applications; such optimization becomes especially important in scalable systems, because the mechanisms of cache coherence-being ignorant of what the application is trying to do-fail to deliver acceptable performance in large-scale multiprocessors.

Scratchpads are the on-chip SRAM, which is a small, high-speed data memory that is connected to the same address and data buses with off-chip memory. Scratchpad, like cache, has a single processor cycle access latency. One main difference between the scratchpad SRAM and data cache is that the SRAM guarantees a single-cycle access time, whereas an access to cache is subject to compulsory, capacity, and conflict misses. Another major difference between cache and scratchpad is the power consumption and area overhead. In scratchpad memories there is no need of comparators for tag matching and the tag area is used for saving data, so with the same memory size, scratchpad has more memory space for data, due to the lack of tags.

Scratchpads are very popular in embedded and special purpose systems with accelerators, where complexity, predictable performance and power are dominant factors in real-time applications such as multimedia and stream processing. Scratchpads are also used in modern heterogeneous multicore systems which support hundreds of general purpose and special purpose cores (accelerators).

Due to the explicit communication mechanisms, systems using scratchpad, become a necessity for large-scale multiprocessors because of the scalable performance. The programmers concern, in those systems, is about worst-case performance and the battery life time and that is why embedded system designers do not choose hardware intensive optimizations in the quest of better memory hierarchy performance as most general purpose system designers would do. The cost in embedded systems burdens the programmers because those systems have to deal with communication and locality explicitly which means how to best control data placement and transport.

Scratchpads memories overcome the problems arise from conflict misses and the cache line granularity transfers which are major cache characteristics. Scratchpad memories can support data transfer of any size, less than the memory size. However, comparing to cache, scratchpad memories do not include logic circuit responsible for performing its read and write operations in bursts. In platforms using cache memories, the data transfers between hierarchy levels are performed by cache circuits which copy cache line size while the processor is not concerned on how the data are fetched. In platforms with scratchpad memory hierarchy, the processor has to perform each data block transfer between levels using Direct Memory Access (DMA) circuits. However for their initialization, extra cycles are needed by processor for computing the data block source and destination addresses in the memory layers and for programming the DMAs to perform the transfers.

When designing a (System-on-Chip) SoC, the most general solution is to employ a memory hierarchy containing both cache and scratchpad in order support both primitives for every kind of application. However the inclusion of scratchpad in the memory system raises the question of how best to utilize the scratchpad space. In SoCs where each processor includes in its memory system both cache and scratchpads in different memory blocks, underutilization problem arises depending on the application running on the system. In cases of non-deterministic applications scratchpads are useless so only caches are fully utilized. On the other hand, when stream processing applications are running on the system, caches are not fully utilized.

Towards the direction of memory system utilization, our work focuses on the configurability of the SRAM-blocks that are in each core, near each processor, so that they operate either as cache or scratchpad or dynamic mix of them.

We also strive to merge the communication subsystems required by the cache and scratchpad into one integrated Network Interface (NI) and Cache Controller (CC), in order to economize on circuits. Network Interface and generally the integration of NI and CC is outside of the scope of this dissertation. In many parts of this document there will be some references to characteristics of this merging.

This master thesis is part of the integrated project, Scalable Computer ARChitecture (SARC), concerned with long term research in advanced computer architecture.

SARC focuses on a systematic scalable approach to systems design ranging from small energy critical embedded systems right up to large scale networked data servers.

It comes at a stage where, for the first time, we are unable to increase clock frequency at the same rate as we increase transistors on a chip. Future performance growth of computers from technology miniaturization is expected to atten out and we will no longer be able to produce systems with ever increasing performance using existing approaches.

As current methods of designing computer systems will no longer be feasible in 10-15 years time, what is needed is a new innovative approach to architecture design that scales both with advances in underlying technology and with future application domains.

This is achieved by fundamental and integrated research in scalable architecture, scalable systems software, interconnection networks and programming models each of which is necessary component in architectures 10+ years from now.

FORTH's responsibility for SARC project is to do the majority of the architectural research, some of the congestion control and all of the FPGA prototyping and NI development.

This work is part of the FPGA prototyping, relevant to cache controllers with scratchpad configurable regions (see list of contributions later in this section).

1.1 Thesis Contributions

In this thesis we present the architecture and the implementation of a node of a multicore chip with an area efficient cache controller with configurable scratchpad space.

Apart from the scratchpad(DMA) support and its functionality (presented before), the design in order to support high throughput memory accesses and power aware characteristics, includes way prediction and selective direct-mapping techniques, without losing much in performance. Way prediction techniques predict according to some algorithm the possible ways that contain the requested data without having to search all the ways of a cache. Selective direct-mapping is a technique where only a specific way is probed per cycle in order to check for a hit/miss. An efficient way prediction algorithm can lead to full utilization of the memory ports and to power reduction if the ways probed are less than the associativity way of the cache. If the ways probed in each access are close to one (on average) apart from the energy efficiency feature, our cache controller is maintaining the performance of a common cache controller.

A feature of the cache controller that has changed due to memory configurability is the replacement policy due to scratchpad regions which are not removable.

The proposed cache controller supports one outstanding miss and multiple hits under this miss. In order to be implemented the outstanding miss; a structure called Miss Status Holding Register (MSHR) is needed which holds the information about the miss request until it is finished.

A final major characteristic of the implemented cache controller is the multiple bank of cache ways support, leading in high throughput due to multiple cache accesses from processor and Network interface.

The contributions of this master thesis are:

1. Design and Implement a run-time configurable cache - scratchpad memory.

2. We also present the benefit on hardware resources of the merging of the communication subsystems required by the cache and scratchpad into one integrated Network Interface (NI) and Cache Controller (CC).

3. A high efficient way prediction algorithm and

4. Multiple other characteristics that make a cache controller performance aware.

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 refers to related work. Chapter 3 presents the architecture of the proposed design. Chapter 4 presents the implementation issues, the evaluation cost of the prototype and the experimental results. Finally, we summarize our work and draw conclusions in Chapter 5.

Chapter 2

Related Work

A significant amount of research and literature is available on the topic of memory hierarchies and especially on scratchpad memories. For example, in [35], [43] and [21] some possible architectures for embedded processors with scratchpad memories are described. Scratchpad memories, in these works, can act in cooperation with caches, either by taking the role of fast buffers for data transfer or by helping prevent cache pollution with the use of intelligent data management mechanisms.

Configurable Memory Hierarchies. Most works treated scratchpad separately from cache memory space [29], [3]. Several research groups have investigated the problem of how to best employ scratchpads in embedded systems. In these works, the major problem is not the space configurability but the data allocation on fixed size memory regions. Panda et al. [30] presented a technique for minimizing the total execution time of an embedded application by carefully partitioning of scalar and array variables used in application into off-chip DRAM (accesses through data cache) and scratchpad SRAM. This technique managed to minimize data cache conflicts, and the experiments showed improvements of 30comparable on-chip memory capacity and random partitioning strategies. In [28] Panda et al. proposed an exploration tool for determining the optimal allocation of scratchpad and data cache. Kademir et al. take compiler-driven approach, employing loop- and data - transformations to optimize the flow of data into a dynamically (software) controlled scratchpad [5]. Their memory model includes a cache, but they are more concerned with “managing the data transfers between the off-chip memory and the scratchpad”, and ignore the cache. Benini et al. take a more hardware-oriented approach, showing how to generate scratchpad memories and specialized address decoders from the application characteristics [5]. In [29] proposed an algorithm which optimally solves a mapping problem by means of Dynamic Programming applied to a synthesizable hardware architecture. The algorithm works by mapping elements of external memory to physically partitioned banks of an on-chip scratchpad memory, saving significant amounts of energy.

Apart from architectures with memory hierarchy containing both cache and scratchpad, there are some that uses only scratchpads. In [44] presented a decoupled [11] architecture of processors with a memory hierarchy of only scratchpad memories and a main memory. Each scratchpad memory level is tied with a DMA controller configured by a specific access processor which performs the data transfers between main memory, the layers of the scratchpad memory hierarchy and the register file of the Execute processor which performs data processing.

[9] and [17] present run time scratchpad management techniques, applied in uniprocessor platforms. In [9] the benefits on hiding memory latency are presented, using DMA combined with a software prefetch technique and a customized on-chip memory hierarchy mapping. The platform used in [9] contains one level of cache, one of scratchpad, a DMA controller, and a controller that controls the DMA engine. The computation of memory addresses are done by the processor that that also handles the processing data, something that means that the architecture is not decoupled as in [24]. Our memory hierarchy merges the cache with scratchpad memory space and uses

the same control mechanisms to handle cache misses and DMA transfers.

Chip Multiprocessors. Moving towards chip multiprocessors (CMP), recently with the appearance of the IBM Cell processor [10] design, which is based on addressable scratchpad memories for its synergistic processing elements, there was renewed interest for the streaming programming paradigm. Addressable local memory usefulness in general purpose CMPs was considered in some studies [36], [13]. Implementations side-by-side with caches or in cache using the sideffects of cache control bits in existing commercial processors are exploited in these studies. In [36] communication initiation (send/receive) and synchronization are considered important for high frequency streaming where transfer delay can be overlapped providing sufficient buffering. Transfer initiation is similar to ours, but data are kept in shared memory and delivered in L2 caches. Addition of dedicated receive storage (in separate address space) increases performance to that of heavyweight hardware support. Address translation hardware is not described. In [13] a scatter-gather controller at the L2 cache accesses off-chip memory and can use in-cache control bits to avoid replacements (most of the time). Cache requirements are not described, while a large number of miss status holding registers is used to exploit the full memory system bandwidth. This study exploits streaming through main memory in contrast to our L1 NI that targets flexible on-chip communication. The first study does not exploit dynamic sharing of caching space with addressable space explored in our L1 NI design. The second does not describe how to implement it and does not consider hardware support for queues. In contrast to our design for scalable CMPs, theses studies target a bus based system.

Network Interface Placement. Network interface (NI) placement in the memory hierarchy has been explored in the past. The Alewife multiprocessor [23] explored an NI design on the L1 cache bus to exploit its efficiency for both coherent shared memory and message passing traffic. The mechanisms developed exploited NI resource overflow only to main memory, which was adequate with 90's smaller processor-memory speed gap and loosely-coupled systems. At about the same time, the Flash multiprocessor [14] was designed with the NI on the memory bus for the same purposes. Cost effectiveness of NI placement was evaluated assessing the efficiency of interprocessor communication (IPC) mechanisms. One of the research efforts within the Flash project explored block-transfer IPC implemented on theMAGIC controller (the NI and directory controller in Flash) and found very limited performance benefits for shared memory parallel workloads [46]. Later in the 90's, Mukherjee et al. [26] demonstrated highly efficient messaging IPC with a processor cacheable buffer of a coherent NI on the memory bus of a multiprocessor node, The memory bus placed NI was (and is) less intrusive and thus easier to implement than the top performing cache-bus placed NI of that study whose advantage faded in the evaluated applications. This body of research in the early 90's explored less tightly-coupled systems than those of today and far less than future many-core CMPs. More recent scalable multiprocessors that use the network primarily for IPC, like Alpha 21364 systems [25], Blue Gene/L [39] and SiCortex systems [16], adopt the latter placement of the network interface on the memory bus, but also bring it on-chip. The same is done in contemporary chips targeted for the server market like Opteron-based AMD CMPs and SUN Niagara I and II processors.

Our NI takes this trend one step further to investigate cache-to-cache IPC free from off-chip main memory and/or directory controller side effect traffic for future highly integrated and scalable systems. To achieve this goal we provide novel hardware support for efficient and fully-virtualized software control of nearly all cache resources and mechanisms.

Coherent Shared Memory Optimizations. Another body of research in the 90's explored optimizations of coherence protocols for producer-consumer communication patterns [7] as well as combination of producer-initiated communication with consumer prefetching [31], [1], [20]. These studies report that prefetching alone provides good average performance while push-type mechanisms may provide an additive advantage. Streamline [6], an L2 cache-based message passing mechanism, is reported as the best performing in applications with regular communication patterns among a large collection of implicit and explicit mechanisms in [7]. The limited promise of message passing mechanisms in the loosely coupled systems studied in the past, has led to the widespread use of hardware prefetchers. In addition, it contributes to the current trend of restricting NIs to off-chip communication, that has prevented the use of explicit communication in fine-grain applications. In the flip side, the advent of CMPs and the abundance of transistors in current and future integration processes raises today a concern about their use and programmability for general purpose applications. Our NI design addresses these concerns in two ways. First, by providing support for non shared memory programming models and paradigms to exploit large CMPs and basic explicit communication mechanisms.

Way Prediction to Reduce Cost, Power and Occupancy. To improve the cost effectiveness of associative caches several techniques that rely on sequential access to narrow memory blocks have been proposed. The narrow memory blocks are area efficient and the tag check circuitry -and energy- is reduced. To mitigate the increased hit latency caused by the sequential access, way-prediction [8], [47] and Selective Direct-Mapping [4] have been proposed. These techniques attempt to predict the matching way of the cache, so the hit can be serviced with a single access in the common case, achieving average performance very close to that of a cache with fully parallel access to all the ways. Our way prediction adds a look-up of all-set combined tag signature in parallel to previous cache level access expecting that most of the time only one will match for a good signature (e.g. CRC). This will allow not only cost reduction with minimum performance penalty, but also power savings and reduced occupancy of the cache since the processor will occupy at most one bank per cycle leaving other banks available for incoming and outgoing network processing.

Replacement policy. Cache replacement algorithms to reduce the number of misses have been extensively studies in the past. Many replacement policies have been proposed, but only a few of them are widely adopted in caches such as random, true LRU (Least Recently Used), and Pseudo LRU. Random policy chooses a random cache line for eviction without a specific algorithm. The randomness emerges from, let say, the number of clock cycle. LRU policy exploits the principle of temporal locality and evicts the cache line which has not been used for the longest time. Apart from local replacement policies there are some proposed policies which adapt to the application behavior, but within a single cache. For instance, Qureshi et .al propose retaining some fraction of the working set in the cache so that some fraction of the working set contribute to cache hits [33]. They do this by bringing the incoming block in the LRU position instead MRU (Most Recently Used), thus reducing cache thrashing. Another adaptive replacement policy is presented in [42], where the cache switches between two different replacement policies based on the behavior.

Some current microprocessors have cache management instructions that can flush or clean a given cache line, prefetch a line or zero out a given line [45], [34]. Other processors permit cache line locking within the cache, essentially removing those cache lines as candidates to be replaced [37], [2]. Explicit cache management mechanisms have been introduced into certain processor instruction sets, giving those processors the ability to limit pollution. In [42] the use of cache line locking and release instructions is suggested based on the frequency of usage of the elements in the cache lines. In [12], policies in the range of LRU and Least Frequently Used are discussed (LFU). Our research field is not close to replacement policy field, but the merging of scratchpad space in the cache; adapt the replacement policy in a way not to evict the scratchpad (locked) regions.

In this master thesis a configured replacement policy is used due to the existence of not evictable memory regions in the second level of memory hierarchy.

Miss Status Holding Registers. The Miss Handling Architecture (MHA) is the logic needed to support outstanding misses in a cache. Kroft [22] proposed the first MHA that enabled a lock-up free cache (one that does not block on a miss) and supported multiple outstanding misses at a time. To support a miss, he introduced a Miss Information/Status Holding Register (MSHR). An MSHR stored the address requested and the request size and type, together with other information. Kroft organized the MSHRs into an MSHR file accessed after the L1 detects a miss. He also described how a store miss buffers its data so that it can be forwarded to a subsequent load miss before the full line is obtained from main memory. Scheurich and Dubois [38] described an MHA for lock-up free caches in multiprocessors. Later, Sohi and Franklin [41] evaluated the bandwidth advantages of using cache banking in non-blocking caches. They used a design where each cache bank has its own MSHR file, but did not discuss the MSHR itself. In this work we support one outstanding miss through the use of one MSHR.

The proposed work supports one outstanding miss so the number of MSHR is only one and its structure is as simple as possible.

Chapter 3

Architecture

In this chapter is presented a detailed description of the architecture of the whole system. Choices made and decisions taken about the cache controller, the scratchpad configurability and the Network interface are presented and discussed.

3.1 Overall Diagrams and Operation

A very abstract architectural organization of the system can be seen in Figure 3.1. It is shown in the left side, the processor which can be a general-or special-purpose or an accelerator. The two hierarchies of cache memory are assuming to be private. The Network Interface (NI) is assuming to be operating at the level of L2 cache. An issue of major significance is where to place (in L1 or L2 memory hierarchy) scratchpad memory and

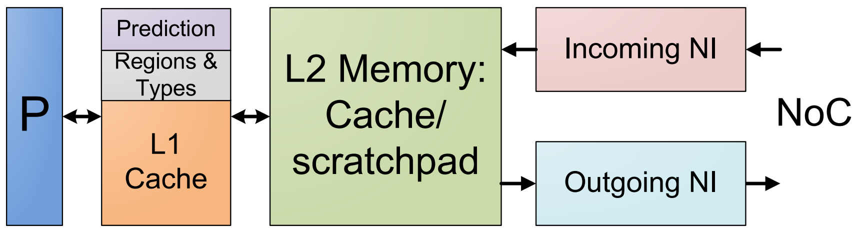

FIGURE 3.1: Overall (abstract) block diagram

the NI functionality in the design. Scratchpad memory and NI are tightly coupled due to the functionality the NI supports to scratchpad in order to be accessed. So the placement of the one influences the other. The decision taken was in favor of the L2 level cache. L1 caches are usually quite small (on the order of 16 KBytes), of narrow associativity, they need to be very fast, and they are tightly integrated into the processor datapath. Interfacing NI functionality so close to the datapath and so tightly coupled to the processor clock cycle would be difficult. During most clock cycles, the processor accesses most or all SRAM blocks that form L1 caches, thus leaving little excess bandwidth for NI traffic. Dedicating a part of the -about- 16 KBytes to NI buffers, data structures, and tables would be extremely tight. Allocating another part of the -about- 16 KBytes as scratchpad memory would be nearly useless, since many modern applications require scratchpad areas in the hundreds of KBytes.

