DYNAMICALLY RECONFIGURABLE ACTIVE CACHE MODELING

ALI BARZEGAR

A Thesis
in
The Department
Of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for the
Degree of Master of Applied Science (Electrical and Computer Engineering)
at

Concordia University
Montréal, Québec, Canada

January 2014

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Ali Barzegar

Entitled: "Dynamically Reconfigurable Active Cache Modeling"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair

Dr. Zahangir Kabir

______________________________ Examiner,

External

Dr. Chadi Assi To the Program

______________________________ Examiner

Dr. Otmane Ait Mohamed

______________________________ Supervisor

Dr. Samar Abdi

Approved by: ______________________________

Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

January 2014

____

Dr. Christopher Trueman
Dean, Faculty of Engineering and Computer Science

ABSTRACT

Dynamically Reconfigurable Active Cache Modeling

Ali Barzegar

This thesis presents a novel dynamically reconfigurable active L1 instruction and data cache model, called DRAC. Employing cache, particularly L1, can speed up memory accesses, reduce the effects of memory bottleneck and consequently improve the system performance; however, efficient design of a cache for embedded systems requires fast and early performance modeling. Our proposed model is cycle accurate instruction and data cache emulator that is designed as an on-chip hardware peripheral on FPGA. The model can also be integrated into multicore emulation system and emulate multiple caches of the cores. DRAC model is implemented on Xilinx Virtex 5 FPGA and validated using several benchmarks. Our experimental results show the model can accurately estimate the execution time of a program both as a standalone and multicore cache emulator. We have observed 2.78% average error and 5.06% worst case error when DRAC is used as a standalone cache model in a single core design. We also observed 100% relative accuracy in design space exploration and less than 13% absolute worst case timing estimation error when DRAC is used as multicore cache emulator.

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest appreciation to my supervisor, Dr. Samar Abdi, for giving me the opportunity to experience the adventure of researching in his lab. Without his guidance, mentoring, and knowledge, this thesis would not have been completed. Further, I would like to thank him again for supporting me during the research and teaching me life lessons.

Secondly, I would like to thank all my teachers and professors that helped me build my background for conducting my research. I am also thankful for examining committee members, Dr. Chadi Assi and Dr. Otmane Ait Mohamed, for reviewing my thesis and for giving me comments and suggestions.

In addition, a great thank you to all my fellow researchers in the embedded systems lab at Concordia University: Ehsan, Kazem, Richard, Partha, Paul, Karim, Shafigh and Zaid for your help and support. I had memorable times with you guys in this office that I will never forget. Thank you very much for tolerating me when I was talking too much in the office.

Last but not least, I would like to thank my wonderful family and friends for supporting me during my studies. I especially thank my parents, brothers, and sisters who never let me down in my life.

Dedicated to my family

and,

in memory of my great father who will always be alive in my mind

CONTENTS

List of Tables

List of Listings

List of Figures

List of Acronyms

ASIC	Application Specific Integrated Circuit
BIC	Built In Cache
BRAM	Block Random Access Memory
CSR	Control Status Register
DCACHE	Data Cache
DDR	Double Data Rate
DRAC	Dynamically Reconfigurable Active Cache
DRAM	Dynamic Random Access Memory
DXCL	Data-Xilinx Cache Link
EDK	Embedded Development Kit
FSL	Fast Simplex Link
FSM	Finite State Machine
ICACHE	Instruction Cache
IXCL	Instruction -Xilinx Cache Link
JTAG	Joint Test Action Group
L 1/2/3	Level 1/2/3
MDM	MicroBlaze Debug Module
MEK	Multicore Emulation Kernel
MPMC	Multi-Port Memory Controller
NOP	No Operation
PC	Program Counter
PLB	Processor Local Bus
Read/Write	Read or Write
RNW	Read Not Write
SOC	System On Chip
TLM	Transaction Level Modeling

CHAPTER 1

Introduction

According to Moore's law, the number of transistors in integrated circuits would double approximately every two years [1]. Due to diminishing transistor size, Moore's law is still holding valid. As a result, the number of integrated modules inside a chip has increased, which has made the integration of several cores feasible [2]. A multi-core processor is composed of several cores communicating through a communication media like buses, point-to-point links, and cross-bars.

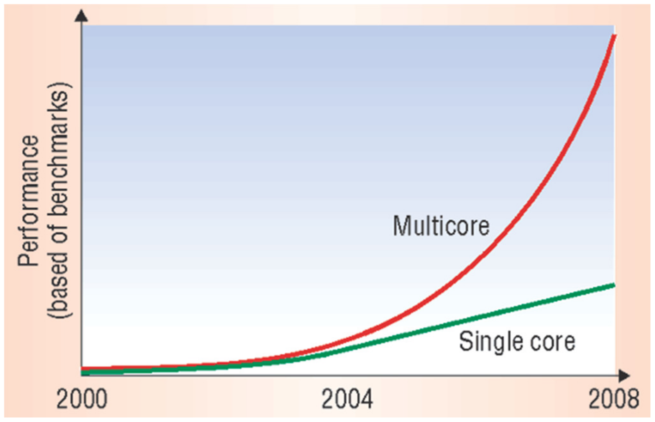


Figure 1.1 Performance of multicore and single core chips over the years [3]

Each core of a multicore embedded processor might not be as fast as a high performance single core processor but, the distribution of the work over different cores enables improvement of the overall system performance [3]. Figure 1.1 compares the performance of an Intel single core and multi-core by running the SPECint2000 and SPECfp2000 benchmarks. It is obvious that the relative performance of multicores has increased significantly over the past few years.

Overall system performance is not only influenced by processor speed, it is also drastically affected by memory access speed. The gap between the processor and the main memory performance is a challenging issue in computer architecture. To address this problem, Memory hierarchy is introduced.

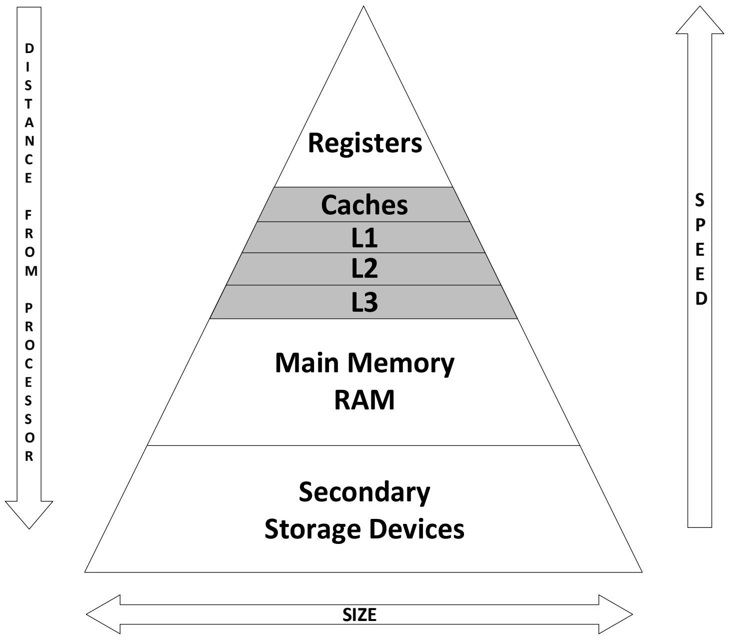


Figure 1.2 Memory hierarchy in computer architecture

Figure 1.2 presents a simple hierarchy that includes registers, three levels of cache, main memory, and disk storage. Cache, specifically, is a fast and relatively small memory used for keeping instructions and data. Data is moved from the slower/larger memory to the faster/smaller memory to decrease the access time in repetitive accesses. Recently, different levels of the cache are introduced to reduce the accesses to the main memory even further.