By contrast, at the L2 level, a processor core will usually be surrounded by several SRAM blocks (layout considerations suggest numbers around 8 or more such blocks); these can often be organized as wide-associativity L2 cache banks. SRAM block sizes at this level range around 32 or 64 KBytes each, so we are talking about a local memory on the order of 256 KBytes to 1 MBytes; scratchpad and NI areas can be comfortably allocated out of such sizes. Processor access time to this second level of the memory hierarchy is a few clock cycles (e.g. 4 to 8), and there is a danger for this increased latency to adversely affect NI performance. However, the challenge that we successfully faced and resolved is to implement the frequent NI operations so that the processor can initiate them using pipelined (store) accesses that proceed at the rate of one per clock cycle.

A table near L1 cache is the way prediction table (Prediction in Figure 3.1) which supports information about the possible "hit" ways of L2 cache in order to minimize the possible L2 way accesses of the processor, leaving plenty of excess bandwidth for use by the NI and save in power. More details about this structure will be discussed in 3.3.2

3.1.1 Regions & Types Table (R&T)

Another structure-table near L1 is the Regions and Types (R&T) table which keeps information relative to permissions. Address region information is needed not only to check load/store instruction permissions, but also to provide caching policy and locality information for use by the local memories and the network interface. While a single table can provide all this information, it is likely that systems will want to use multiple tables, often containing copies of the same information, so that the processor and the NI can access them in parallel. We call these tables Regions and Types (R&T) tables, and their structure can be as illustrated in Figure 3.2. We expect R&Ts to be quite small -say 8 to 16 entries. While TLBs are just caches containing the most recently used entries of the (quite large) page translation tables, we expect R&Ts to usually fully contain all the region information relevant to their purpose. The big difference between the two structures is as follows. Physical pages end up being scattered all over physical address space, thus page tables contain a large number of unrelated entries. On the contrary, in virtual address space, applications use a small number of address regions: each of them may potentially be large, but they each consist of contiguous addresses; thus, each such region can be defined using a single pair of “base-bound” registers (or a single ternary-CAM entry, if its size is a power of 2). For example, a typical protection domain may consist of the following set of address regions: (i) one contiguous (in virtual address space) region for the code of application; (ii) a couple of regions for (dynamically linked) library codes; (iii) a private data and heap address region; (iv) a (private) stack region; (v) a “scratchpad memory” region (locations that are locked in the local memories of the participating processors, so that the coherence protocol cannot evict them from their “home”); and (vi) a couple

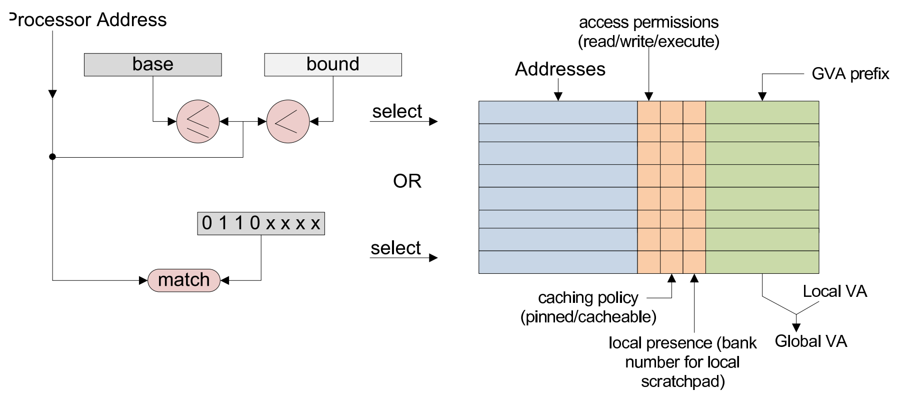
Processor Address
base
bound
select
OR
0 1 1 0 x x x x
select
match
access permissions (read/write/execute)
Addresses
GVA prefix
caching policy (pinned/cacheable)
local presence (bank number for local scratchpad)
Local VA
Global VA

FIGURE 3.2: Regions and Types table: address check (left); possible contents(right)

of shared regions, each of them shared e.g. with a different set of other domains.

Address regions are of variable and potentially large size. Thus, the tables describing them, R&T, cannot be structured like traditional page tables. Instead, each R&T entry must contain a base and a bound address defining the corresponding region; at run-time, each address must be compared against all such boundary values, as illustrated in the top left part of Figure 3.2. The comparisons involved can be simplified if region sizes are powers of 2, and if regions are aligned on natural boundaries for their size; in this case, ternary CAM (content-addressable memory) -i.e. CAM that allows don't care bits in its patterns- suffices for implementing the R&Ts, as illustrated in the bottom left part of the Figure 3.2.

The protection purposes discussed earlier in this section can be served by an R&T placed next to the processor, as shown in Figure 3.1, and containing, for each entry, a couple of bits specifying read/write/execute permissions; access to absent regions is forbidden. However, it is convenient to add more information to this R&T, to be used when accessing the local memories. The most important such information determines if the relevant region corresponds to local scratchpad memory (non-evictable from local memory), and if so in which local SRAM bank.

The cache controller may also need to know e.g. which remote scratchpad regions are locally cacheable (non-exclusively), and which ones are non-cacheable. Besides adding a few bits to each entry, such information is also likely to increase the number of entries in the R&T, because the global scratchpad region must now be broken down into finer-granularity portions, distinct for local and for remote scratchpad regions. We expect that an R&T of size 12 to 16 entries will suffice for both purposes. The right part of the R&T shown in Figure 3.2, labeled GVA prefix, illustrates another possibility for using region tables. A large system is likely to need wide global virtual addresses (GVA) -say 48- to 64-bit wide. Yet, within such systems, there may be several accelerator engines or processors that use narrow datapaths -e.g. 32-bit or even narrower. How can a processor or engine that generates narrow (virtual) addresses talk to a system that communicates via wide global virtual addresses? The GVA prefix part of the ART in Figure 3.2 provides the answer. Say that the processor generates 32-bit local virtual addresses (VA), and this processor is to be placed in a system using 48-bit GVAs. The 8-entry R&T of Figure 3.2 allows this processor to access up to 8 regions anywhere within the 256 TBytes GVA space, provided that the sum of their sizes does not exceed 4 GBytes. The idea is as follows: the processor can generate 4 Giga different 32-bit addresses; the R&T prepends 16 bits to each of them, thus mapping it (almost) anywhere in the 256 TBytes GVA space; eight different mapping functions exist, one per region defined in the R&T, each function being a linear mapping (fixed offset addition, simplified as bit concatenation owing to alignment restrictions). The description and presentation of the R&T is chosen in this part of the chapter because it is critical to know its functionality and usage for the following sections.

3.1.2 Integrated Scratchpad and Cache

Scratchpad space corresponds to cache lines that are pinned (locked) in the cache, i.e. cache line replacement is not allowed to evict (replace) them. The address of a scratchpad word must be compatible with the cache line where that word is allocated, i.e. the LS bits of the address must coincide with the cache line index; this minimizes address multiplexing, hence speeds up accesses. For proper cache operation -whenever some part of the local memory is allocated as cache- at least one of the ways of this local memory should contain no scratchpad regions, but it is programmer decision.

L2 cache lines can be configured as scratchpad, i.e. locked into local memory, using either of two ways: a lock bit can be set in the line's tag, or the processor's R&T table may specify that certain addresses correspond to scratchpad memory and are mapped to this specific cache line (and in this specific way bank). The latter method consumes an R&T entry, but offers two advantages: any number of contiguous cache lines, up to a full L2 cache way, can be mass-configured as scratchpad using a single R&T entry; and the tag values of these R&T-configured scratchpad lines become irrelevant for proper cache/scratchpad operation, and are thus available to be used for other purposes. More details of scratchpad allocation and configuration are presented in section 3.4.

3.1.3 Incoming - Outgoing Network Interface

Even the Network Interfaces are outside the scope of this master thesis, it is critical for the presentation of this work to mention some characteristics and functionality to understand how the whole system works and how these interfaces influence it. In section 3.5 are presented some remarkable features that help the flow of the presentation.

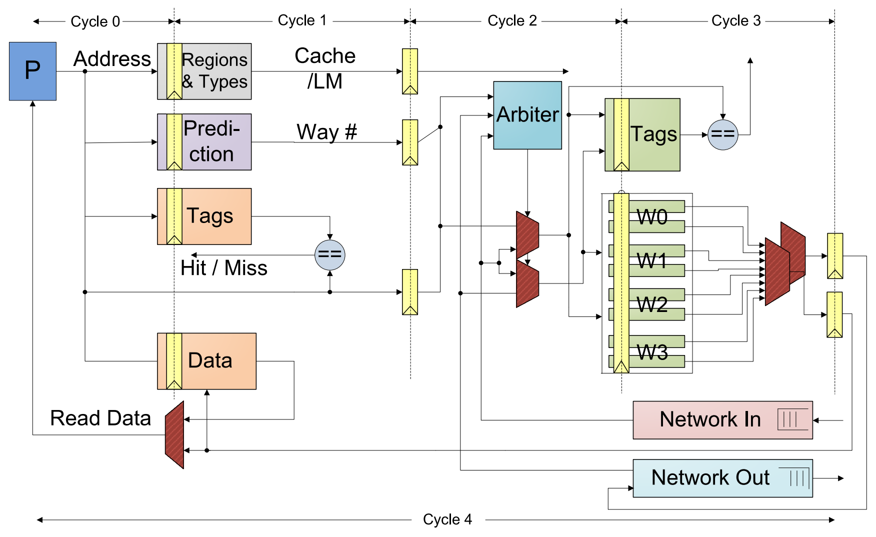

FIGURE 3.3: Read Data Path - detailed Block diagram, timing information

3.2 Architecture Diagrams

3.2.1 Detailed diagrams - Datapaths

Figures 3.3 and 3.4 even that we are in architecture chapter are drawn with a rough notion of implementation issues and of time running horizontally, from left to right. They contain a lot of implementation characteristics but they are presented here, in order to describe in an understandable way the architecture of the proposed design. The processor issues memory requests to the first level of its memory hierarchy; these often hit in L1, and responses come back within 1 clock cycle. The R&T table is accessed in parallel with L1 caches, to provide access permission information and the way the data resides if the region is scratchpad. For accesses to non-scratchpad regions that miss in L1, one may consider providing L2 way-prediction (e.g. read and match narrow signatures of all L2 tags) in parallel with or very soon after L1 access. Load or store instructions that miss in L1, or that write through to L2, or that have to bypass L1, are directed to the "L2" local SRAM blocks, presumably with a knowledge of the particular bank (way)

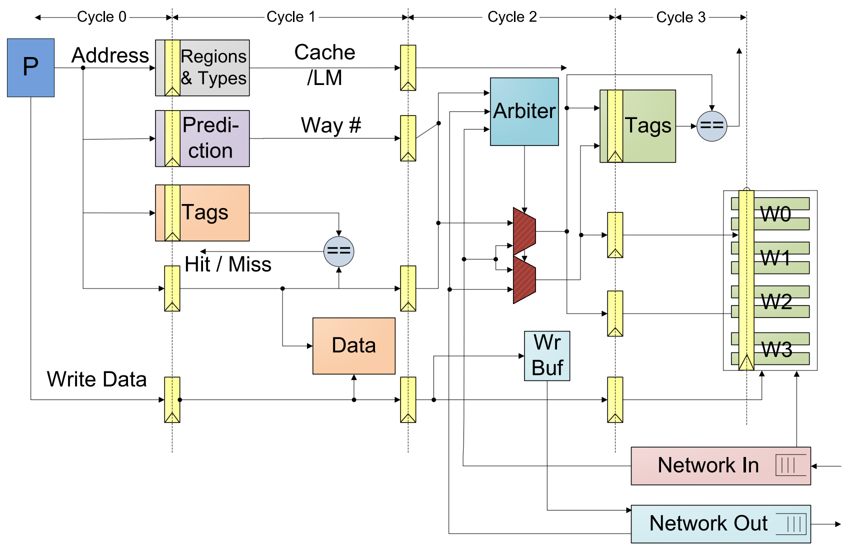

FIGURE 3.4: Write Data Path - detailed Block diagram, timing information

that is targeted. This "L2" local memory consists of multiple interleaved banks, and which are shared between the processor on one hand, and the network interface (NI) and cache controller (CC) on the other hand. These two agents compete for accesses to the L2 banks (ways); owing to wide (e.g. 8-way) interleaving, L2 memory should provide sufficient throughput to serve both of them.

Network Out and Network In are the interfaces to the Network in cases of DMAs and cache miss services through the same functionalities (integration of controllers). This issue is outside the scope of this work even though in section 3.5 will be discussed some details.

The L2 memory space is dynamically shared (through arbiter) among three functions: cache controller (processor side); incoming NI; and outgoing NI. The space allocation among the three functions can be changed at run time; we consider this to be an important feature of new NIs, by contrast to old NI architectures that used dedicated NI memory, thus leading to frequent underutilization of either the NI or the processor memory, and to costly data copying between the two.

One major difference between Figures 3.3 and 3.4 is the presence of a module called Write Buffer (Wr Buf in the Figure 3.4). It is combined with the outgoing Network Interface adding performance to packet generation in order to start transmission to the network as fast as possible. For more details see section 4.4.

3.2.2 Detailed diagrams - Timing

Although in Figures 3.3 and 3.4 have presented a timing aspect of the architecture, in Figure 3.5 have been removed all the functionalities that will beguile us and only timing has remained.

According to architecture and the timing diagram of Figure 3.5, L1 cache, Way Prediction and R&T tables accessed in parallel and respond in one cycle, taking decisions about L1 hit/miss, the possible L2 "hit" ways, the region and the type of memory the requested data may exist. Between L1 and L2 level caches there is a pipeline register making the path between them shorter and leaving L2 arbitration in a separate clock cycle. Arbitration and BRAM port driving (de-multiplexing to all BRAM ports) is the 3rd cycle and multiplexing from all BRAMs to L2 output and the response to processor are the last two cycles.

3.3 Caching functionality

Having discussed the overall architecture we now step inside in more details about the architecture. In this section it is presented the L2 cache controller. The major features of its organization and functionality are the following.