Memory management in multi-core processors becomes more challenging than conventional single-core processors. The memory management challenge refers to provide a consistent view of memory with various cache hierarchies. This consistency is harder to achieve in multi-core processors where cache hierarchy of several cores should be consistent. In addition to cache consistency problem, the number of pins on the chip remains almost constant with adding more cores. Therefore, adding more off-chip memory is not feasible and using one off-chip memory for all cores may increase access time. To overcome pin constrain challenge, the number of main memory accesses is expected to be minimized.

1.1 Motivation

Prior to a single or multicore system implementation, it is general practice to build a prototype of the system. Though prototyping, the designer is capable of testing and exploring design options in order to avoid unexpected problems and to optimize the system. In addition, for early system validation and determining the functional accuracy of the design, it is necessary to have a model of instruction and data cache; a model that is highly accurate and can simulate the cache in high speed.

Trends toward multicore processing lead us to extend the cache model to emulate multiple caches of the cores. Level 1 (L1) cache modeling for multicore emulation is more involved than implementing a built-in regular cache in the physical core. The cache model should be capable of simulating different caches of the virtual cores by dynamically changing its context. The model also should be dynamically reconfigurable, so the time required time for emulating different configurations be reduced.

The target caches to be modeled, both in single and multi-core systems, are instruction and data, L1, direct mapped, with write-through policy caches since

- To achieve the best performance of a system, it is necessary to have both instruction and data caches. As a result, the target system has instruction cache as well as data cache.

- L1 cache has a great impact on the system performance since it is the closest level of cache to the processor. Embedded systems might use only the primary level (L1) of cache in order to reduce the complexity and cost of production; hence, we only targeted L1 cache.

- Direct mapped caches use less logic than set-associative and full-associative caches; therefore, direct mapped cache consumes less power than other associative caches, and is preferable in simpler embedded systems.

- Write-through writing policy in data cache ensures that the cache-store remains consistent. It is important to keep data consistent during multicore emulation, so write-through policy was chosen for the writing policy.

1.2 System Prototyping

Prototyping is the procedure of building a model of a system in order to check for flaws and to optimize the product before manufacture. Virtual and FPGA Prototyping are common techniques that can be used for early system validation.

1.2.1 Virtual Prototyping

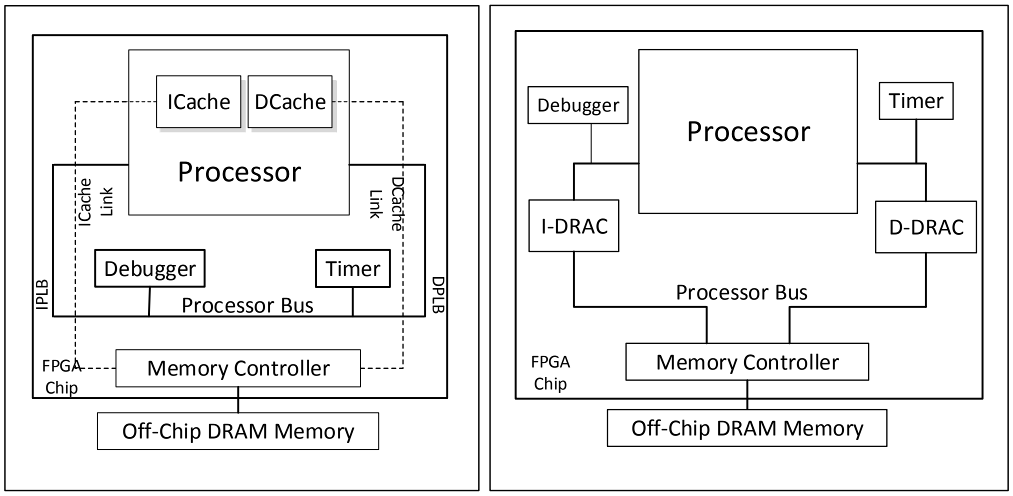
Virtual Prototyping provides an early, abstract functional model of the hardware. This technique uses software simulation libraries and tools to build a software model of the design. Designers use virtual prototyping to test, optimize, validate and verify their design in a software framework. Virtual prototyping uses a computer software-based model of a system or component to avoid early system problems and reduce time-to-market.

Virtual prototyping involves producing a platform or a set of functional models for different system components such as processor, memory, or bus. These techniques offer the designers a low cost tool on a computer that benefits from scalability, flexibility, and ease of debugging; however, current virtual prototyping tools [4], [5] compromise cycle-accuracy for simulation speed. Virtual prototyping is performed according to a functional model of the system. This model accelerate the simulation and evaluation of the system but it sacrifices the accuracy for the speed.

1.2.2 FPGA Prototyping

FPGA prototyping refers to the process of verifying the functionality and performance of the prototype by implementing the design on a field programmable gate array (FPGA). FPGA prototyping involves instantiating processor cores, bus, memory, and the peripherals on a chip. FPGA prototypes are typically, several orders of magnitude faster than cycle accurate virtual prototypes, while still providing cycle accuracy and almost at-speed simulation of the target design. This technique is also the pre-silicon stage of SoC and ASIC design in order to avoid expensive silicon re-spin. Although the speed of FPGAs are lower than ASIC chips, the configurability of FPGA distinguishes them over ASIC technology.

The main drawback of FPGAs is the lack of scalability. Since most of the design is implemented by the logic gates and the logic gates are limited on FPGA, it is hard to implement several design on a single FPGA chip. Besides, debugging of a design on FPGA is difficult and time consuming in multicore design. As much as the number of cores increases, the complexity of the design increases.



(a) System with built-in cache (b) System with DRAC model

Figure 1.3 Cache modeling in single core design

1.3 Methodology

Figure 1.3 presents the methodology of DRAC in a single core design. Figure 1.3 (a) shows the overview of a single processor system with built-in data and instruction cache. The memory controller, DRAM, and peripherals such as timer and debugger module are also implemented in the design. The communication between the cache, processor and main memory is through cache link interconnection. The target built-in cache is a level 1 (L1) direct mapped cache. Direct mapping allows simple and fast speculation of the information. L1 cache also has the greatest impact on the system performance. Figure 1.3 (b) illustrates the modeled system with instruction and data DRAC. We excluded both built-in instruction and data cache (IBIC and DBIC) and implemented DARC emulator instead. The cache is controlled in software by the driver that is provided with the model. It can be simply enabled, disabled, or reconfigured during program execution time.

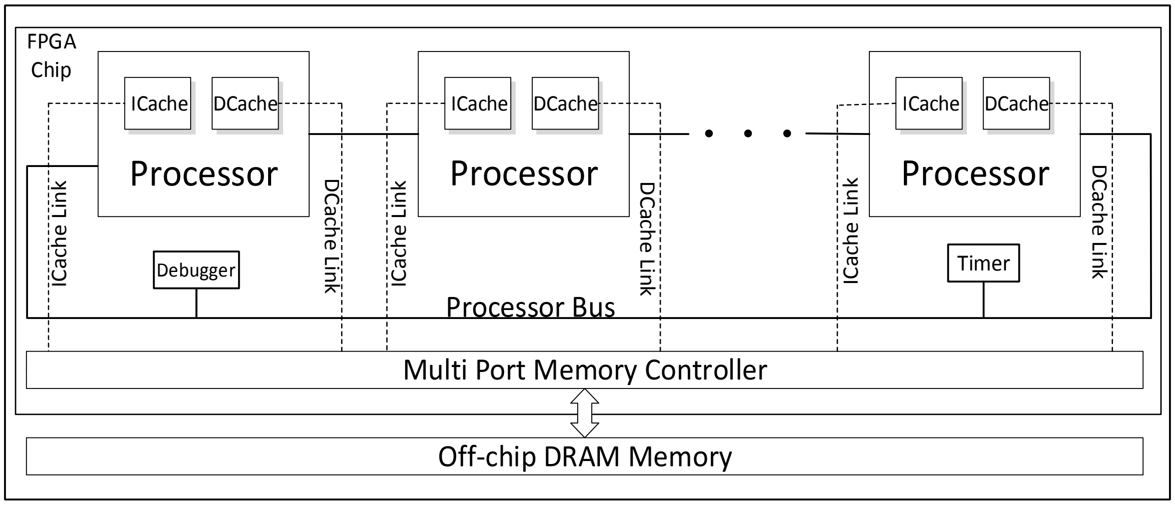


Figure 1.4 Full FPGA prototype of a multicore design

Figure 1.4 shows the full FPGA prototype of a target multicore design. In this design, processors running in parallel and have separate accesses to the main memory provided by Multi Port Memory Controller (MPMC). Each processor has its own separate L1 instruction and data cache. Hybrid prototyping, which is a promising prototyping technique, targets the multicore design, and emulates the system on a single physical core depicted in Figure 1.3 (b). Hybrid prototyping provides the benefits of scalability of virtual prototyping, as well as the cycle-accuracy and speed of FPGA prototyping

The key idea is a Multicore Emulation Kernel (MEK), which is a software layer that executes on a single target core that is physically implemented in FPGA. The MEK dynamically schedules different tasks running on independent cores on a single physical target core, to simulate software execution on a multicore platform. The MEK manages the state of the individual cores and the logical simulation times. The original hybrid prototyping system uses Block RAM (BRAM) for the program and data, since it does not support memory hierarchy. This work extends hybrid prototyping by supporting memory hierarchy consisting of L1 cache, and off chip main memory, implemented in DRAM.

L1 cache modeling for hybrid prototyping is more involved than implementing a built-in regular cache in the physical core. The cache model should be capable of simulating different caches of the virtual cores by dynamically changing its context. We proposed a Dynamically Reconfigurable Active Cache (DRAC) model to support multiple L1 cache contexts. DRAC is designed as a run-time configurable cache.

DRAC is also designed as an active cache model. It actively interacts with the target system and provides required instruction and data for the processor. Similar to BIC, DRAC uses BRAM for data and tag memory; hence, it speeds up the system performance in compare to the system without cache. Upon a memory request, DRAC fetches the regarding instruction or data from the main memory, keeps a copy of the information in data memory, updates the tag memory, and delivers the information to the processor. On the next access to the same address location, DRAC retrieves the information from its data memory and decreases the fetch time. The active behavior of DRAC also provides the model the ability to control memory transaction latency over the bus. In order to model BIC behavior, DRAC injects extra cycles in the required cases to scale the memory transaction latencies. Thereby, the execution time will be a linear factor of the system with BIC.

The run-time configurability of DRAC provides MEK the ability to change the cache configuration during program execution. The configurability has to be run-time since the emulation process is fully controlled by the MEK software. The MEK can change the cache configurations, like the size of the cache, depending on the emulated core’s configuration. Therefore, it is possible to have different cache contexts and configurations for each core.

1.4 Related Work

There are different approaches to model the cache behavior. Cache modeling can be broadly divided into software-based and hardware-based modeling techniques. Each has its advantages and drawbacks. In following section, we discuss some of these techniques.

1.4.1 Software Cache Modeling

Software models are developed in a software environment on a host computer. They support a high degree of configurability; however, software-based simulators compromise accuracy for simulation speed, and vice versa. Since software models are running on host computer and there are plenty of resources on computer, software models promotes scalability and flexibility in cache modeling.

1.4.1.1 Trace driven simulators

Trace driven simulation has been one of the popular cache modeling approaches for many years. Trace driven simulators consist of three stages: trace collection, trace reduction, and trace processing. They extract the memory traces through the software, and use the extracted data to simulate the behavior of the cache. These simulators are developed in a software environment and offer a good degree of configurability, however they suffer lack of accuracy. Since the entire performance of the cache is not visible in this technique, it is difficult to determine the error. Processing the traces can also be time consuming which is not desired for design space exploration.

The early trace driven cache simulator proposed by Denning [7] collected memory references based on their independent probability. This model does not consider the locality of memory references inherent in a real trace. Dinero IV [8] is another simulator based on trace collecting written in C language. This technique needs repetitive simulations to reach to higher accuracy and is not eligible for multicore processing. MMCacheSim [9] is a highly configurable matrix multiplication cache simulator that can be used in multicore processing system. This model that is developed in java, simulates the execution time and number of cache misses on processor with different cache sizes, lines, levels, associativity, and replacement policies. Like other trace driven simulators, this model suffers from lack of accuracy.

1.4.1.2 Transaction Level Modeling

Transaction-level modeling (TLM) is a high-level approach to model cache behavior. This is a widely used technique for abstracting the system behavior in both processing and memory level. TLM modeling is the response to increase the software simulation speed by abstracting communication details. TLM models extract the memory access information and abstract them into cache models. This software simulation technique faces two major challenges: the accuracy, and system performance. Such these simulators, are not accurate enough since they cannot statistically resolve addresses during source code instrumentation. Cache modeling brings large overhead in simulation performance and reduces the speed.

Statistical models [4] randomly generate cache misses according to certain miss rates. Such these models, are not able to catch entire data access pattern of a specific program, therefore they are not accurate. Another type of TLM simulators use the same addresses as host data [10], [11], [12]. In these simulators it is assumed that locality of reference is similar to the host and target memory; otherwise, the simulator is not accurate. Beside this challenge, the memory accesses would not be visible in source code.

Hybrid source Level Simulation data cache simulator [13] is another method to make TLM cache models faster and more accurate. This technique combines the statistical analysis of data flow in machine code with high abstract source level simulation of the application. The main drawback of this technique is the need for repetitive simulation for increasing the accuracy of the simulator. This reduces the simulation speed and creates an overhead due to binary translation. Moreover, this approaches does not take multicore processing into account. The cache supports only single core designs.

Zhonglei Wang et al. [14] proposed a method to increase the accuracy of TLM cache modeling and accelerate both instruction and data caches. Since data addresses cannot be resolved at compile time, there is always difficulty in data cache modeling in TLM. Their proposed model addresses this problem and increases the data cache accuracy. The model support multicore processing; however, in lower cache sizes, the accuracy and the system performance decreases.

1.4.1.3 Single pass simulators

Single pass cache simulators are introduced in order to reduce the number of required simulations. Such methods are able to simulate different cache designs in a single pass through the benchmark traces by evaluating multiple cache configurations simultaneously. Based on data structures and the methodology, single pass simulators divide into two categories: Stack-based algorithms, and tree-based algorithms. Mattson et al. [15] proposed the earliest stack-based algorithms for fully-associative caches. In order to include the set-associativity, direct mapped policy, and to reduce the simulation time, Hill and Smith [16] extended the model. However, their technique have a comparably high simulation time.