3.3.1 Sequential Tag Array

One of our main goals is to achieve configurability of the local SRAM space, and be able to use it either as cache or as scratchpad at both coarse and fine granularities. We would also like to use the same memory blocks for cache and scratchpad regions. The parallel nature of set-associative caches results

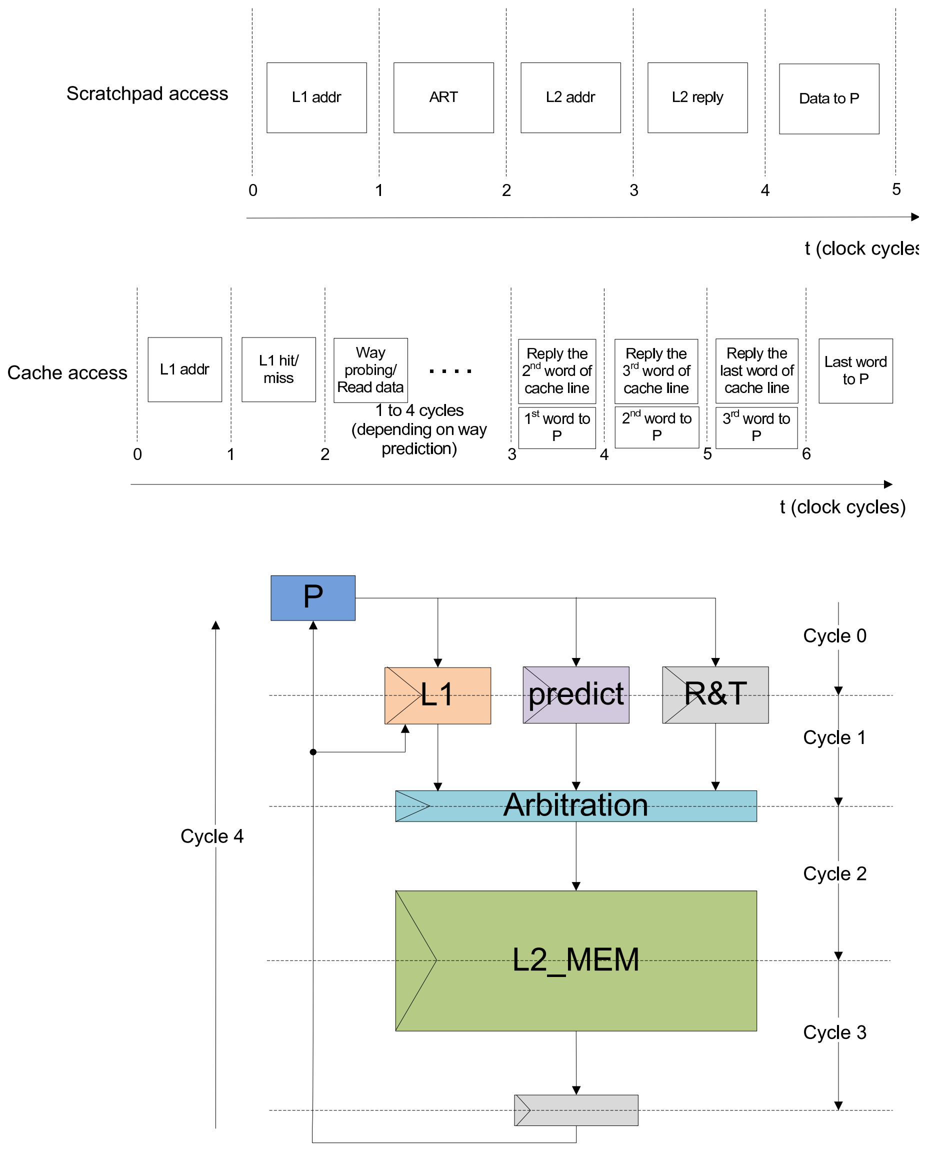

FIGURE 3.5: Timing Diagram

in a requirement for simultaneous access to memories that implement the cache ways (tags + data) so as to determine quickly the hit or miss and serve the issued request. To support both cache and scratchpad in coarse grain reconfiguration, one natural and convenient way is to utilize multiple memory banks and configure them either as cache ways or as local scratchpad memory. While this approach is straightforward for the data arrays of the cache, the tag arrays are of different total size (and in most cases of different dimensions) than the data array. The common caches consist of small, fast memories containing tags and larger containing data (Figure 3.6). This uniformity of memory block arrangement in case of scratchpad allocation, preclude the tag memory utilization leaving the tag memory blocks unused. From scratchpad point of view, the tag memory blocks can be fully utilized if their size is equivalent to the data memory blocks (Figure 3.7). This approach is straightforward for the scratchpad to use all memory bits, but the cache is forced to use larger (and hence slower) memory blocks for the tag arrays. The optimized memory block organization that conforms to both cache and scratchpad memory arrangements is depicted in Figure 3.8. All memory banks are equally sized and many (possibly all) of tags memory blocks are placed together in a single array. This organization saves considerable space since the utilization of the tag array is greatly improved, but disallows the parallel access in the tags for all the banks, forcing a sequential check of the tags. This limitation increases both hit and miss latencies and affects the performance.

The sequential access approach increases cache access time but it can prove very effective in reducing the energy dissipation in embedded systems and balance the performance-energy tradeoff.

3.3.2 Way Prediction

In sequential access, a 4-way-associative cache accesses one tag way after the other (in parallel with the data way) until someone hits or even the fourth

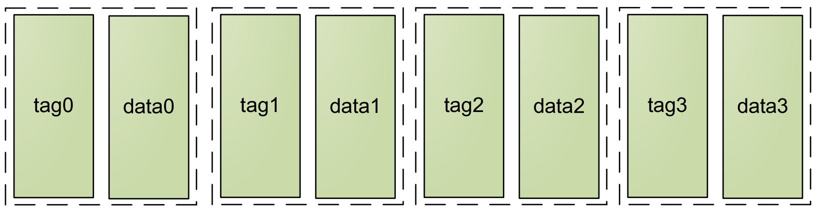

FIGURE 3.6: Common tag organization

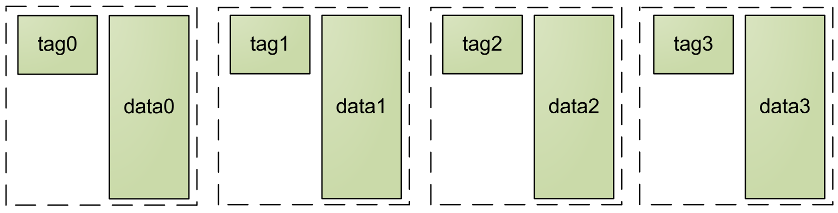

FIGURE 3.7: Scratchpad optimized tag organization

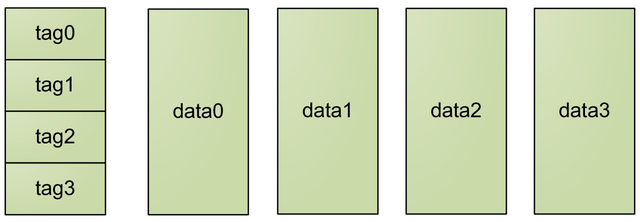

FIGURE 3.8: Proposed tag organization

one misses. Phased caches [19] wait until the tag array determines the matching way, and then accesses only the matching way of the data array, dissipating about 75% less energy than a parallel access cache. Sequential access, however, serializes the tag and data arrays, adding as much as 60% to the cache access time [32]. If a tag memory block takes 1 cycle to be accessed, an eight way associative cache will take 8 cycles to detect a miss. Here arises the dilemma to decide between performance and memory block utilization due to scratchpad characteristic. Schemes like Way-Prediction [32], [8], [47] and Selective Direct-Mapping [4] go one step further, by predicting the way that will likely hit, and thus minimize the wasted energy. While most schemes try to use and benefit from PC values to predict a way, the way prediction model we propose uses "signatures" of the address tags. The signature of the proposed design derives from the bitwise XOR of the three most significant bytes of the address of the request. These two popular schemes represent the two extremes of the trade-off between prediction accuracy and early availability in the pipeline. The PC is available much earlier than the XOR approximation but the XOR approximation is more accurate. Instead of probing all tags in parallel and comparing against the MSBs of the address, or use prediction tables, we use an SRAM block to store representative signatures - a few address bits or a simple checksum with uniform properties - from every address tag. The comparison of the signatures with the actual address gives an indication for a specific way that is highly possible to produce a hit, and then we have to probe the way to verify that it is not a false positive hit. We have to carefully select the signature format so as no false negatives occur - Bloom Filters' properties fit very well. The way prediction module executes in parallel with L1 tag matching losing no cycles to generate the way prediction mask.

Moreover, the proposed method reduces the required SRAM bank throughput from the processor side allowing the NI to efficiently share the memory throughput. This scheme with the memory blocks organization we adopted (Figure 3.8) gain both in configurability and efficiency, as well as in power consumption. As for the dissipation, it is likely that the signatures - accessed in every memory request - are placed in a small SRAM block that will be more energy efficient.

3.3.3 Replacement Policy

The design of an efficient cache hierarchy is anything but a trivial issue. Cache replacement policy is one of the pivotal design decisions of any memory hierarchy. Replacement policy affects the overall performance of the cache, not only in terms of hits and misses, but also in terms of bandwidth utilization and response time. Another cache hierarchy decision is the set associativity, which offers good balance between hit rates and implementation cost. Associativity and replacement policy are strictly related characteristics. The higher the associativity of the cache, the more vital the replacement policy becomes.

The replacement policy, in common cache architectures, must select according to an algorithm between a static number of cache lines which is the number of way associativity. As the associativity number increases the possible replaceable cache lines increase, so the algorithm becomes more complex.

In our design things concerning replacement algorithm are more complex than common designs. As mentioned earlier scratchpad areas are not removable. In case the replacement policy chooses a pinned (locked) cache line for eviction, it has to change its decision and choose a cacheable block of the same index. The complexity has to do with the possible replaceable cache lines of a set which are not fixed because scratchpad can be any block in memory. Lock bit is a way that a cacheable block is distinguished from scratchpad one. So before the replacement policy starts executing the lock bits of the set must be checked first, in order to exclude the scratchpad blocks from the possible replaceable cache lines.

3.3.4 Bank Interleaving

One of the major performance features of the unified cache - scratchpad memory is the multi-banking. It is an extensively used feature in high bandwidth memory subsystems when accessed by several agents (processors, accelerators, NIs, e.t.c.). To allow simultaneous memory access by a number of processors, such memory subsystems consist of multiple banks that can be independently addressed. Memory blocks are typically interleaved with low-order bits of the address over the available memory banks in order to minimize bank conflicts. In order to avoid conflicts the simultaneous references must address different banks. These multi-banking, interleaved concepts are recently applied to high bandwidth, multi-ported caches.

Multi-ported caches apart from interleaving are also implemented in one of other three ways: either by conventional and costly ideal multiporting, by time division multiplexing, or by replicating multiple single-port copies of the cache. Conceptually, ideal multi-porting requires that all p-ports of a p-ported cache be able to operate independently, allowing up to p cache accesses per cycle to any addresses. However ideal multiporting is generally considered too costly and impractical for commercial implementation for anything larger than a register file.

The time division multiplexed technique (virtual multiporting) employed in the DEC Alpha21264 [40], achieves dual-porting by running the cache SRAM at twice the speed of the processor clock. In this thesis we do not explore this technique.

The data cache implementation in the DEC Alpha 21264 [40] provides an example of multiport through multiple copy replication. This architecture keeps two identical copies of a data set in each cache. A drawback of this technique is to keep both copies coherent. Another major cost is the area duplication in the die. In this thesis we do not explore this technique.

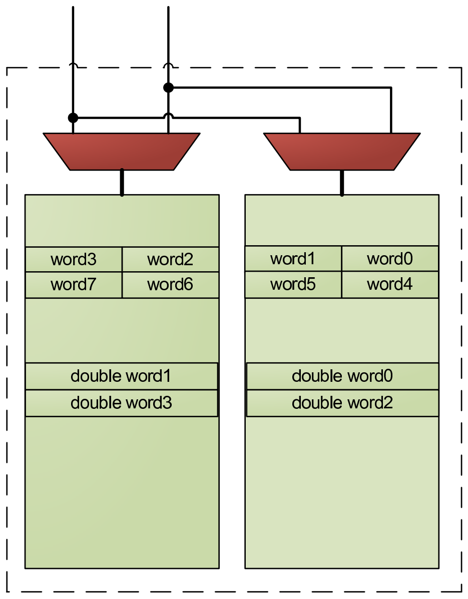

FIGURE 3.9: Bank interleaving in word and double alignment

In the proposed architecture 2-bank interleaving is supported as shown in Figures 3.3 and 3.4. These two banks are shared between the processor, the outgoing and the incoming NI. The imbalance scheduling between the chosen bank will increase the bank conflicts (3 agents, 2 banks) and degrade the delivered performance. The scheduling is performed by an arbiter who chooses the agent that will access each bank, each cycle.

In Figure 3.9 it is shown that each way is consisted by two banks, where in the left bank are placed the odd double words of a cache line and the right the even ones. Each bank is one double-word (64 bits) width. The concurrent access of the two banks maximize the possible throughput to 128bits/cycle if both banks are accessed concurrently even if the two agents - processor and network interface - that access the memory banks are 32 bits.

3.3.5 Multiple Hits under Single Outstanding Miss

For pipeline computers that allow out-of-order completion, the processors need not to stall on a cache miss. The processor will continue fetching instructions while waiting cache to return the missing data. In the current design the processor is not out-of-order so when a load is issued the processor stalls until the data return. In cases of store instructions the processor need not to wait on something. So when a store miss occurs, the processor does not stall and continues fetches instructions from instruction memory. This cache feature escalates the potential benefits of outstanding misses by allowing the data cache to continue supply cache hits during a miss. This outstanding miss (or multiple hits under one miss) optimization reduces the effective miss penalty by being helpful during a store miss instead of ignoring the requests of the processor. In case of miss under miss the cache stalls waiting the first miss to complete. The proposed cache controller supports one outstanding miss something reduces the stall memory time about 40-60% according the SPEC92 benchmarks [15]. In order to support the above characteristics, nonblocking (lockup-free) caches require special hardware resources called Miss Status Holding Registers (MSHR). These registers are used to hold information on outstanding misses. These structures are fully associative and bypassed. It is beyond the scope of this work to discuss in details MSHR. Due to the one outstanding miss supported by cache controller, the organization of the MSHR is very simple, holding the address and the data, in word granularity, of the outstanding store miss. In case of an access in the same cache line that refers the current outstanding miss, a tag bit named pending is set and the pipeline stalls in order to serve the miss and after the request in the same cache line. This bit serves the occasion where two back to back requests, refer to the same missing cache line. Without the functionality the Pending bit offers, if the first access is a miss and the second will be also, the pipeline stalls. When the first miss returns, the second starts to be served as a miss even the data exist in L2 cache. In other implementations tag matching will executed again in order to decide that we have a hit. With Pending bit functionality we gain in time, not needing to match the tags again or to serve a miss in a present cache line, losing the data stored before (due to write back property).

3.3.6 Deferred Writes

Another feature of the cache controller is the deferred writes. Due to the pipeline and as observed in the Figure 3.10 the final cycle of a store hit is overlapped with the first cycle of a load. As shown in the Figure 3.10, during cycle 1 the store access write the data in the cache, while the next request (load) reads the data banks in parallel with tags (no phased cache). In this cycle both requests try to access the same port of the memory blocks. One solution is to delay one cycle the load access or to delay the store completion. The choice that was taken is the second and that is the reason why this cache controller is load optimized. The store request is postponed and the special buffer holds the address and the data o the store request. The buffer is flushed when memory block port is idle. When the buffer is valid and a load request in the related cache line issued, the data are bypassed. In the occasion of a write back of the related cache line, the buffer is first flushed to memory banks and then the write back is served.

3.3.7 Access Pipelining

After presenting the major characteristics of the architecture of the cache controller, we show the pipeline of a common access. According to the timing information presented in Figures 3.3, 3.4 and 3.5, Figure 3.11 shows in more details the different cycles of a load request (for example). Something that did not have shown in the previous Figures is the way probing (cycles 2 to 5 at most) where every predicted way is probed sequentially until a hit or a miss happens. When a hit happens, the whole cache line must be read

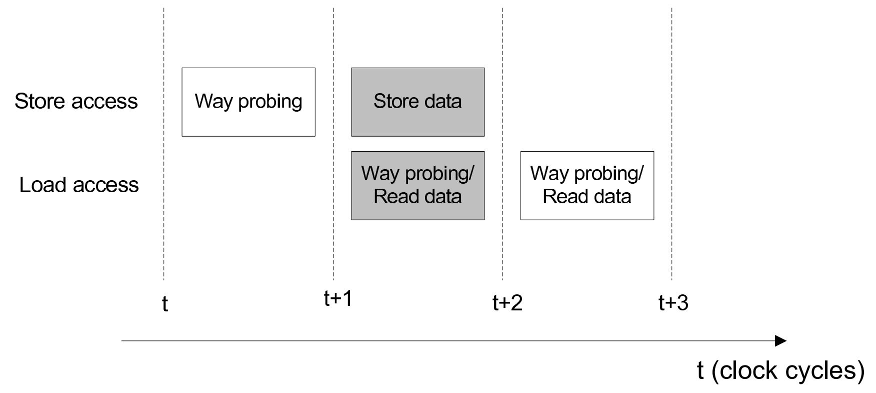

FIGURE 3.10: L2 memory 2 stage pipeline

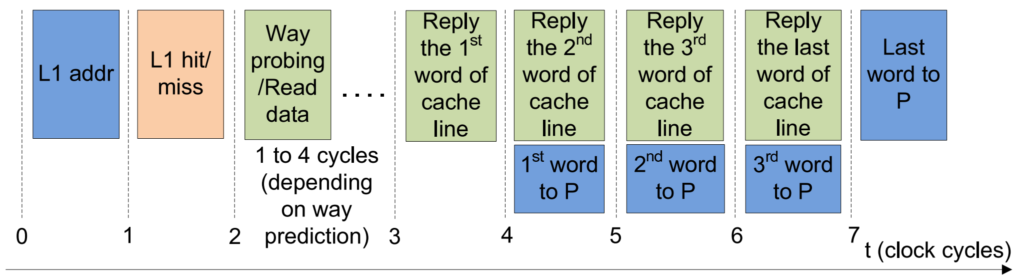
L1 addr
L1 hit/ miss
Way probing /Read data
1 to 4 cycles (depending on way prediction)
Reply the 1st word of cache line
Reply the 2nd word of cache line
Reply the 3rd word of cache line
Reply the last word of cache line
Last word to P
1st word to P
2nd word to P
3rd word to P
0
1
2
3
4
5
6
7
t (clock cycles)

FIGURE 3.11: Cache memory access pipeline (L1 miss, L2 hit)

which means some extra cycles according to the width of the path between L2 and L1 and the size of the cache line. The second level of cache hierarchy is designed to have 2 stage pipeline. The first for tag matching and memory bank driving and the other for memory bank response. Taking this into account, store hit accesses coming the one after the other are served back to back, something that cannot be happen in cases of stores and loads without the support of deferred writes presented in the section before.