To reduce the processing time, tree algorithm was introduced. This signal pass simulation algorithm stores the data accesses in a more efficient way than stack-based algorithm. Sugumar and Abraham [17] proposed a model that reduced the processing time by specializing the cache parameters being varied. All the mentioned single pass simulators only support one level of cache. T-SPaCS [18] supports two levels of instruction cache. This cache model uses a stack-based algorithms to model L1 and L2 caches. The main drawback of this model is the complexity of the model, and simulating exclusively instruction cache.

Janapsatya et al. [19] proposed a simulation algorithm to reduce the complexity and simulation time of the cache model. This model is categorized as tree-based single pass simulators and is capable of rapidly find the L1 cache miss rate for an application. It is also a good model to quickly explore different cache parameters from a single read of a large program trace. The main problem of this technique is the long simulation time (2.5 days for 268 cache configuration). All discussed single pass simulators are only capable of being used in a single core design, and are not applicable for multicore emulation.

1.4.2 Hardware Cache Modeling

Hardware cache models are introduced to address the simulation speed and accuracy issues of software simulators. Although software platforms provide flexibility and scalability for cache modeling, they are suffering from cycle accuracy and simulation speed. As software models get more complicated, longer simulation time is needed to perform the modeling. Increasing the complexity of the design in multicore processing systems and the need for more accurate cache models have lead the designers to develop cache models in hardware. FPGA technology is commonly used for hardware modeling. This technology promotes fast and accurate platform for online evaluation of the model. Hardware cache models can be classified into passive and active emulators.

1.4.2.1 Passive cache emulators

Most of the FPGA-based emulators are passive models. Passive emulators are like hardware monitors and bus probes. They are connected to the processor bus and collect transaction traces over the bus. Based on the statistics generated from these data traces, the target cache is modeled. Such models do not sacrifice the speed for the accuracy. They perform at-speed simulation since the embedded software is executed on the actual soft or hard core processor. This way the accuracy and the speed increase. Passive emulators also provide monitoring of the traces over the simulation. Therefore, the embedded designer is capable of observing the memory transaction and simply validate system functionality. Having all these advantages, passive models do not provide full system performance. Since the results in passive models are only depended on the system statistics, it is not possible to observe the entire system behavior and the effect of cache on the system.

Yoon et al. [20] proposed RACFCS cache model that can generate the accurate and long traces to simulate cache. This cache emulator is based on trace collecting on the fly. It directly connects to the processors output pins and stores address references, data, and control signals. RACFCS configurability and programmability are the main advantages of this model. The simulation speed is acceptable in this model (110 minutes for all SPEC benchmark), however there is no discussion about the accuracy in their research.

MemorIES design by Nanda et al. [21] is an online cache emulator that supports different cache sizes, associativity, line sizes, attributes, and cache writing algorithms in real-time. It consists of several FPGA boards and DRAM memory. It sits on a symmetric multiprocessor (SMP) bus, passively monitors all the bus transactions, and emulate shared L3 cache. There is a controller board that plugs into the memory bus of the host system and trace transactions.

Based on these information, the emulator generates cache statistics such as hit ratio, read/write ratio, and amount of cache-to-cache interventions. MemorIES is specifically designed for IBM S70 class RS/6000 or AS/400 servers.

Ravishankar and Abdi [22] proposed P-cache that is an L1 data cache emulator. P-cache is an FPGA-based passive model that is connected to the processor bus and probes memory traces. It provides cycle-accuracy and observability over software debugging and analysis. P-cache model also supports different configurations like different cache sizes, line sizes, and writing policies, but the configurability is static. The model it is not a dynamic cache model. It should be synthesized after changing each configuration. Furthermore, the experimental results are only based on target system statics and the model does not provide any information about system performance. Besides, it is not possible not use this model in the multicore emulation system since it only support one cache context.

1.4.2.2 Active cache emulators

Active cache emulators are introduced to model the entire performance of the cache. Such emulators actively interact with the processor and the main memory. The active emulators are complementary for passive cache emulators. They are not only tracing the memory transaction over the bus, they also provide instructions or data for the target processor and act as an actual cache. The system using active emulator experiences similar hit and miss latencies to the built-in cache. Hence, the system speed up/down due to cache behavior is observable by the emulator. Like other FPGA-based emulators, active models can perform at-speed emulation and reduce the simulation time. Moreover, the accuracy is slightly better than passive emulators since the cached is modeled in more detail.

PHA$E developed by Chalainanont Et al. [23] is a real time L3 cache simulator that is capable of being used in Pentium-based system hosts. This hardware simulator implemented on Xilinx XC2V1000 FPGAs supports different cache configurations and set associativity. This model is useful for investigating cache hierarchy efficacy due to its real-time capability and programmable feature. However, it cannot be used in a multicore emulation system since it is specifically designed for a single core system.

RMP [24] is a FPGA-based emulator using rapid-prototyping methodology for multiprocessor system emulation. It consists of several FPGA boards that each emulates a single processor. In RMP, the entire hardware of the target machine, including the cache, is emulated by the platform. Two level of cache is implemented in system. The first level is a direct mapped write-through with block size of 16 bytes, and the second level is a two-way set-associative write-back cache and block size of 16 bytes. RAMP [25] is another FPGA based emulator that supports the instantiation and integration of hundreds of cores. Similar to RMP, RAMP emulates the entire target system including cache. Both RMP and RAMP emulators suffer from high cost and design time for such full system prototyping.

Active Cache Emulator (ACE) proposed by Nurvitadhi et al. [26] is another active cache emulator. ACE is an FPGA-based emulator that models an L3 cache actively in real-time. It provides feedback to its host, by injecting time delays to the memory transactions. So, it seems the system experiences hit or miss of the actual cache. ACE architecture is specifically designed to interface with a front-side bus (FSB) of a typical Pentium-based PC system. It emulates an eight-way write back cache, and sits on the processors slot. Besides all ACE advantages, it is specifically designed for a typical Pentium-based PC system.

Table 1.1 Comparison of different cache modeling techniques

	Feature / Model	Accuracy	Simulation Speed	Run-time Configurability	Multicore Emulation Support
Software Modeling	Trace Driven	×	×	✓	✓
	TLM	×	✓	✓	✓
	Single Pass Through	✓	×	✓	×
Hardware Modeling	Passive	✓	✓	×	✓
	Active	✓	✓	✓	✓

Table 1 compares different cache modeling techniques and summarizes each technique's capability. DRAC model is designed as an active cache emulator. It offers the emulation speed of active models, the observability of passive models, and run-time configurability of software models to support multiple cache contexts.

1.5 Thesis Contribution

This thesis presents a novel dynamically reconfigurable active instruction and data cache model which supports hybrid prototyping technique. Proposed DRAC emulator is ready to be used in multicore emulation kernel.

The main contributions of this work are,

- Introducing a standalone cycle-accurate L1 instruction and data cache model on the FPGA. DRAC that is designed as an active cache emulator, provides functional and cycle accuracy as well as system observability.
- Improving simulation speed in compare to other software simulators and passive hardware emulation techniques. Implementing DRAC on the FPGA and utilizing

on-chip BRAM as data and tag memory for the cache, significantly speeds up the simulation speed.

- Parameterizing the cache timing model in order to make it a general cache model that can work with different processors and memory buses.
- Extending the cache model to be a run-time reconfigurable emulator and supporting multiple cache contexts. Since multicore emulation process is controlled by the software layer in hybrid prototyping, DRAC's configurability and its cache context can be changed during program execution.
- Extending the hybrid prototyping system to be scalable to realistic multicore designs with cache hierarchy. Thereby, it is possible to run large size multi-thread applications and study design space exploration.

1.6 Thesis Outline

The rest of this thesis is organized in 5 chapters. Chapter 2 introduces DRAC design as a standalone cache model and provides the detailed architecture. Chapter 3 explains how DRAC emulator is extended to be used in hybrid prototyping. This chapter takes a closer look at hybrid prototyping methodology, multicore emulation kernel, and the system design. Chapter 4 presents the timing model of DRAC and the approach used for execution time estimation. DRAM behavior and its effect on the cache is studied in this chapter. Chapter 5 shows the experimental results of DRAC as a standalone cache model and as a multicore cache emulator in hybrid prototyping. At the end of this chapter, design exploration is discussed and different designs are compared to each other. Finally, chapter 6 concludes our work and presents the future work.

CHAPTER 2

DRAC Model

DRAC is a cycle-accurate data and instruction cache model. It can model cache of a single core design, as well as emulating different caches of a multicore design. The design architecture of DRAC makes it an on-chip peripheral cache model that can work with most of the processors. Through some changes in the interface, it is possible to customize it for different processors.

DARC is implemented as an interface between the processor and the main memory. It receives memory accesses from the processor, processes the requests, and delivers the instructions and data, as needed. Therefore, it actively interacts with the system. The active behavior of the emulator provides the feedback for the emulator to inject necessary time delays. By adding these time delays, the program's execution time, when using DRAC, will be a multiple of the execution time with built-in cache.

2.1 DRAC Interface

In bus architecture, master/slave is a model of communication for the devices on the system bus. The master devices connected to the bus are able to initiate transactions, while the slave devices can only respond to the transaction requests. DRAC is as a slave peripheral on the processor side, and master on the memory controller side to the processor bus. Bus mastering provides the full control of DRAC over the main memory.

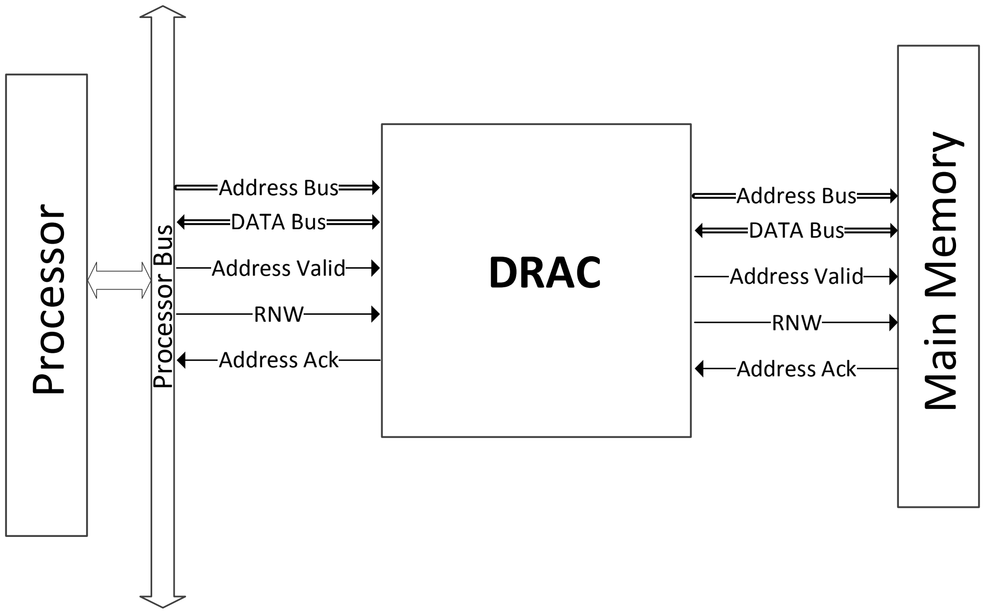


Figure 2.1 DRAC interface and its input and output signals

Figure 2.1 shows the interface between DRAC and the processor bus. Since DRAC actively interacts with the modeled system, it is designed as a dual port peripheral. The input and output signals on the both sides are *Address Bus*, *Address Valid*, *Read/Write*, and the *Address Acknowledgement*. For any read or transaction, the processor put the desired address on the *Address Bus* and set the *Address Valid* signal high to inform DRAC that there is a request from the processor. DRAC is sensitive to *Address Valid* signal on the processor side; whenever detecting the request, it initiates the fetch process. The fetch process starts by setting *Address Valid* signal on the main memory side high, and putting the regarding address on the *Address Bus*. The main memory performs the memory transaction and set the *Address Acknowledgment* signal high, saying the transaction is over. When DRAC detects the *Address Acknowledgment* signal from the main memory, it sets the *Address Acknowledgment* signal on the processor side high to inform the processor the data is written or fetched to/from the main memory.

2.2 DRAC Architecture

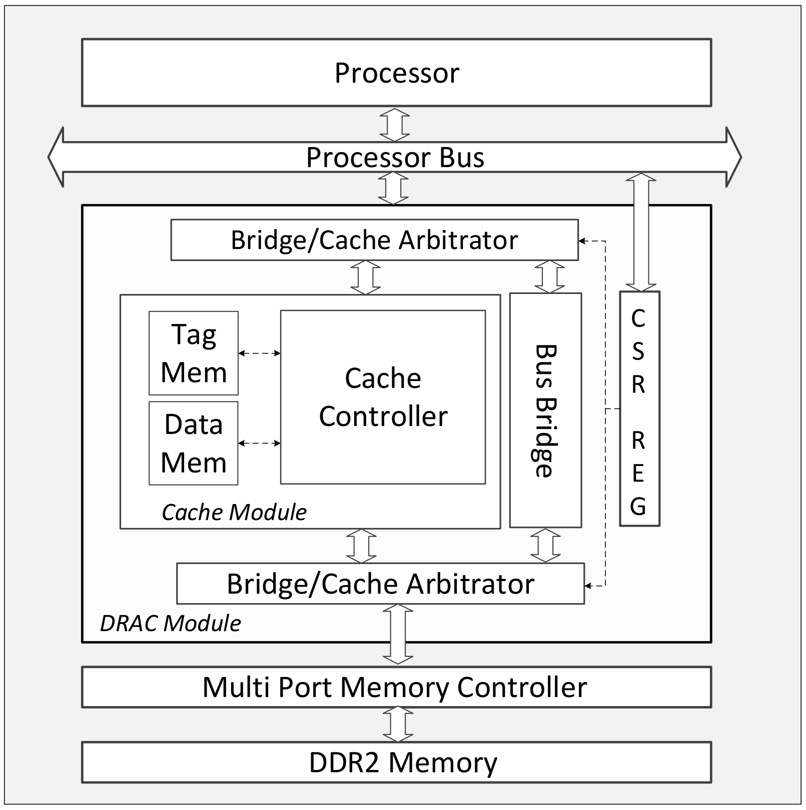


Figure 2.2 Top level design of DRAC

Figure 2.2 presets the top level architecture of DRAC. DRAC consists of several modules such as Bridge/Cache Arbitrator, Bus Bridge, Control Status Register, and tag and data memory. It connects to the processor from one side, and to the main memory on the other side.