3.4 Scratchpad functionality

Some real-time applications - especially in the embedded domain - seek for predictable performance that caches cannot guarantee. Instead, local

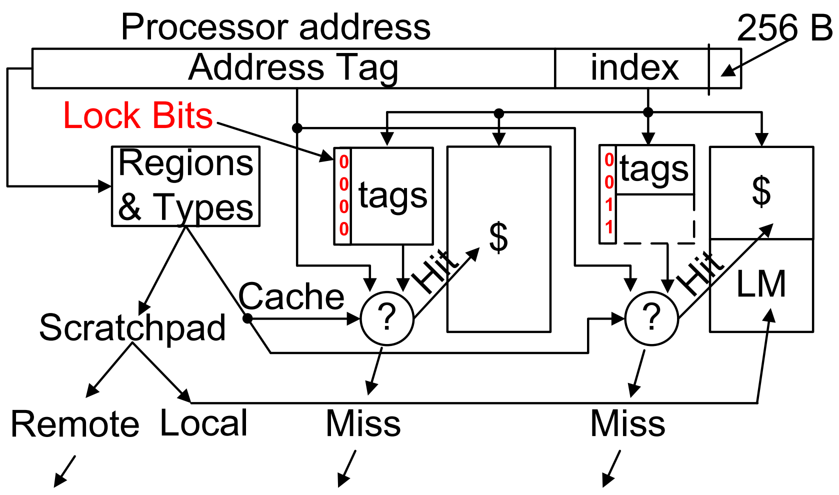

FIGURE 3.12: Abstract memory access flow

scratchpad memories can be used so that the programmer can reason about the access cost and the system performance. Coarse grain configurability at the granularity of the cache data memory module is relatively easy to support but may prove inflexible for all cases. To provide more flexibility in the system, besides the coarse grain configurability we provide support for fine grain configurability, at the granularity of the cache block.

Scratchpad functionality is influenced by the R&T table which distinguishes the scratchpad from cacheable area. In an abstract architecture as Figure 3.12 shows, the tag matching is valid only if the R&T table detects a cacheable area access. In case of hit the cache data are accessed otherwise the next level of cache triggered. When a scratchpad request detected the only thing that matters is if it is Local so the scratchpad data are accessed or Remote so a communication with a remote scratchpad is initiated. Scratchpad regions are of 256Bytes granularity and that is why the least significant bits of address (Figure 3.12) are ignored.

We propose a special bit that can used to "pin" specific cache-lines and prevent their eviction from the cache controller. Therefore apart from the

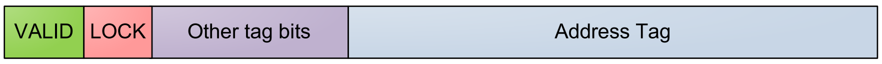

FIGURE 3.13: Tag contents

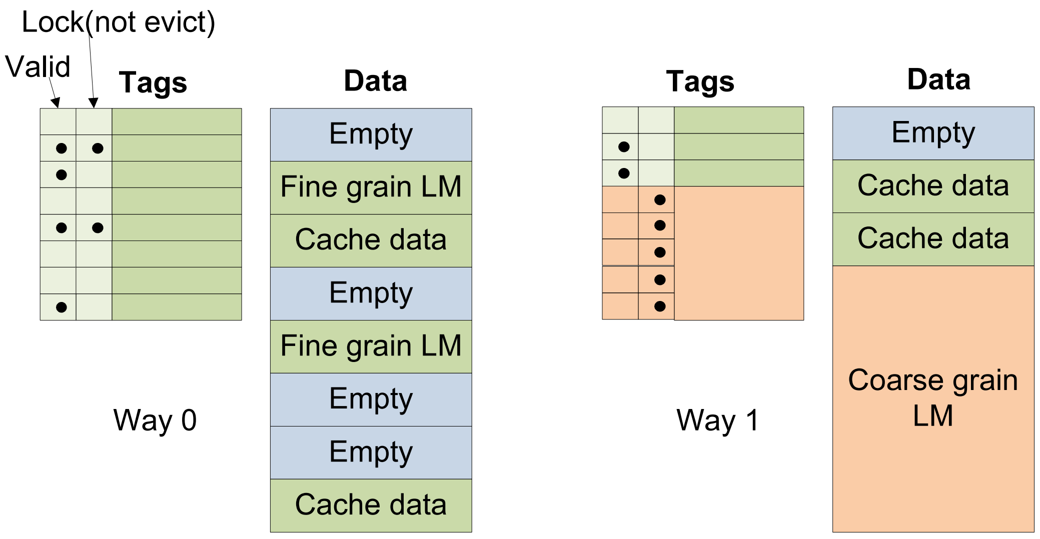

FIGURE 3.14: Local memory configurability

required the tag fields (Figure 3.13): valid bit, address tag and other tag bits including coherence bits and the pending bit mentioned in 3.3.5, we add an extra lock bit.

The use of the "lock bit" allows us to pin several independent cache-lines and since no eviction will ever occur, these cache-lines emulate scratchpad behavior. The valid bits of these cache lines are unset, so no tag matching is performed for the access of them. The application may use a special instruction or a special address range that provides access to the tags, similar to scratchpad, to pin specific cache-lines. Such a feature, illustrated in Figure 3.14, allows small fragmented scratchpad areas that use the normal cache access behavior - tag comparison - and have predictable access time; a long-term use of a cache-line.

Scratchpad areas can also be configured at coarser granularities (e.g.

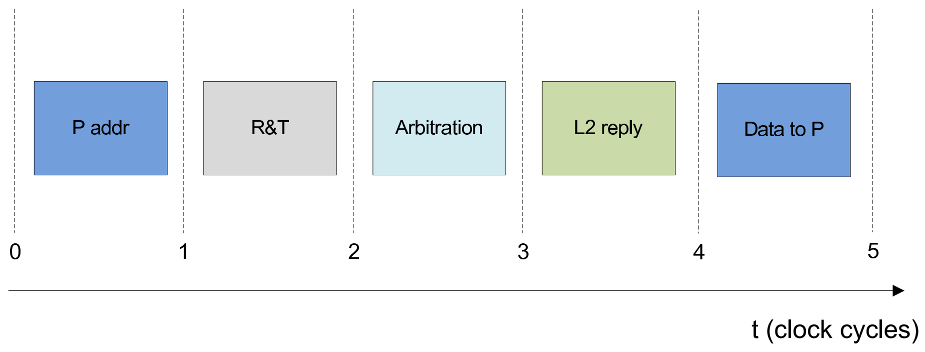

FIGURE 3.15: Pipeline of a scratchpad access

large contiguous blocks) using the R&T Table. The R&T Table compares every requested memory address against a set of regions to ensure protection but also provides configuration information. A couple of regions can be marked as block scratchpad regions and so the cache controller omits the tag comparison step as shown in Figure 3.12, and accesses directly the specific way provided by the R&T in index equal with the LSBs of the address. The cache-lines configured as block scratchpad should be handled by the cache controller as invalid and locked, marked in tags, so as not be used by the tag matching mechanism and not to be replaced, Figure 3.14. The rest of the tag bits, in the latter cache-lines, are used for other functions by the NI, such as network buffer meta-data.

3.4.1 Access Pipelining

One of the innovations of the current work is the fast access of the scratchpad memory. As previous timing diagrams (Figures 3.3, 3.4) and Figure 3.15 show, a scratchpad access lasts 5 cycles. One cycle takes the R&T table access, one cycle the arbitration and scratchpad addressing. BRAMS take one cycle to respond the data and a final cycle takes the transfer to processor.

3.4.2 Scratchpad Request Types

The possible scratchpad requests are the following:

Scratchpad data read and write: common scratchpad region accesses which last 4 cycles from the moment the processor issues the request to L1 cache, as discussed above. The address is generated from the processor and the scratchpad selected way is an information taken from the R&T table.

Scratchpad tag read: in this case the processor or the NI wants to read the tag bits of a scratchpad block. The NI reads the lock bit for example to see if an incoming packet to an address refers to cache or scratchpad area. The access from processor lasts 4 cycles as a common scratchpad access.

Scratchpad tag write: in this case the processor updates the tag bits. If for example a cache line must be locked the processor has to write the lock in the tags of this cache line. In order to perform scratchpad tag write, we firstly read the tags (read modify write) because there are occasions when the new value is calculated from the old one. So the duration of this request is 4 cycles also.

3.5 The Network Interface (NI)

This section briefly (it is outside of the scope of this thesis) presents the Network Interface Architecture. The following subsections explain some primitives relative to configurability, management of network packets, descriptors and communication mechanisms (DMAs).

Our work presents a simple, yet efficient, solution for cache/scratchpad configuration at run-time and a common NI that serves cache and scratchpad communication requirements. The NI exploits portions of the scratchpad space as control areas and allows on demand allocation of memory mapped DMA command registers (DMA command buffers). The DMA registers are thus not fixed and this permits multiple processes/threads to allocate separate DMA command buffers simultaneously. Therefore, our NI offers a scalable and virtualized DMA engine. The scratchpad space is allocated inside the L2 cache and consequently the NI and its control areas are brought very close to the processor. Accessing NI control areas and starting transfers has very low latency when compared to traditional approaches and additionally allows sharing of the memory space between the processor and the NI. Our NI also offers additional features such as fast messages, queues and synchronization events to efficiently support advanced interprocessor communication mechanisms; the description of those features is beyond the scope and page limit of this paper. Another noteworthy concept that is followed by our work is the use of Global Virtual Addresses and Progressive Address Translation [18] that simplify the use of DMA at user-level and facilitate transparent thread migration.

3.5.1 NI Special Lines

The cache-lines in the block scratchpad regions can be either used as simple memory or as NI sensitive memory, e.g. NI command buffers. We exploit the address tag field of the tags, not used in block scratchpad regions, and use it as NI tag to handle the NI sensitive buffers. A couple of bits in the NI tag indicate whether each cache-line behaves as network buffer or as scratchpad. If a cache-line is configured as NI command buffer then any write access there, is monitored. If a command (multiple stores) is written in the appropriate format, then automatic completion is triggered and the buffer is handed to the NI for processing. To achieve automatic completion of the NI commands, we use several bits from the NI tag field to store temporarily the size of the command and the completion bitmap. When a command buffer is completed then it is appended in a ready queue and then the NI tag stores queue pointer values. Figures 3.16 and 3.17 shows the contents of tag of normal and special cache lines. The address and other bits (coherence) of tag part that belong to a normal cache line are free when the space is used as block scratchpad/special cache lines. In these cache lines the lock bit should be set and the valid bit unset as described in section 3.4.

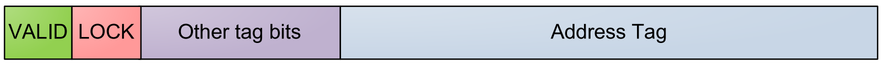

FIGURE 3.16: Normal cache line

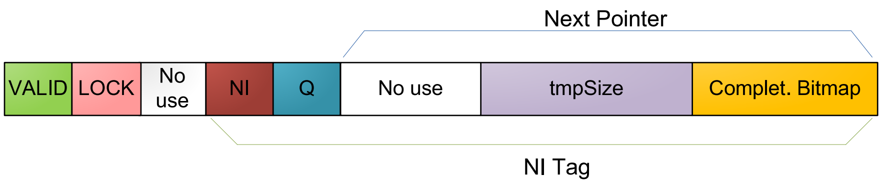

FIGURE 3.17: Special cache line

The information stored in the tags of special cache lines (Figure 3.17) are:

- NI bit: This bit indicates whether a specific cache-line is configured as scratchpad or as NI sensitive region. If the region is declared as scratchpad, then all the other bits of the "NI tag" are free and are not used.

- Q bit: This bit is used by the NI to distinguish between NI command buffers and NI Queues when a cache-line is declared as NI sensitive. The NI command buffers are used by the software to issue commands for outgoing packets, either messages or RDMAs.

- tmpSize: This field is used to store temporarily the size of the message or RDMA descriptor. This field requires as many bits as needed to store the size of a message or descriptor in bytes and depends on the cache-line size.

- Completion Bitmap: This field is used to keep a bitmap with the completed words inside a cache-line and it used by the automatic completion mechanism as described later in this section. This field requires

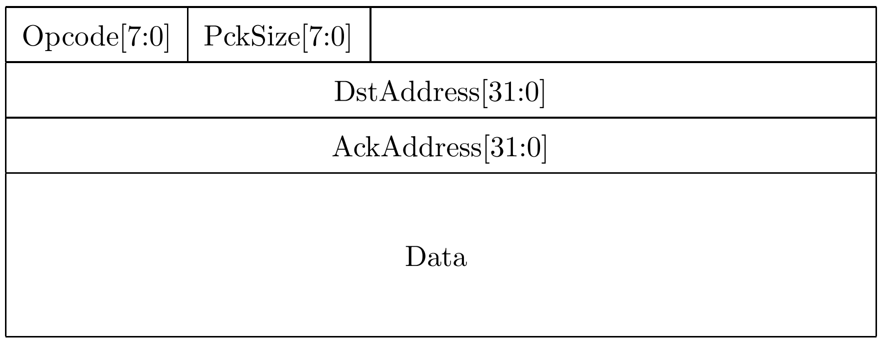

FIGURE 3.18: Message descriptor

as many bits as needed for a mask that marks all the words in a cache-line.

- Next Pointer: When a message or RDMA is completed, then the field tmpSize and Completion Bitmask are not used any more by the completion mechanism. We use the latter bits together with the "No use" bits in order to store a pointer when this cache-line/command buffer is appended in the linked-lists of the scheduler. The bits available for pointer use are 27-bits in a system with 32-bit tags, 32-bit cache-words and 64-byte cache-lines, which are enough to point to 128M cache-lines, it is more than enough.

In order for the software to configure the "NI Tag", that actually controls the use of these cache-lines, it should use a special address range that provides access to the tag contents of the whole cache. Using this address range, all the tags contents appear as scratchpad memory and can be simply read or written. This address range is possibly declared in the R&T.

3.5.2 Message and DMA Transfer Descriptors

The NI defines a command protocol that describes the format and the fields of the commands that can be issued by the software. The command descriptors are issued by a series of stores, possibly out-of-order, coming from

Opcode[7:0]	DescrSize[7:0]	DMASize[15:0]
SrcAddress[31:0]		
DstAddress[31:0]		
AckAddress[31:0]		
Cnt[7:0]	Stride[23:0]	

FIGURE 3.19: Copy - Remote DMA descriptor

the processor and the NI features an automatic command completion mechanism to inform the NI controller that a new command is present. The format of the supported command (message - copy (Remote DMA)) descriptors is shown in Figures 3.18, 3.19. The message descriptor's fields are described below (Figure 3.18):

- Opcode: The opcode of the command. Differentiates Messages from DMA descriptors.
- PacketSize: The size of the packet.
- DstAddress: The destination address of the transfer.
- AckAddress: The address contains information about the completion of the transfer.
- Data: The data to be transferred.

The copy- RDMA descriptor's fields are described below (Figure 3.19):

- Opcode: The opcode of the command. Differentiates Messages from DMA descriptors.
- DescrSize: The size of the descriptor.
- DMASize: The size of the DMA.
- SrcAddress: The source address of the transfer.

- DstAddress: The destination address of the transfer.
- AckAddress: The address which informed about the completion of the transfer.
- Cnt: The Number of the DMAs.
- Stride: The Stride of the DMAs.

3.5.3 Virtualized User Level DMA

The NI control areas are allocated on software demand inside scratchpad regions by setting the special-purpose “NI Sensitive” bit inside the tags. Therefore, any user program can have dedicated DMA registers in the form of memory-mapped DMA command buffers; this lead to a low-cost virtualized DMA engine where every process/thread can have its own resources. To ensure protection of the virtualized resources, we utilize permissions bits in the R&T and require the OS/runtime system to update the R&T appropriately on context switches. Moreover, the support of dynamic number of DMAs at run-time promotes scalability and allows the processes to adapt their resources on their communication patterns that might differ among different stages of a program.

The NI defines a DMA command format and protocol that should be used by the software to issue DMA operations. The DMA command includes as described before the following common fields: (i) opcode - descriptor size, (ii) source address, (iii) destination address, (iv) DMA size. The DMAs are issued as a series of stores filling the command descriptors, possibly out-of-order. The NI features an automatic command completion mechanism to inform the NI controller that a new command is present. The completion mechanism monitors all the stores on “NI Sensitive” cache-lines and marks the written words in a bitmap located in the tag. The address tag and other control bit in locked cache-lines are vacant and therefore the NI is free to utilize them. These bits are used to maintain the pointers for queues of completed DMA commands that have not yet been served either due to network congestion or high injection rate.

Note that all addresses involved in the DMA commands are Global Virtual Addresses and thus the user program does not need to call the OS kernel or the runtime to translate the addresses. The NI only consults, upon DMA packet departure, a structure, located on the network edge, to find the next-hop(s) based on the packet's destination address. This structure stores information regarding the active set of destinations/nodes that each NI communicates with. More details are not note worthy because we are directing outside the scope of this master thesis.

3.5.4 Network Packet Formats

In order to keep the architecture of our NI decoupled from the underlying network, we have defined a packet format that meets the NI needs, while at the same time assumes as few details for underlying network as possible. Figure 3.20 illustrates the proposed minimal packet format. These packets are used to initiate request to DDR controller in order to serve a cache miss. The remote read packet asks the DDR for data and the remote write performs the write back operation. We propose that the links are 32-bit wide. The fields of the packet and their use are explained below:

Remote Read Packet:

- PckOpcd: The opcode of the packet. Differentiates Read and Write packets.
- PckSize: The size of the packet without the last word (CRC)
- RI: If the underlying network supports source routing this field is the Routing Information for packet traveling in the Network through switches/Routers.
- DstAddress (Descriptor source Addr): The address where to store the

PckOpcd[6:0]	PckSize[8:0]	RI[15:0]
DstAddress[31:0] (Descriptor source Addr)		
Opcode[7:0]	DescrSize[7:0]	DMASize[15:0]
SrcAddress[31:0]		
DstAddress[31:0]		
AckAddress[31:0]		
Cnt[7:0]	Stride[23:0]	
CRC[31:0]		

Figure 3.20: Remote read packet

descriptor of the packet.

- Opcode: The opcode of the command. Differentiates Messages from DMA descriptors.
- DescrSize: The size of the descriptor.
- DMASize: The size of the DMA.
- SrcAddress: The source address of the transfer.
- DstAddress: The destination address of the transfer.
- AckAddress: The address contains information about the completion of the transfer.
- Cnt: The Number of the DMAs.
- Stride: The Stride of the DMAs.
- CRC32: This field is a CRC32 checksum that protects the payload from transmission and soft-errors.

Remote Write Packet:

PckOpcd[6:0]	PckSize[8:0]	RI[15:0]
DstAddress[31:0] (Descriptor source Addr)		
AckAddress[31:0]		
Payload (up to 256 bytes)		
CRC[31:0]		

Figure 3.21: Remote write packet

It is composed by the PckOpcd, PckSize, RI, DstAddress (Descriptor source Addr), CRC32 present above, plus the Payload field which is up to 256bytes (Figure 3.21).

Chapter 4

Implementation and Hardware Cost

This section describes the implementation of a chip multiprocessor with a L2 cache controller with configurable scratchpad space. Scaling down the characteristics of modern chip multiprocessor systems, we implement in FPGA Xilinx Virtex II-pro the most features of the architecture described in Chapter 3.

4.1 FPGA Prototyping Environment

4.1.1 Target FPGA

The whole system has been designed and implemented on a Virtex-II Pro FPGA, embedded in a Xilinx University Program board. The size of the FPGA is 30K and the speedgrade is -7C. The FPGA is equipped with two embedded PowerPC processors and as many Microblaze softcore processors as can fit in the FPGA area. As it can be observed in Table **??**, the specific FPGA is rather a medium one. However, the attractive element of all the system was the low price of the XUP board, which enables a multimode system, out of many XUP boards, to be built.

Device	**PowerPC Processor Blocks**	**Logic Cells**	**Slices**	**18Kb Blocks**	**DCMs**	**Max User I/O Pads**
XC2VP2	0	3186	1408	12	4	204
XC2VP4	1	6768	3008	28	4	348
XC2VP7	1	11088	4928	44	8	396
XC2VP20	2	20880	9280	88	8	564
XC2VPX20	1	22032	9792	88	8	552
XC2VP30	**2**	**30816**	**13696**	**136**	**8**	**644**
XC2VP40	2	43632	19392	192	8	804
XC2VP50	2	53136	23616	232	8	852
XC2VP70	2	74448	33078	328	8	996
XC2VPX70	2	74448	33088	308	8	992
XC2VP100	2	99216	44096	444	12	1164

TABLE 4.1: Virtex II Pro Resource Summary

4.1.2 Timing Considerations

The clock frequency of the system is constrained by two factors that constitute the upper and the lower ceiling. The first factor comes from the inability of the Xilinx DDR controller to operate in clock frequencies lower than 100 MHz. This behavior is a documented bug and has also been reported in [27]. The DDR controller cannot operate if the other parts of the design run with clock other than multiples of 50MHz. Even the most recent version of the controller, which comes along with the latest version of EDK 9.1 software, has this disadvantage. On the other hand, the complexity of the implemented system restricts the use of high frequencies. As it will also be shown below, the system is not able to operate above 70 MHz. Thus, the cut of the two sets of possible frequencies, which meets all the criteria, is chosen to be 50MHz, the frequency of the whole system.

4.2 System Components

4.2.1 The Processor

The MicroBlaze embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx Field Programmable Gate Arrays (FPGAs). Figure 4.1 shows a functional block diagram of the MicroBlaze core. The MicroBlaze soft core processor is highly configurable, allowing you to select a specific set of features required by your design. The processor's fixed feature set includes:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus
- Single issue 5-stage pipeline

In addition to these fixed features, the MicroBlaze processor is parameterized to allow selective enabling of additional functionality but all these are outside of the scope of this implementation.

MicroBlaze uses Big-Endian bit-reversed format to represent data. The hardware supported data types for MicroBlaze are word, half word, and byte.

All MicroBlaze instructions are 32 bits and are defined as either Type A or Type B. Type A instructions have up to two source register operands and one destination register operand. Type B instructions have one source register and a 16-bit immediate operand (which can be extended to 32 bits).

4.2.2 Processor Bus Alternatives

The MicroBlaze core is organized as a Harvard architecture with separate bus interface units for data and instruction accesses. The following three

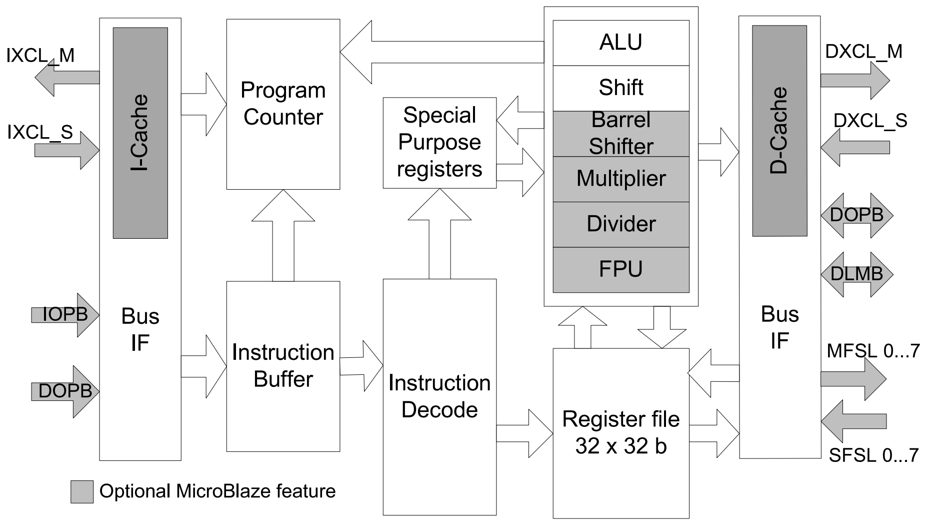

Figure 4.1: MicroBlaze Core Block Diagram

memory interfaces are supported: Local Memory Bus (LMB), IBM's On-chip Peripheral Bus (OPB), and Xilinx CacheLink (XCL). The LMB provides single-cycle access to on-chip dual-port block RAM. The OPB interface provides a connection to both on-chip and off-chip peripherals and memory. The CacheLink interface is intended for use with specialized external memory controllers. MicroBlaze also supports up to 8 Fast Simplex Link (FSL) ports, each with one master and one slave FSL interface.

The major (what we needed) bus interfaces that a MicroBlaze can be configured are:

- A 32-bit version of the OPB V2.0 bus interface (see IBM's 64-Bit On-Chip Peripheral Bus, Architectural Specifications, Version 2.0). It supports both instruction (IOPB) and data (DOPB) interface.
- LMB provides simple synchronous protocol for efficient block RAM transfers. It supports both instruction (ILMB) and data (DLMB) interface.
- FSL provides a fast non-arbitrated streaming communication mecha-

nism. It supports both master (MFSL) and slave (SFSL) interfaces.

- Core: Miscellaneous signals for: clock, reset, debug e.t.c.

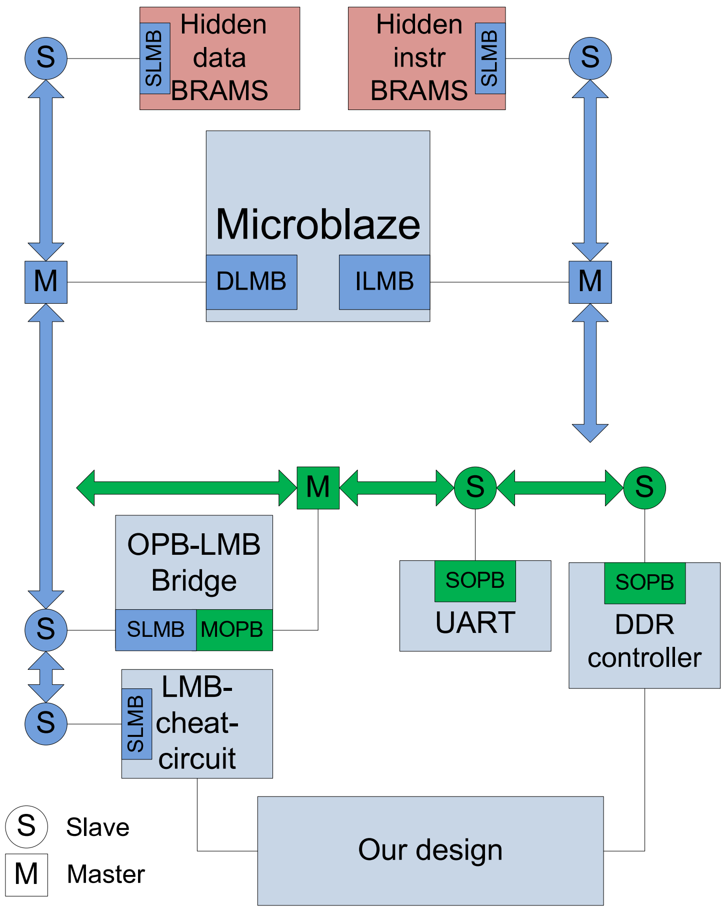

FIGURE 4.2: Block diagram

The OPB bus used in order to connect the processor through an OPB-LMB bridge (Master OPB) with the UART and the DDR controller (Slave OPB). In our design we are not using the Microblaze hidden caches connected in LMB bus, but we need the LMB bus speed (single cycle access). In order to connect to this fast bus our L1 cache, we "cheated" the LMB controller and L1 cache connection to processor through LMB interface achieved. Timing constraints lead us to use an OPB-LMB bridge in order to connect Microblaze with the DDR controller through another path apart from the cache hierarchy path. Figure 4.2 shows a diagram of what we have said so far in this chapter.

FSL interface connects DDR controller (through a switch) with cache hierarchy in order to serve the DDR accesses (misses, DMAs e.t.c.).

4.2.3 L1 Cache

The decision to place the scratchpad space in L2 cache, lead us to add to our memory hierarchy a first level cache. Common implementations of L1 caches have size 32 -128 KBytes and up to 4-way associativity. Scaling down these characteristics due to the FPGA resources, an efficient size for L1 cache is 4KBytes. Due to its small size it is chosen to be direct map. The small size of cache comes with small cache line size which is 32bytes. Some other characteristics are:

- Inclusive property with L2 cache.
- Single cycle access.

L1 cache research is out of the scope of this work so it is as simple as possible and it is not further discussed.

4.2.4 Way Prediction

Way prediction is the technique used to increase the throughput of memory banks, to minimize the L2 cache access time which has increased due to sequential tag access and to economize on the power dissipation of the circuit. Way prediction table is accessed in parallel with L1 so no extra cycle consumed in L2 in order to do the prediction. In terms of the implementation of this feature, we used 4 memory blocks, one for each L2 way and so many entries per block as the number of L2 cache lines per way. Each entry consists of 1 byte, which is the size of the signature (or partial match). Figure 4.3 shows the way prediction table organization and the prediction functionality. The signature in this implementation is created from the XOR function of the three most significant bytes of the processor address. So in order to predict the possible hit L2 ways, the three most significant bytes of the processor address are bitwise XORed and compared with the four signatures, of the index relevant to the processor address. In case of a L2 miss the way prediction table has to be updated with the new signature something that is responsible for, the L2 cache controller. The port A of the way prediction table is read port dedicated to processor for prediction and the port B dedicated to L2 controller only for update (write port).

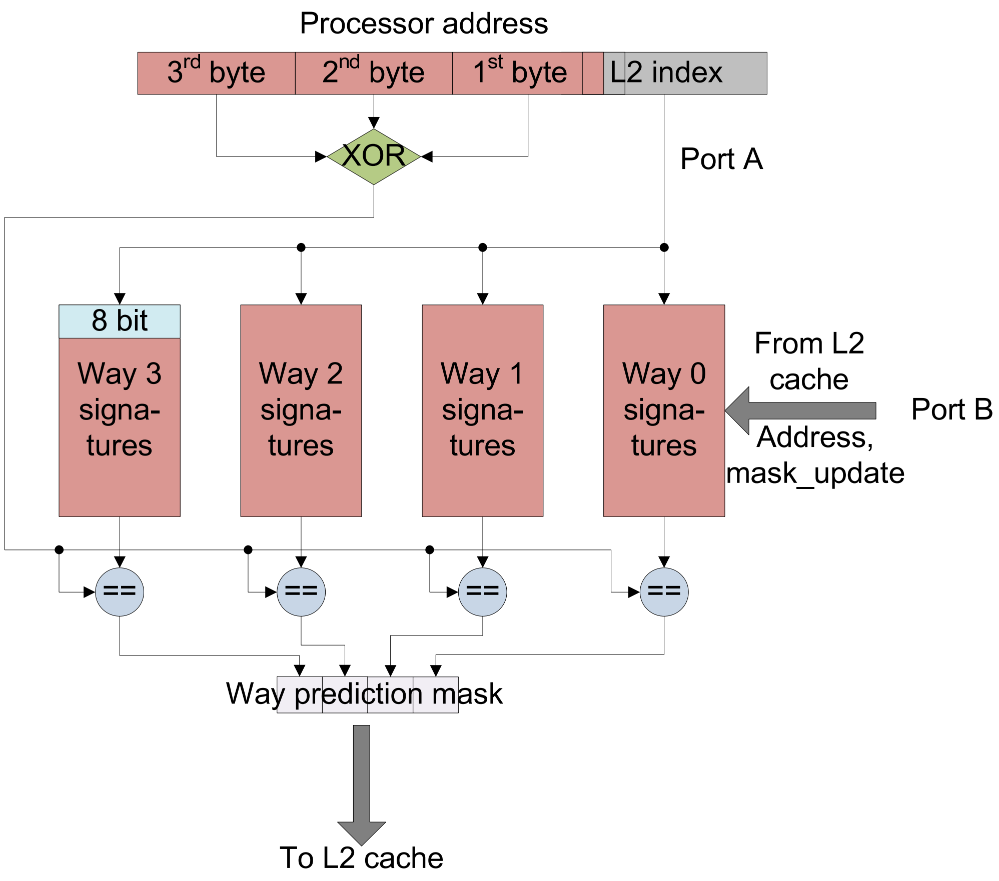

FIGURE 4.3: Way prediction module functionality

4.2.5 Regions and Types Table

R&T table as described with details in Chapter 3, in section 3.1.1, supports the processor with information related to permissions and scratchpad regions in memory level of hierarchy. In the implementation of this master thesis, is a much simpler structure than that presented in 3.1.1. The classification of the regions is done according the processor address bits as shown in Table 4.2. The second most significant bit of the address differentiates the scratchpad (LM) from the cacheable regions. The LM accesses are separated in Data, Tag, Register and Remote accesses. Bits 17:16 separate the different scratchpad regions from each other as presented in the following table. R&T table must reside in both processor and incoming NI side in order to classify the accesses according the permissions and the memory region they refer to. The incoming NI side does not access the R&T table and the differentiation between cache and scratchpad (DMAs) accesses is accomplished by reading the lock bit of the corresponding cache line. Address bits 29:18 denote the node ID which only in remote accesses if different from the node issues the request. Finally bits 15:14 are the L2 cache way in case of scratchpad accesses. Address bit 31 has been chosen to denote the accesses in our address space.

	addr[30]	**addr[29:18]**	**addr[17:16]**	**addr[15:14]**	**addr[13:0]**
Cache	1	X	X	X	L2 Index
LM data	0	Node ID	X	Way select	L2 Index
LM tag	0	Node ID	01	Way select	L2 Index
Regs	0	Node ID	10	X	L2 Index
Rem	0	Not Node ID	X	X	L2 Index

Table 4.2: Regions classification in R&T table

Scratchpad regions can be in data memory blocks, in tag memory blocks (when for example lock bits must be updated), in registers or remote scratchpads of other nodes.

4.3 Integrated L2 Cache and Scratchpad

4.3.1 General features

The remarkable characteristics of the cache controller have been noticed in Chapter 3. Here will be presented some implementation details. Even L2 cache sizes vary from 256 to 2048 KBytes; the FPGA prototype forced us to downscale the size and implement a 64 KByte L2 cache. The optimal [15] associativity way of this size, is 4, and that is the reason why we decide to make our cache, 4-way associative. The decision about the cache line size has taken influenced by both L1 and L2 cache implementation. Concerning pipeline, it has two stages.