2.2.1 Control Status Register (CSR)

CSR is a 32 bit control register that resets, enables/disables, sets the size of the cache, and changes the mode of DRAC to swap mode (swap will be discussed in the next chapter). This register is the controlled by the processor. In every clock cycles, DRAC always checks this register to set its status. Since it is possible to write a value into CSR register during program execution time, the configuration of DRAC can be changed on program run-time. DRAC is sensitive to any change of CSR value; depending on CSR value, DRAC changes its status.

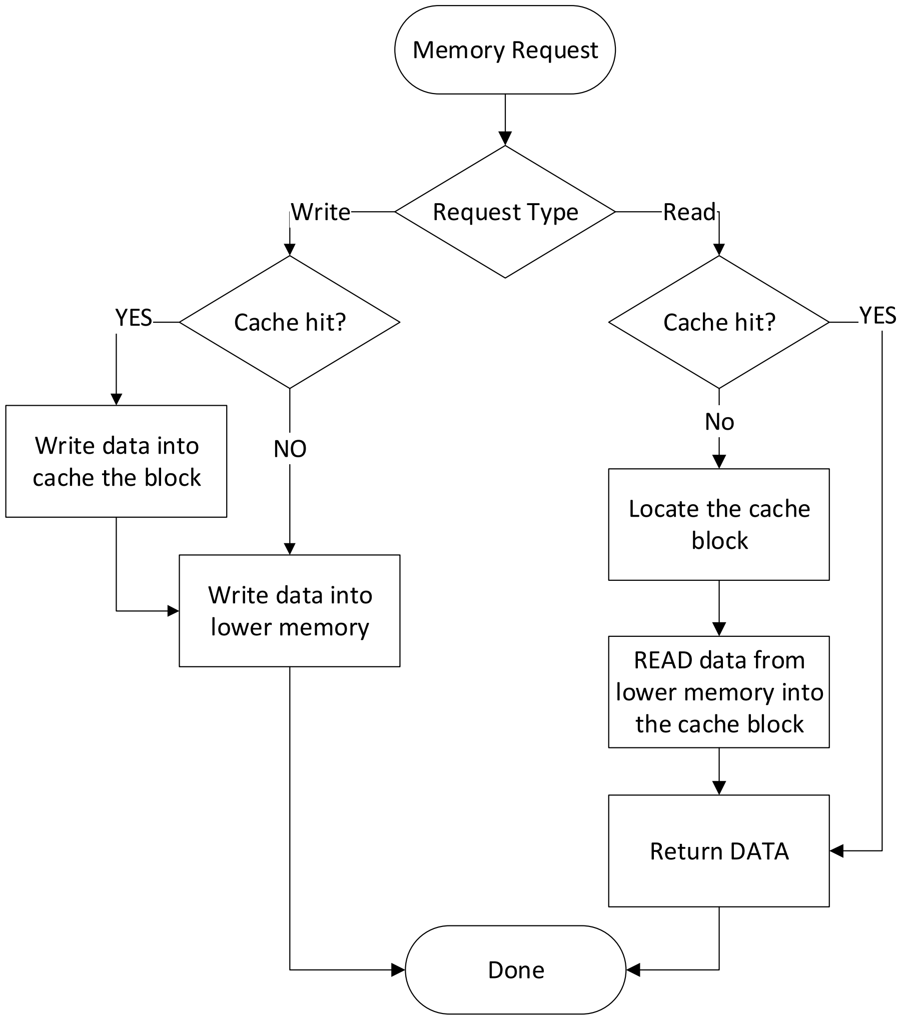
2.2.2 Bridge/Cache Arbitrator & Bus Bridge

DRAC is placed between the processor and the memory controller; therefore, all memory transactions are through DRAC. The active behavior of DRAC requires it to have an extra module beside the cache, which is called Bus Bridge. This module is responsible for establishing the connection between the processor and the off-chip DDR memory when the cache is inactive. The transactions received by the bridge are decoded on the processor side of the bridge. The Bus Bridge, then, generate the necessary sequence of signals to perform the transaction on MPMC side.

The Bridge/Cache arbitrator dedicates the processor bus to the Cache Module or the Bus Bridge according to CSR value. The arbitrator multiplexes the address bus, data bus, and controlling signals between the Cache Module and the Bus Bridge. When the cache is disabled, the cache module is bypassed and the bus is dedicated to the Bus Bridge. Since the processor has to have a direct access to the main memory at the system start-up, DRAC is on bridge mode by default. Once the proper value is written to the CSR, the arbitrator assigns the bus to the Cache Module; Thereby, the cache takes care of memory transactions.

2.2.3 Cache Module

Cache Module is mainly composed of a controller and two Block RAMs as data and tag memory.



Memory Request
Request Type
Write
Read
Cache hit?
YES
NO
Cache hit?
YES
No
Write data into cache the block
Write data into lower memory
Locate the cache block
READ data from lower memory into the cache block
Return DATA
Done

Figure 2.3 Flow chart of the cache with write through policy

Figure 2.3 presents the flow chart of the cache operation. DRAC is supposed to be a direct mapped, 4-word cache block size, with write through policy. In every write and read request, first the tag memory is checked; if the data is in the cache, it is a hit case, otherwise it is a miss. Since the writing policy is write through, the data is updated both in the cache and the main memory on write hit cases. On write miss cases, the data will be written directly in the main memory. In case of a read miss, the corresponding memory block in the main memory will be fetched and returned to the processor. On a read hit, the corresponding data will be delivered from the cache data memory.