4.3.2 Bank Interleaving

As mentioned in Chapter 3, L2 local memory consists of multiple interleaved banks for throughput-bandwidth purposes. The multibanking in the prototype is implemented with 2-ported memory blocks per way, where these ports are shared between the processor, the outgoing and the incoming NI. In the proposed implementation the first port is dedicated to processor and the other is shared between the NIs (Incoming, Outgoing). Theses ports are 64bit wide so the maximum possible throughput is 128bits/cycle even if processor and network is only 32 bits.

4.3.3 Replacement policy

The replacement policy has been adopted in the implementation of the cache controller was not an easy question. Replacement policy interests us only in terms of the scratchpad (locked) regions which are not allowed to be evicted. On the other hand we have to select a replacement policy which does not add area and control overhead in our cache design. Schemes like LRU and Pseudo - LRU add complexity to the L2 cache controller in case of any access (LRU information has to be updated). LRU algorithms use some history information to decide the replaced cache line, so we have to store this information somewhere, something inefficient in FPGA environment especially when replacement policy algorithms is not the research target. The decision was in favor to random replacement policy which is very simple in implementation (only a counter) and need not "history" update. Table 4.3 [15] shows that for our cache the differences between random and LRU policies are negligible.

	2-way		**4-way**		**8-way**	
Size	**LRU**	**Random**	**LRU**	**Random**	**LRU**	**Random**
16KB	114.1	117.3	111.7	115.1	109.0	111.8
64KB	103.4	104.3	**102.4**	**102.3**	99.7	100.5
256KB	92.2	92.1	92.1	92.1	92.1	92.1

Table 4.3: Misses per 1000 Instructions for LRU and Random Replacement policies

4.3.4 Control Tag Bits

An implementation issue in L2 cache is the tag bits organization. The tags are accessed sequentially as mentioned in Chapter 3. In order to be accessed in parallel they have to be placed in a single memory block with a single dedicated port for processor and another for the network interface. The replacement policy has to decide as fast as possible the cache line to be replaced. So the critical, for the replacement policy, tag bits (dirty and lock) are placed in a separate, small (two bits per cache line) memory block, in order to be accessed in parallel (single cycle) for all the four ways and not sequential as all the other tag bits do. This small memory block is two ported, both dedicated to processor, one for read and the other for write purposes in order not to stall the pipeline in cases when a store is followed by a load; where in the same cycle a dirty bit have to be updated (store access) and to be read for the replacement policy of a predicted load miss (way prediction mask equals to zero). A copy of the lock bit is also placed in the tag bits because the incoming NI has to access it to decide if an incoming request is directed to scratchpad or cacheable regions.

4.3.5 Byte Access

In order to support byte accesses (in way prediction table and in data banks), 1-byte wide memory blocks used with separate read - write enable signals. In case of scratchpad tag write, the read modify write property used because we needed bit access. So every L2 cache access in both data and tag banks lasts at least 2 cycles.

4.4 The Write Buffer and Fill Policy

Apart from issues discussed in Chapter 3 concerning NI, here we present some implementation issues. Close to network there is a module called write buffer (Figure 4.4) serving performance issues. During processor DMA descriptor creation, the descriptor is saved in the write buffer (if it is empty). It has one cache line size. Due to the fact that it is closer to network than L2 memory banks, the departure of descriptor to network when it is ready is time efficient. Write buffer supports auto-completion detection which means that when processor finishes creating the descriptor, write buffer detects the completion without needing an extra store of the processor to signal it. This detection is performed by comparing the eight valid bits of the word of the write buffer with the size of the message placed in the first words of the write buffer. In cases of cacheable requests to outgoing NI write buffer is bypassed. Completion detection triggers the DMA engine in outgoing NI which builds packets for the network and sends them.

When a packet reaches its destination the incoming NI has to detect if the packet refers to cacheable or scratchpad area. In case it refers to cacheable area the memory access (write) has to follow wraparound and critical word first policy for size of one cache line, in contrast to scratchpad area, where the data are stored starting from destination address for the size referred in the incoming packet.

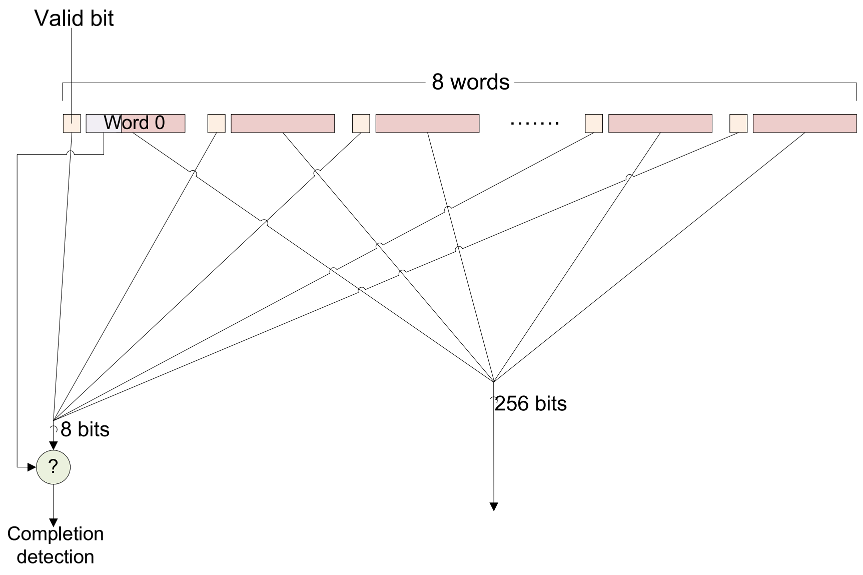

Figure 4.4: Write buffer

4.5 Network on Chip (NoC)

The communication between the different cores and the main memory is accomplished through an unbuffered crossbar with a simple round robin scheduler. It is not further discussed because it is outside the scope of this master thesis.

4.6 Hardware Cost

4.6.1 Hardware Resources

This section presents some metrics showing the implementation cost and the area benefits derived from the cache controller with configurable scratchpad.

Table 4.4 presents the utilization of resources required for the system. The numbers presented correspond to the whole experimental system including parts, such as the processor(MicroBlaze), the DDR controller and I/O path and which are not relevant to the proposed design. The resources occupied by our implementation only, are shown in Table 4.5. The numbers presented are those derived by "synthesis" procedure of Xilinx ISE 9.1 tool.

Resources	**occupied**	**available**	%
FFs	5,192	27392	18
LUTs	8359	27392	30
Slices	7607	13696	55
BRAMs	61	136	44
IOBs	119	556	21
PPC	0	2	0
GCLK	5	16	31
DCMs	2	8	25
Equivalent Gate Count	4,541,071		

Table 4.4: Utilization summary for the whole system

As it can be seen, the whole system occupies the half of the logic-related resources available in the FPGA. Specifically, 55% of the available slices host logic of the system. The 21.5% is occupied by the implemented design, while the rest by soft-cores provided by Xilinx. As far as the memory resources of the system are concerned, 61 out of the 136 available BRAM blocks are used. 40 BRAM blocks are dedicated to the L2 memory system, 3 to L1 cache, 4 to way prediction table and 5 to the Interfaces to the Network as FIFOs. The rest of the blocks from the private memory are available to the processors. Finally, only 21% of the available IOBs are used, mainly for communicating with the external DDR memory. The rest of them can be easily used by a network module, which will provide connectivity with the rest of the world.

Block	FFs	LUTs	Slices	BRAMs
L1 Cache	120	519	269	3
Way prediction	33	58	19	4
R&T	0	11	6	0
L2 control	391	1296	699	0
Memory blocks	0	405	225	40
Incoming NI	286	561	318	0
Outgoing NI	1192	1819	825	1
NoC interface	315	662	588	4
Total	2437	5702	2949	52

Table 4.5: Utilization summary of the implemented blocks only

4.6.2 Floorplan and Critical Path

Figure 4.5 depicts a floorplaned view of the whole design. Our implementation is the dark green box in the upper left corner of the figure. The major modules that shown here are the L1 and L2 cache the outgoing and the incoming Network Interface. The NoCFSLif box shown in the bottom is the interface between the NoC and the DDR controller through the FSL bus.

More details about the area breakdown of the design are presented in the tables and figures of this chapter.

As mentions in section 4.1.2 the frequency of the design is the cut of the frequencies the DDR operates and the critical path of the circuits. As it will also be shown below, the system is not able to operate above 70 MHz. Thus, the cut of these two sets of possible frequencies (50MHz), which meets all the criteria, is chosen to be the frequency of the whole system.

The two dominant critical path are presented in Table 4.6, verifying the 70MHz frequency derived from the ISE Xilinx tool.

The first observation form the table above is that the wire delay is over

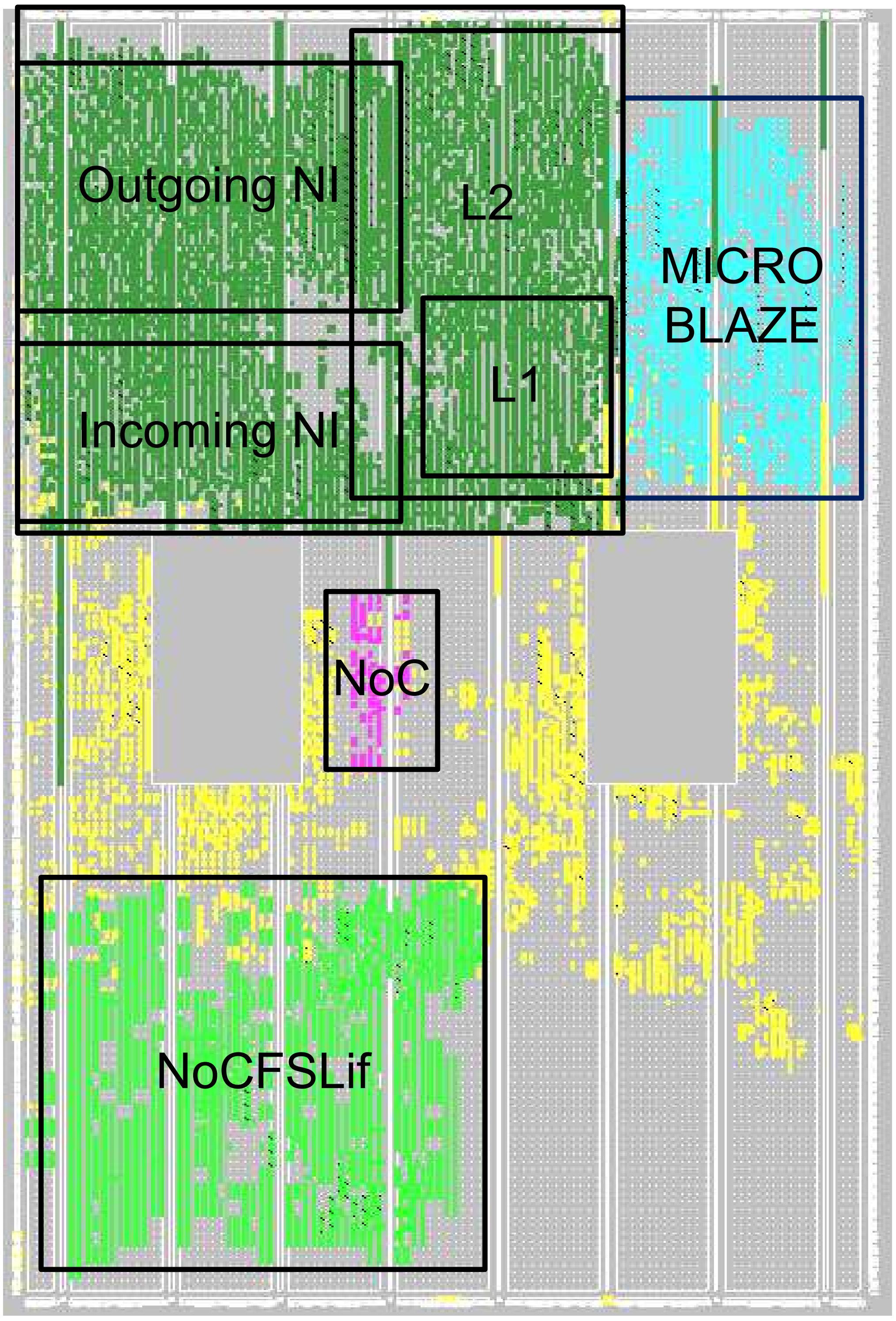

FIGURE 4.5: Floorplanned view of the FPGA (system)

Action	**Logic delay**	**Route delay**	**Total delay**
Write hit	4.661 ns (33%)	9.464 ns (66	14.126 ns
L2 Miss Ending	2.809 ns (20%)	11.236 ns (80%)	14.045 ns

Table 4.6: Delay of critical paths

65% of the total delay something that is relevant to the FPGA characteristics. The path that determines the frequency of the design is the store hit. The tag memory has just been clocked and output the corresponding tag lines. Address and tags are checked for equality. Cache hit is resolved and data are written in the cache. After tag matching in the same cycle (due to pipelining) the cache Ready signal is set and the next request is dequeued from the intermediate (L1 - L2) pipeline register and the tag banks are feed with the new address request. The other critical path presented here is relevant to miss completion. The incoming NI fill the data from DDR and informs L2 cache controller to dequeue the next request from the intermediate pipeline register and the tag banks are feed with the new address request.

4.6.3 Integrated Design Benefit

In order to have a clear essence about the area covering the proposed design, it is compared in terms of LUTS (Figure 4.6) and Registers (Figure 4.7) with a scratchpad only, a cache only design and a sum of the two implementation using corresponding mechanisms. The two figures present an area breakdown of the whole implemented system (besides the processor and the DDR controller).

The area breakdown consists of different implementation modules. These are L1/WP/R&T, L2 control, the outgoing and the incoming NI and the interfaces with the NoC.

Moving on details relevant to the presented different parts of each de-

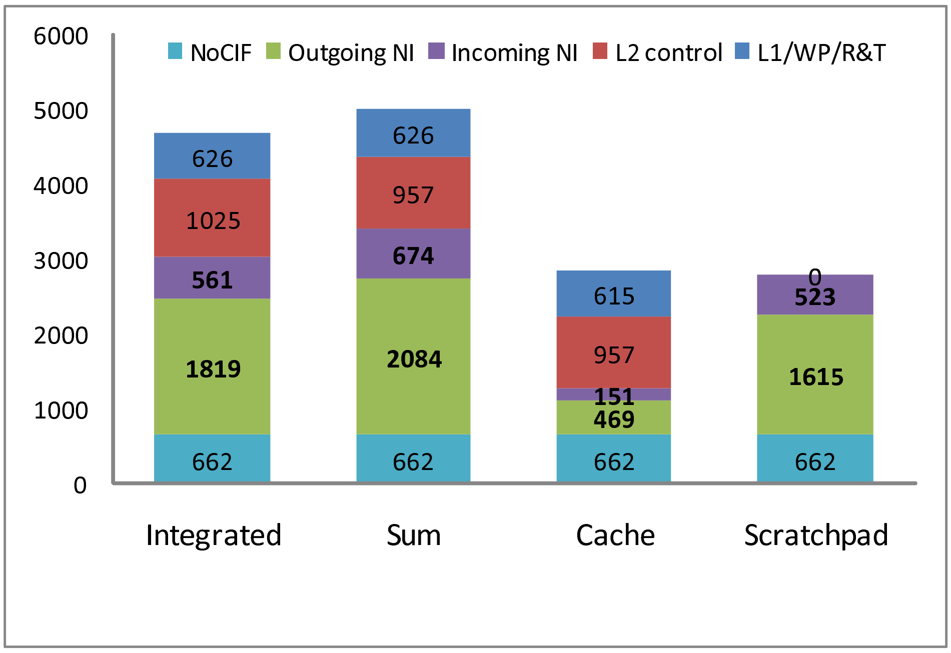

FIGURE 4.6: Area cost breakdown in LUTS

sign, L1/WP/R&T consists of the first level of memory hierarchy, the way prediction table, the R&T table (and a pipeline Register which shortens the path between L1 and L2 cache memories). The incoming and the outgoing NIs are the interfaces of the node to the other world. The outgoing NI contains apart from the necessary FIFOs, and some structures that make the design optimized enough. L2 control is the L2 cache controller managing the processor accesses and initializing the miss serving requests to the outgoing NI. The final part of the break down is the NoC Interface which contains the interface between the crossbar (NoC) and the incoming and outgoing NIs.

The cache only system (third bar Figures 4.6, 4.7) consists of an L1 cache, the way prediction table and the R&T table. The L2 control supports only cacheable accesses. The incoming and outgoing NIs support only cache miss serving using a DMA engine of one cache line DMAs.