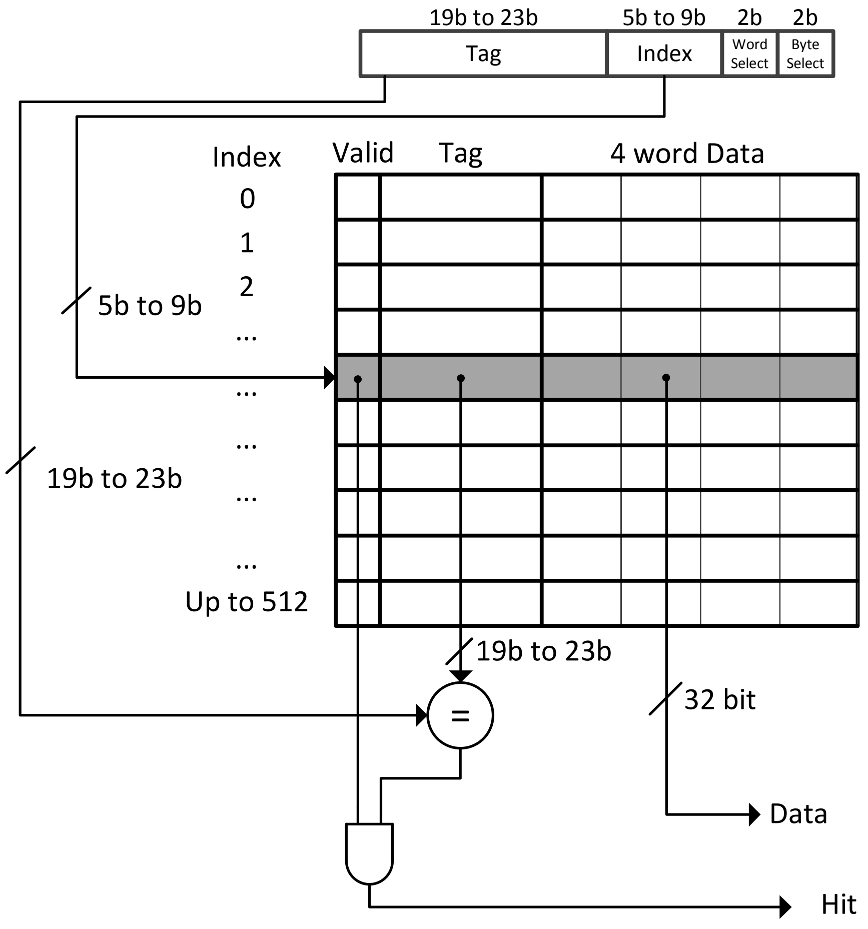


Figure 2.4 Data retrieval process in DRAC

Figure 2.4 shows the data and tag memory that holds up to 2048 words or 8K Byte; memory address is 32 bits. When a memory request is generated, the tag from the cache is compared to the most significant bits of the address to determine whether the entry is in the cache or not. If the tag and the most significant 20 bits of the address are equal, it is a hit case and data is returned to the processor; otherwise, the memory block, which consists of 4 words, is fetched from the main memory. Since DRAC is supposed to be direct mapped cache, there is no replacement policy in the cache design.

Data and tag memory are made up on chip BRAM on the FPGA. DRAC is designed as a size variable cache. It can be set to five different cache sizes: 256B, 1KB, 2KB, 4KB, and 8KB. Depending on CSR value, DRAC can be set to each one of the configurations. To have different size configurations on run-time, we dedicated 16KB of BRAMs for data and tag memory; we, then, utilize a part of the BRAMs as per the cache size requirement. It is the responsibility of the cache controller responsibility to assign allocate amount of BRAM for different cache sizes.

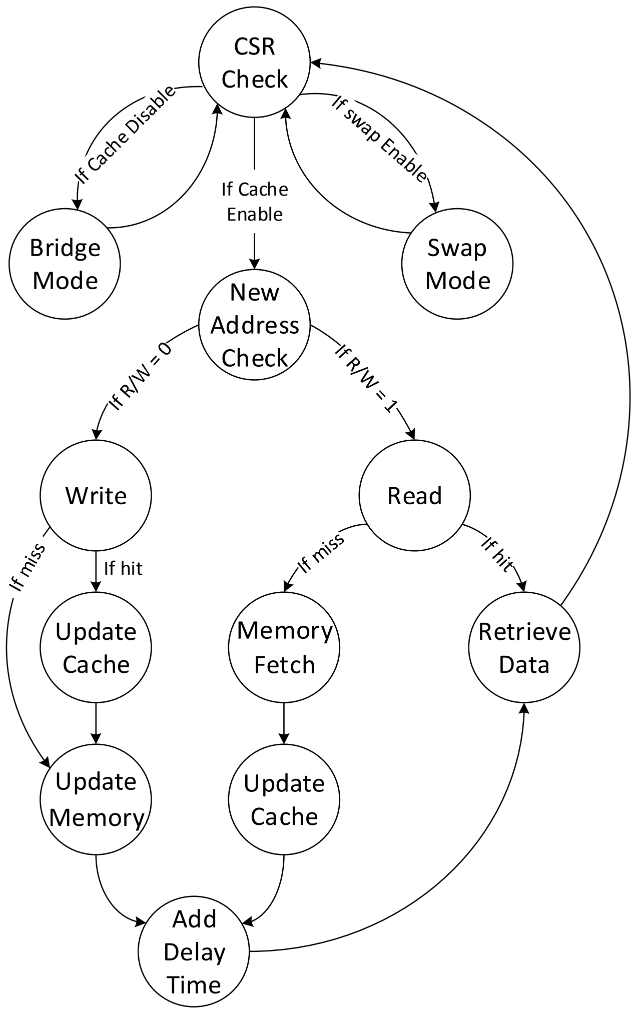


Figure 2.5 Finite state machine of cache controller

The cache controller is key module of DRAC, which is used in both data and instruction cache models. The only difference between the data and the instruction cache is read-only. In order to simplify DRAC design, we used the same controller for both instruction and data. Figure 2.5 demonstrates the cache controller's finite state machine (FSM).

Cache Controller always checks the CSR value before any memory transaction. If CSR is set to cache enable, the FSM in cache controller is triggered and the state is changed to address check. In this state, the module checks the Address valid Bit (Addr_valid) signal on the bus in every clock cycle; if it is detected, then the controller checks R/W signal and goes to Read or Write state. In both Read and Write states, the cache module first checks the tag memory in order to locate the memory block in the cache. In the read state, if the data is found in the cache, it is a hit case, and the cache retrieves the data from its own data memory to the processor; otherwise, it is a miss case and the cache should fetch the regarding memory block from the main memory. The controller's last state is Add Delay Time. This state inserts delays depending on our timing model. The algorithm will be discussed in next section. DRAC is assumed to be a write-through cache. Hence, in the case of write, it updates both the cache and the main memory. At the end of each transaction, DRAC sets the acknowledgment signal in the processor's bus, to inform the processor the memory transaction is done.

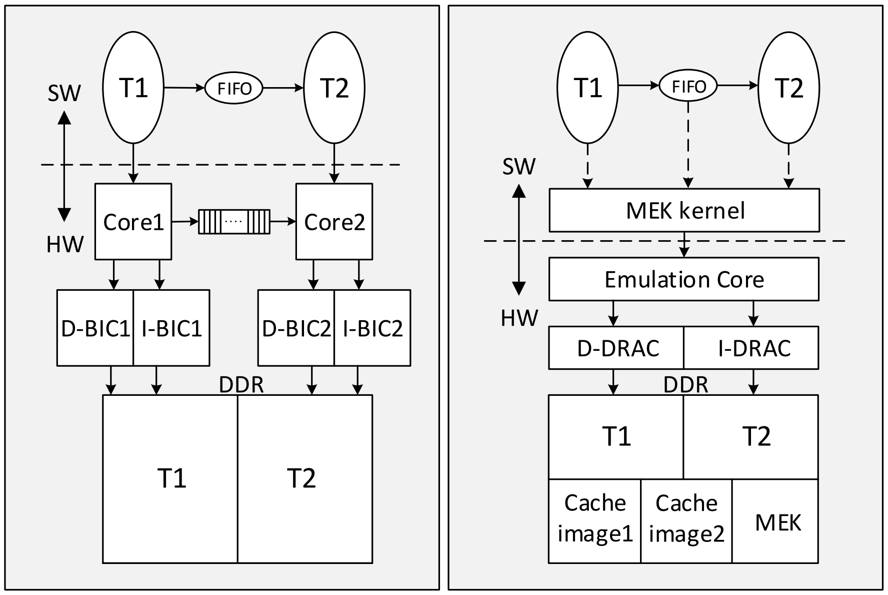
2.3 DRAC Features

DRAC active behavior speeds up the simulation process by providing instruction and data for the processor in repetitive accesses. The parametric design of the model makes it generic enough to work with most buses. The run-time configurability makes it flexible enough for the embedded designer to quickly explore different design options and to change the cache size during program execution time.