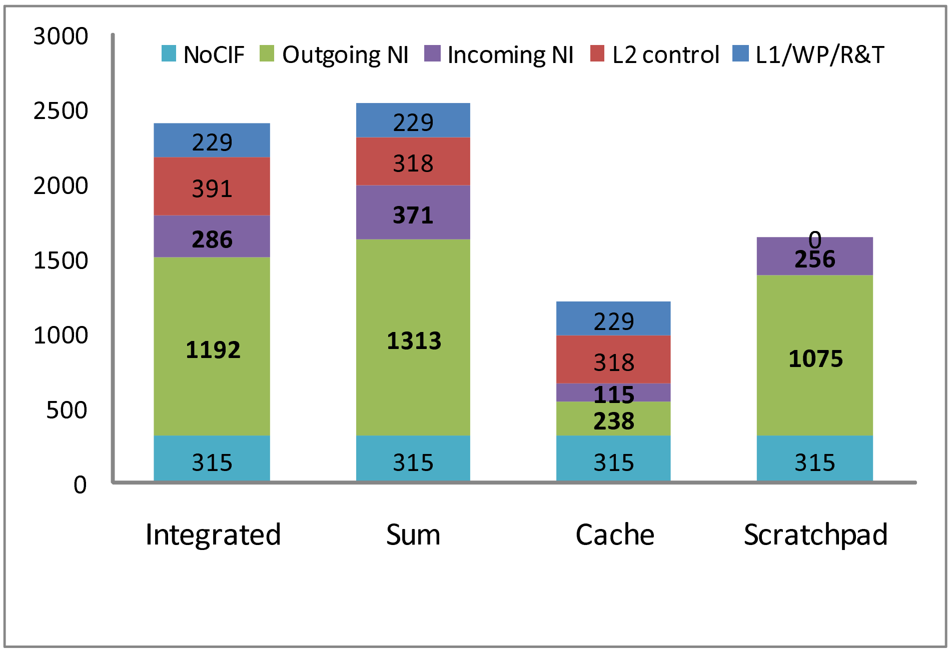

FIGURE 4.7: Area cost breakdown in Registers

In contrast with cache only design, the scratchpad only one do not include way prediction and the L2 control. The outgoing and the incoming NIs in the scratchpad implementation are more complex comparing to the cacheable one due to the message and DMA support and the write buffer and monitor optimization structures.

Making a hypothetical system supporting two disjointed controllers for cache and scratchpad accesses, its area breakdown would match to this of the second bar of Figures 4.6, 4.7. NoCIF counted only once while all the other module area are added.

The first bar matches to the proposed implemented design with integrated cache controller with scratchpad configuration. It is obvious in both figures that the integrated design is area efficient comparing to the sum of the two controllers due to the merging of functionalities; especially in the outgoing and incoming NIs. The gain reaches 8% in LUTS and 5% in

Registers measurements.

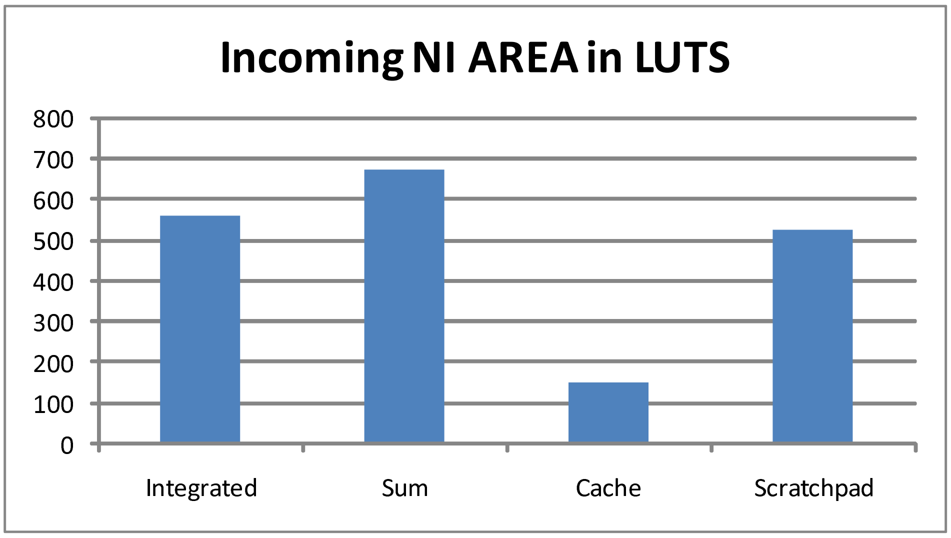

FIGURE 4.8: Incoming NI area cost in LUTs

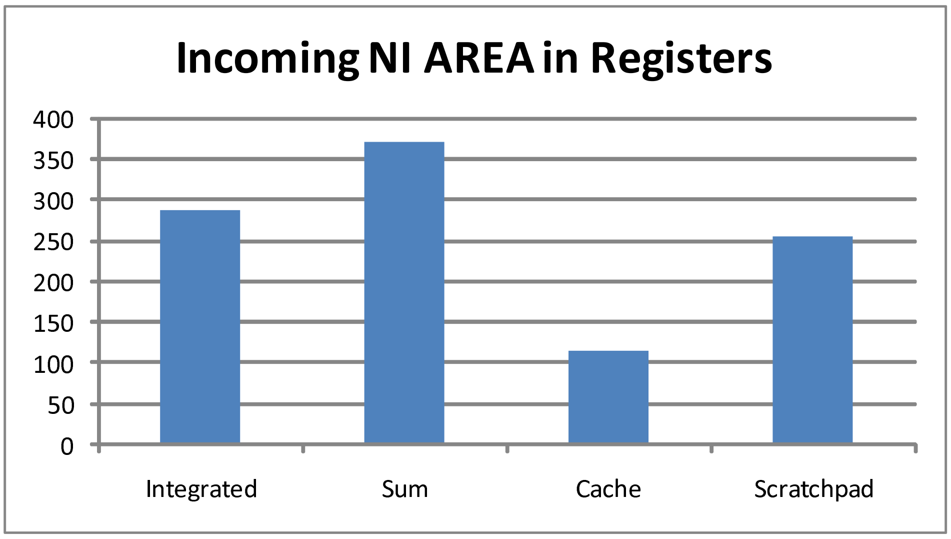

FIGURE 4.9: Incoming NI area cost in Registers

As mentioned above the area gain is observed in the outgoing and incoming NIs due to the FSMs which handle the misses, the messages and the

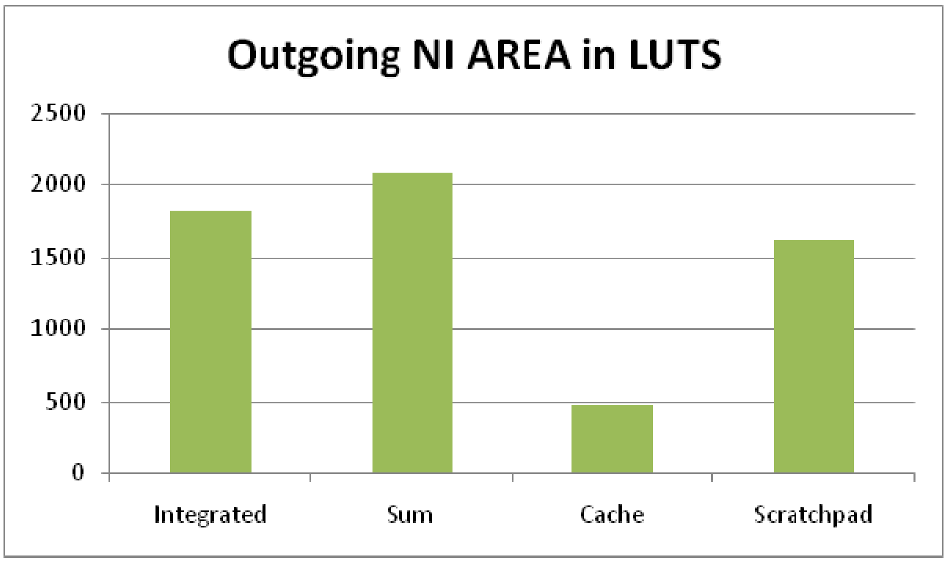

FIGURE 4.10: Outgoing NI area cost in LUTs

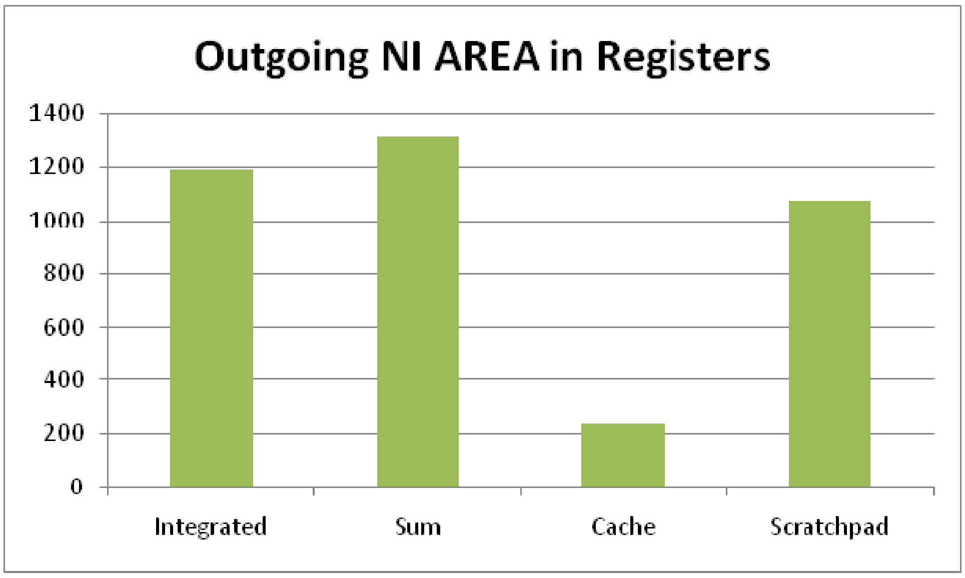

FIGURE 4.11: Incoming NI area cost in Registers

DMAs. Figures 4.8, 4.9, 4.10, 4.11, 4.12, 4.13 isolate these two modules from the whole system and present a more careful view of the four different

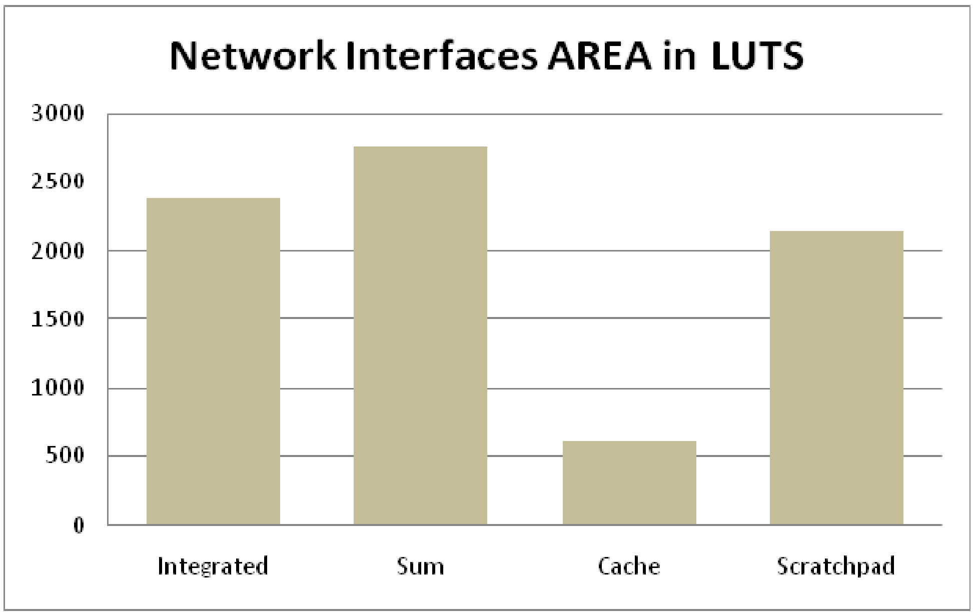

FIGURE 4.12: NI area cost in LUTs

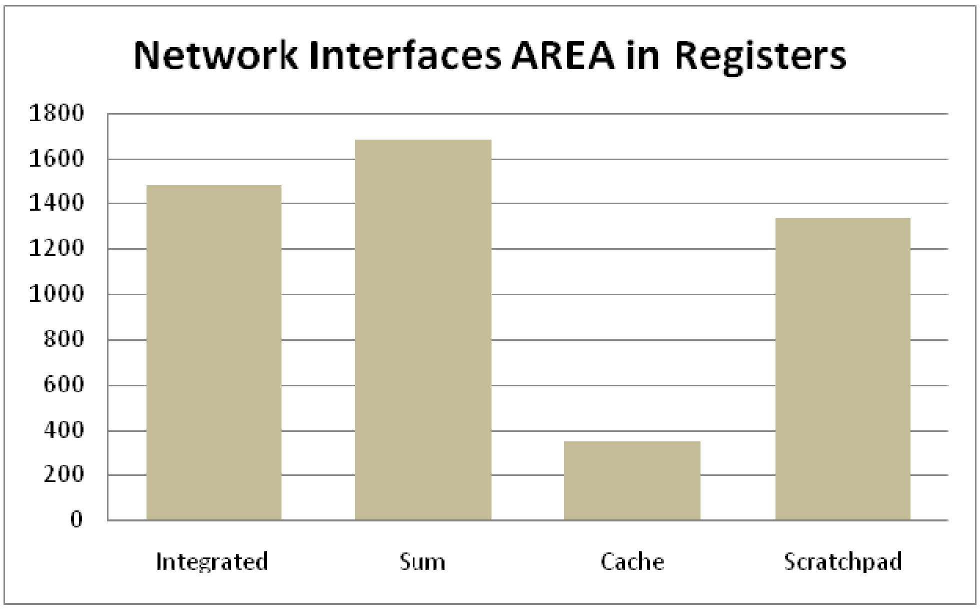

FIGURE 4.13: NI area cost in Registers

systems. The saving achieved in area by merging the two controllers is in terms of 12% in outgoing NI and 20% in incoming NI as the following figures present. The total area saving (sum of incoming and outgoing NIs) is in terms of 13%.

4.7 Design Testing and Performance Measurements

This section presents some evaluation measurements and verification testing of the cache controller. The system parameters used in both evaluation and verification are presented in Table 4.7.

Processor	Clock frequency = 50 MHz, Internal BRAMs enabled (16KB), Internal Data & instruction Cache disabled, No O/S. Processor operate in standalone mode, No Memory translation - Global addressing used
L1 Cache	Clock frequency = 50 MHz, Direct map, Write through, Size = 4KB,
L2 Cache	Clock frequency = 50 MHz, 4-way associative, Write back, Size = 64KB
DDR Memory	Clock frequency = 100 MHz, Size = 256MB

Table 4.7: System Parameters

Type of Access	**Access Latency in clock cycles**
L1 Read Hit	2
L1 Write Hit	2
L2 Read Hit	5 (critical word) / 8 (cache line)
L2 Write Hit	5
Scratchpad Read	5
Scratchpad Write	5
DMA (8 bytes)	29
Message	29
Remote Write (4 bytes)	29

TABLE 4.8: Access Latency (measured in processor Clock Cycles - 50 MHz)

4.7.1 Testing the Integrated Scratchpad and L2 Cache

A large part of the verification procedure was carried out in the implementation phase of the system. Each part was intensively simulated to check as many cases as possible. Furthermore, parts were put together and simulated in order to check their in-between communication. However, deeper investigation of the system is required in order to verify the correctness of the system. Larger programs were written to produce large amounts of cache traffic. In this way, the system's functionality was stretched out to cover all the various cases that it is supposed to support. Apart from small and large simulation random testbenches, the design tested in real hardware using small test programs and two algorithms; a matrix multiplication and a sorting algorithm, the merge-sort with various sizes of workloads.

Chapter 3 presents timing diagrams and pipeline stages of the proposed design. These values are verified through the programs run on FPGA.

In table 4.8 are presented the latencies of different types of accesses derived from simulation. These values are already mentioned in some points in chapter 3. The last three entries of Table 4.8 are not explained due to the fact that are outside the scope of this thesis, but mentioned here because they are types of accesses of the whole design.

4.7.2 Performance Measurements

- **Miss Latency**

Figures 4.14, 4.15 present the miss latency in clock cycles for the two applications (matrix multiplication and merge-sort) for different problem sizes. A simple miss (without write back) takes 79 clock cycles, something that is shown in Figure 4.14 in the first bar were only simple (without write backs) misses happen. In every other occasion in merge-sort the data do not fit in L2 cache so there are cases of write backs, and that is the reason why the miss latency is increased to 100 cycles and remains there for every workload. In matrix multiplication (Figure 4.15) the first two bars present workloads that fit the L2 cache so no misses (apart from cold start) occur. In every other case a behavior similar to that of merge-sort is observed. Miss latency stacks at value 100 due to the fact that the writebacks are proportional to the total number of misses.

Merge-sort algorithm uses two arrays of the same size; the first is the output and the other a temporary. So for an array size (Figure 4.15) 65536 bytes the total workload is 65536 integers/array*4 bytes/integer*2 arrays = 512KBytes. Matrix multiplication uses 3 matrixes of the same size, the two that are multiplied and the product. The workload is estimated as follows: for a matrix size 128x128 the total workload is 128*128*4bytes/integer*3 arrays = 192KBytes.

- **Way Prediction Evaluation**

As mentioned in the previous chapters, way prediction is something significant in our implementation for throughput and power efficiency matters.

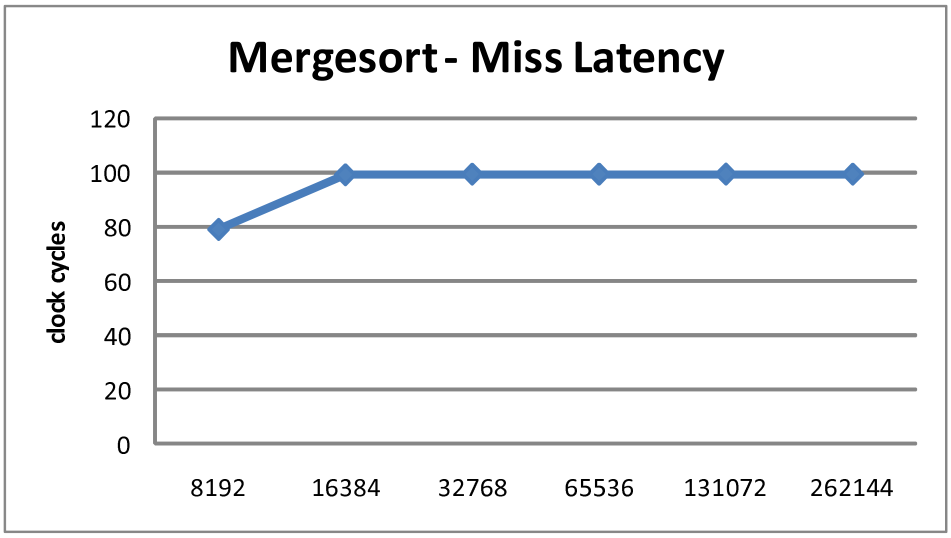

FIGURE 4.14: Merge-sort Miss Latency in clock cycles

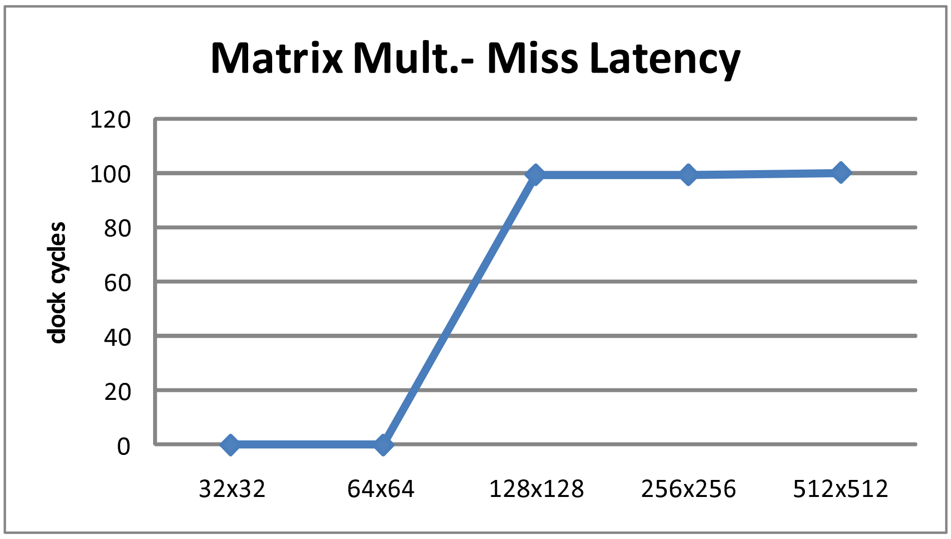

FIGURE 4.15: Matrix Multiplication Miss Latency in clock cycles

Between the PC based and the access address based we choose the second one for accuracy purposes even though address is “valid” later in the pipeline path. During the implementation a noteworthy issue is the decision of the function that will predict the possible hit L2 ways. We compared (through simulation) some different applications including matrix multiplication, FFT (and some SPEC benchmarks). XOR-based way prediction generation mask algorithms compared with simple algorithms (without functions on the address bits). Different sizes of signatures were compared each other to study their accuracy. Below in the tables are presented some of the results (not all of the taken). The algorithm is false negative free as mentioned in the relative section.

Matrix multiplication	Miss rate = 43.8% Hit rate = 56.2% Accesses = 1039717			
Algorithm	XOR function signature		Simple function signature	
Signature size	4 bits	8 bits	4 bits	8 bits
Miss prediction	39.31%	**43.7%**	0%	**39.53%**
One hot mask	56.12%	**56.12%**	28.68%	**42.2%**
Two ones mask	1.32%	**0.08%**	45.34%	**17.2%**
Three ones mask	2%	**0.07%**	0%	**1%**
Four ones mask	0.82%	**0.03%**	25.86%	**0.32%**

Table 4.9: Matrix Multiplication way prediction measurements

Miss prediction mentioned in the tables 4.9 and 4.10 means when the prediction mask is all zeros, so it has been detected a miss without tag matching. One hot mask means that the possible way the data located is only one (direct map caches). Two ones mask is with two possible hit ways and so on for the other two cases.

As it can be seen by the Tables 4.9 and 4.10 the 8 bits signatures is

FFT	Miss rate = 5.45% Hit rate = 94.55% Accesses = 785646			
Algorithm	XOR function signature		Simple function signature	
Signature size	4 bits	8 bits	4 bits	8 bits
Miss prediction	4.87%	**5.43%**	0.03%	**5.01%**
One hot mask	85%	**94.3%**	28.5%	**88.8%**
Two ones mask	6.21%	**0.07%**	45.3%	**5.95%**
Three ones mask	2.25%	**0.14%**	20.9%	**0.17%**
Four ones mask	1.46%	**0.08%**	5.3%	**0.09%**

Table 4.10: FFT way prediction measurements

more accurate than the 4 ones. It is noteworthy to observe the accuracy of the miss prediction (the access is miss without checking the tags) of 8 bit signature. In matrix multiplication the miss rate is 43.8% of 1 million accesses (workload) and the predicted are 43.7% with XOR function and 39.53% without function (only take the 8 least significant bits if the tag address). It is also remarkable to see that the accuracy of the 4bit XOR signatures is more efficient than the 8 bit simple function signature. Another observation concerns the hits. The 8 bit XOR signature predicts the only correct way (one hot mask) with great possibility, converting the 4-way associative cache to direct map in terms of power consumption without converting the cache to phased even it is behaved so.

The same behavior is observed in all the applications was tested.

All these reasons lead us to choose the 8bit XOR-based signature generation for the way prediction algorithm.

Chapter 5

Conclusions

In this master thesis we present an approach for a configurable second level of memory hierarchy. The configuration is achieved between cacheable and scratchpad regions. The configuration apart from the memory space which is configured dynamically between cache and scratchpad is achieved in the "control" level. The mechanisms that manipulate the L2 memory space are unified despite their cacheable or not nature. The proposed cache controller is equipped with several features making it performance and area efficient and power aware. Some of them are: Way Prediction, Bank Interleaving and Outstanding Miss support.

The miss serving of a cache line is achieved through the same functionality the DMAs performed (integrated control).

The measurements taken from the hardware tool reports indicate the benefits on hardware resources due to the integration of the two memory subsystems.

The area gain of the integrated controller comparing to two distinct controllers reaches 7% on average in both LUTS and Register measurements.

Future work will aim at extending the current work in the following ways:

- Increase the level of integration of the functionalities of cache and DMA (scratchpad) controllers.

- Minimize the critical path, increasing the circuit frequency “porting” the design in the larger and faster FPGAs.
- Move the L2 cache tags in a cycle earlier (already exists this cycle) minimizing the critical path and the false positives accesses generated by the way prediction algorithm.
- Generate a DDR controller supporting DMAs as presented before, in this master thesis.
- Support cache coherence providing the mechanisms to share memory between different nodes.

Bibliography

[1] H. Abdel-Shafi, J. Hall, S. V. Adve, and V. S. Adve. An evaluation of fine-grain producer-initiated communication in cache-coherent multiprocessors. In *HPCA '97: Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture*, page 204, Washington, DC, USA, 1997. IEEE Computer Society.

[2] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic. Performance evaluation of cache replacement policies for the spec cpu2000 benchmark suite. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 267–272, New York, NY, USA, 2004. ACM.

[3] F. Angiolini, L. Benini, and A. Caprara. Polynomial-time algorithm for on-chip scratchpad memory partitioning. In *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, pages 318–326, New York, NY, USA, 2003. ACM.

[4] B. Batson and T. N. Vijaykumar. Reactive-associative caches. In *PACT '01: Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 49–60, Washington, DC, USA, 2001. IEEE Computer Society.

[5] L. Benini, A. Macii, E. Macii, and M. Poncino. Increasing energy efficiency of embedded systems by application-specific memory hierarchy

generation. *IEEE Des. Test*, 17(2):74–85, 2000.

[6] G. T. Byrd and B. Delagi. Streamline: Cache-based message passing in scalable multiprocessors. In *ICPP (1)*, pages 251–254, 1991.

[7] M. Byrd, G.T. Flynn. Producer-consumer communication in distributed shared memory multiprocessors. *Proceedings of the IEEE*, 87(3):456–466, Mar. 1999.

[8] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. In *HPCA'96: Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture*, page 244, Washington, DC, USA, 1996. IEEE Computer Society.

[9] M. Dasygenis, E. Brockmeyer, B. Durinck, F. Catthoor, D. Soudris, and A. Thanailakis. A combined dma and application-specific prefetching approach for tackling the memory latency bottleneck. *IEEE Trans. Very Large Scale Integr. Syst.*, 14(3):279–291, 2006.

[10] D. P. et al. The design and implementation of a first-generation CELL processor. In *Proc. IEEE Int. Solid-State Circuits Conference (ISSCC)*, February 2005.

[11] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and J. M. Mendias. An integrated hardware/software approach for run-time scratchpad management. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 238–243, New York, NY, USA, 2004. ACM.

[12] B. R. Gaeke, P. Husbands, X. S. Li, L. Oliker, K. A. Yelick, and R. Biswas. Memory-intensive benchmarks: Iram vs. cache-based machines. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 203, Washington, DC, USA, 2002. IEEE Computer Society.

[13] J. Gummaraju, M. Erez, J. Coburn, M. Rosenblum, and W. J. Dally. Architectural support for the stream execution model on general-purpose processors. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 3–12, Washington, DC, USA, 2007. IEEE Computer Society.

[14] J. Heinlein, K. Gharachorloo, S. Dresser, and A. Gupta. Integration of message passing and shared memory in the stanford flash multiprocessor. *SIGOPS Oper. Syst. Rev.*, 28(5):38–50, 1994.

[15] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[16] InfiniBand Trade Association. Infiniband architecture specification release 1.2.1. http://www.infinibandta.org/specs/register/publicspec/, January 2008.

[17] M. Kandemir, J. Ramanujam, M. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh. A compiler-based approach for dynamically managing scratch-pad memories in embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(2):243–260, Feb. 2004.

[18] M. Katevenis. Interprocessor communication seen as load-store instruction generalization. In *The Future of Computing, essays in memory of Stamatis Vassiliadis, K. Bertels e.a. (Eds.), Delft, The Netherlands*, pages 55–68, Sept. 2007.

[19] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: an energy efficient memory structure. In *MICRO 30: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitec-*

ture, pages 184–193, Washington, DC, USA, 1997. IEEE Computer Society.

[20] D. Koufaty and J. Torrellas. Comparing data forwarding and prefetching for communication-induced misses in shared-memory mps. In *ICS '98: Proceedings of the 12th international conference on Supercomputing*, pages 53–60, New York, NY, USA, 1998. ACM.

[21] T. Koyama, K. Inoue, H. Hanaki, M. Yasue, and E. Iwata. A 250-mhz single-chip multiprocessor for audio and video signal processing. *Solid-State Circuits, IEEE Journal of*, 36(11):1768–1774, Nov 2001.

[22] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA '81: Proceedings of the 8th annual symposium on Computer Architecture*, pages 81–87, Los Alamitos, CA, USA, 1981. IEEE Computer Society Press.

[23] J. Kubiatowicz and A. Agarwal. Anatomy of a message in the alewife multiprocessor. In *ICS '93: Proceedings of the 7th international conference on Supercomputing*, pages 195–206, New York, NY, USA, 1993. ACM.

[24] A. Milidonis, N. Alachiotis, V. Porpodas, H. Michail, A. P. Kakarountas, and C. E. Goutis. Interactive presentation: A decoupled architecture of processors with scratch-pad memory hierarchy. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 612–617, San Jose, CA, USA, 2007. EDA Consortium.

[25] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The alpha 21364 network architecture. *IEEE Micro*, 22(1):26–35, 2002.

[26] S. S. Mukherjee, B. Falsafi, M. D. Hill, and D. A. Wood. Coherent network interfaces for fine-grain communication. *SIGARCH Comput. Archit. News*, 24(2):247–258, 1996.

[27] N. Njoroge, S. Wee, J. Casper, J. Burdick, Y. Teslyar, C. Kozyrakis, and K. Olukotun. Building and using the atlas transactional memory system. In *in Proceedings of the Workshop on Architecture Research using FPGA Platforms, held at HPCA12. 2006.*

[28] P. Panda, N. Dutt, and A. Nicolau. Local memory exploration and optimization in embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(1):3–13, Jan 1999.

[29] P. R. Panda, N. D. Dutt, and A. Nicolau. Efficient utilization of scratch-pad memory in embedded processor applications. In *EDTC '97: Proceedings of the 1997 European conference on Design and Test*, page 7, Washington, DC, USA, 1997. IEEE Computer Society.

[30] P. R. Panda, N. D. Dutt, and A. Nicolau. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Trans. Des. Autom. Electron. Syst.*, 5(3):682–704, 2000.

[31] D. K. Poulsen and P.-C. Yew. Data prefetching and data forwarding in shared memory multiprocessors. In *ICPP '94: Proceedings of the 1994 International Conference on Parallel Processing*, pages 280–280, Washington, DC, USA, 1994. IEEE Computer Society.

[32] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *MICRO 34: Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 54–65, Washington, DC, USA, 2001. IEEE Computer Society.

[33] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. *SIGARCH Comput. Archit. News*, 35(2):381–391, 2007.

[34] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for mlp-aware cache replacement. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 167–178, Washington, DC, USA, 2006. IEEE Computer Society.

[35] F. Raam, R. Agarwal, K. Malik, H. Landman, H. Tago, T. Teruyama, T. Sakamoto, T. Yoshida, S. Yoshioka, Y. Fujimoto, T. Kobayashi, T. Hiroi, M. Oka, A. Ohba, M. Suzuoki, T. Yutaka, and Y. Yamamoto. A high bandwidth superscalar microprocessor for multimedia applications. *Solid-State Circuits Conference, 1999. Digest of Technical Papers. ISSCC. 1999 IEEE International*, pages 258–259, 1999.

[36] R. Rangan, N. Vachharajani, A. Stoler, G. Ottoni, D. I. August, and G. Z. N. Cai. Support for high-frequency streaming in cmps. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 259–272, Washington, DC, USA, 2006. IEEE Computer Society.

[37] S. Sair and M. Charney. Memory behavior of the spec2000 benchmark suite. Technical report, IBM T. J. Watson Research Center, 2000.

[38] C. Scheurich and M. Dubois. The design of a lockup-free cache for high-performance multiprocessors. In *Supercomputing '88: Proceedings of the 1988 ACM/IEEE conference on Supercomputing*, pages 352–359, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[39] SiCortex - Lawrence C. Stewart and David Gingold. A new generation of cluster interconnect. http://www.sicortex.com/products/white_papers/, December 2006.

[40] R. L. Sites and R. T. Witek. *Alpha AXP architecture reference manual (2nd ed.).* Digital Press, Newton, MA, USA, 1995.

[41] G. S. Sohi and M. Franklin. High-bandwidth data memory systems for superscalar processors. *SIGOPS Oper. Syst. Rev.*, 25(Special Issue):53–62, 1991.

[42] R. Subramanian, Y. Smaragdakis, and G. Loh. Adaptive caches: Effective shaping of cache behavior to workloads. *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 385–396, Dec. 2006.

[43] M. Suzuoki, K. Kutaragi, T. Hiroi, H. Magoshi, S. Okamoto, M. Oka, A. Ohba, Y. Yamamoto, M. Furuhashi, M. Tanaka, T. Yutaka, T. Okada, M. Nagamatsu, Y. Urakawa, M. Funyu, A. Kunimatsu, H. Goto, K. Hashimoto, N. Ide, H. Murakami, Y. Ohtaguro, and A. Aono. A microprocessor with a 128-bit cpu, ten floating-point mac's, four floating-point dividers, and an mpeg-2 decoder. *Solid-State Circuits, IEEE Journal of*, 34(11):1608–1618, Nov 1999.

[44] D. Talla and L. K. John. Mediabreeze: a decoupled architecture for accelerating multimedia applications. *SIGARCH Comput. Archit. News*, 29(5):62–67, 2001.

[45] W. Wong and J.-L. Baer. Modified lru policies for improving second-level cache behavior. *High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on*, pages 49–60, 2000.

[46] S. C. Woo, J. P. Singh, and J. L. Hennessy. The performance advantages of integrating block data transfer in cache-coherent multiprocessors. In *ASPLOS-VI: Proceedings of the sixth international conference on Architectural support for programming languages and operating systems*, pages 219–229, New York, NY, USA, 1994. ACM.

[47] K. C. Yeager. The mips r10000 superscalar microprocessor. *IEEE Micro*, 16(2):28–40, 1996.