CHAPTER 3

DRAC Extension for Hybrid Prototyping

The DRAC model includes the functionality of a standalone cycle-accurate data and instruction cache, and additional logic to support multicore hybrid prototyping. The hybrid prototyping technique simulates multicore system using an emulation kernel on top of a single physical instance of a core. Thus, a single cache that is capable of switching its context over different virtual cores is needed. To realize this concept, an extra module has been implemented in the cache that can swap the cache contents across different virtual cores. Each virtual core's cache can be configured independently; however, this requires DRAC to change its configuration during run-time. The run-time configurability of DRAC provides the emulation kernel to change the configuration of the cache.



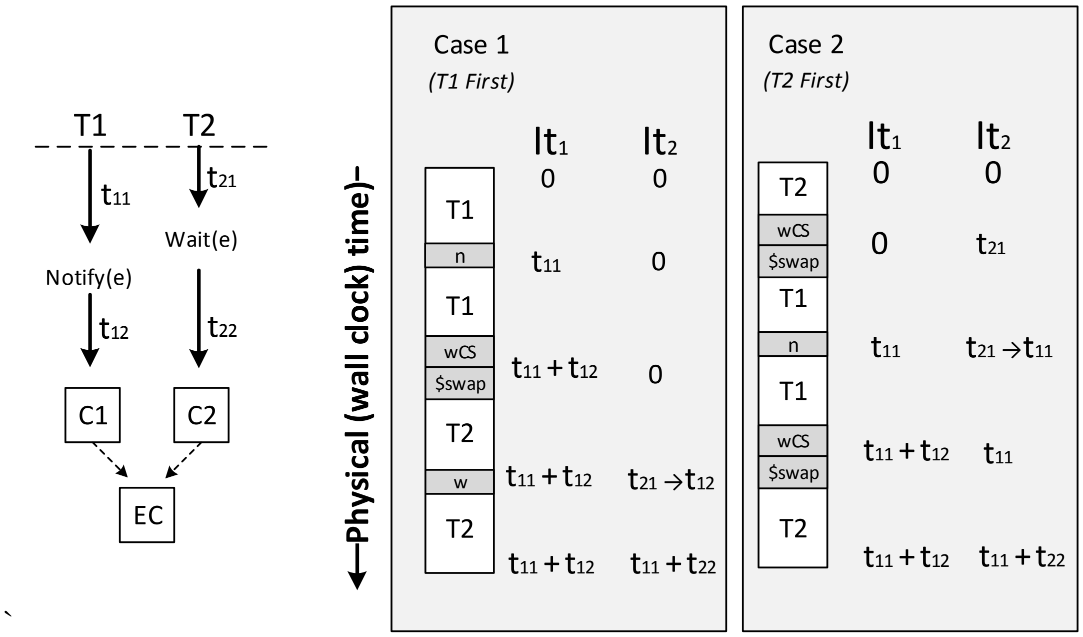
(a) A multicore design (b) Equivalent hybrid prototype

Figure 3.1 Hybrid prototype structure for a two core design

3.1 Methodology

Figure 3.1 presents the layered structured of a multicore design and its hybrid prototype. In the target design, which is shown in Figure 3.1 (a), T1 and T2 are tasks running on separate cores. Each core has its own L1 cache and separate memory space on DDR. The communication between the cores is performed using FIFO-based communication channels. Hybrid prototyping introduces an additional software layer on top of an emulation core. Figure 3.1 (b) illustrates the hybrid prototype that incorporates the MEK. The emulation core and the main memory in hybrid prototyping are of the same type as that used in the multicore design. However the built-in caches have been replaced by DRAC models. DRAC is customized to support different cache contexts for the two cores. For each cache context, a separate space on DDR is dedicated as cache image. Before the MEK starts emulating a core, it loads the corresponding cache image from DDR to DRAC. Similarly, after the MEK stops emulating a core, it saves the corresponding cache image to DDR. Hence the cache images are swapped in DRAC, when the MEK switches from one core to another.

Hybrid prototyping offers the designer to develop virtual platforms over the real-world connected FPGAs. The MEK platform created based on hybrid prototyping technique. It emulates the execution of any multi-tasking C/C++ application on a single core design. The MEK provides a simple environment for the embedded designers to test and explore their design without the knowledge of multiple cores configuration or the data path among the cores. The accuracy of hybrid prototyping is 100% since the application is running on the same core as it is targeted for.



(a) Emulation of Tasks on two different cores (b) Possible emulation schedules

Figure 3.2 Simple example of simulation with MEK

The MEK layer dynamically schedules multiple tasks and simulates the execution time of a full FPGA multicore system. The MEK and DRAC model support the context switch among different tasks. The emulator dynamically saves the context *(program, stack pointers, registers, and state of the instruction and data caches)* of the yielded task and loads the context of the active task. During kernel call the cache is completely disabled and is not polluted by the emulator. Figure 3.2 (a) illustrates the execution behavior of two tasks running on a 2core design. Task T1 executes for t_{11} time, notifies the global event *e*, and continues the execution to the end for t_{12} time. Meanwhile, C2 executes task T2 for t_{21} time, and waits for the global event *e*. After T2 gets notified, it continues execution for t_{22} period and terminates. Both cores are simulated on an emulation host core (*EC*) which is the same type as c1 and c2.

Figure 3.2 (b) shows two possible simulation schedules on EC. A task may be in four possible states: RUNNING, READY, BLOCKED or TERMINATED. The MEK maintains the logical times, lt_1 and lt_2, on C1 and C2, respectively. The logical time for a core is the time until which the core has been simulated. At logical time 0, the MEK may pick either C1 or C2 to simulate first. If the MEK schedules C1 to be simulated first, it runs T1 on EC until *e* is notified. The MEK saves the event's notification and its logical timestamp t_{11}. Since event notification is non-blocking in a discrete event model, the MEK allows T1 to execute until it is terminated. Then, the MEK does a context switch (CS). During CS the contents of the data and instruction cache is saved in the main memory. Since it is the first CS, the cache is flushed and C2 runs T2 from its logical time 0 until it reaches *wait(e)* at logical time t_{21}. At this point the MEK checks for any notifications of e that were made after logical time t*21*. Indeed, since $t_{11}>t_{21}$, the MEK finds that *e* was notified by T1 before T2 executed *wait(e)*. Therefore, the MEK updates the logical time of C2 to t_{11} to model T2 being blocked on the wait from t_{21} to t_{11}. Finally, T2 is resumed and runs to completion.

If the MEK schedules C2 to be simulated first (Case 2), it runs T2 on EC from C2's logical time 0 until it reaches *wait(e)* at C2's logical time t_{21}. Since no notifications of e are found, the MEK stores the wait on *e* with timestamp t_{21}, and blocks T2. It then does a context switch from C2 to C1. It saves T2 cache contents in the main memory and flush the cache since T1 is not been started yet. To emulate C1, the MEK runs T1 from C1's logical time 0 until the notification of e at C1's logical time t_{11}. Upon notification, the MEK checks if there are any pending waits on *e* at or before logical time t_{11}. Indeed, task T2 is blocked since C2's logical time t_{21} ($< t_{11}$) on e. Therefore, the MEK unblocks T2 and updates C2's logical time to t_{11} in order to account for the blocking time. The MEK continues simulating C1 until termination of T1, followed by a context switch to C2. The kernel swaps T1 cache with T2 cache contents. The kernel saves T1 cache contents, and loads T2 data and instruction cache images from the main memory. MEK continues C2 simulation until termination of T2.

3.2 Architecture

In order to realize the swap we implanted another module beside the cache controller, called Swap Controller.

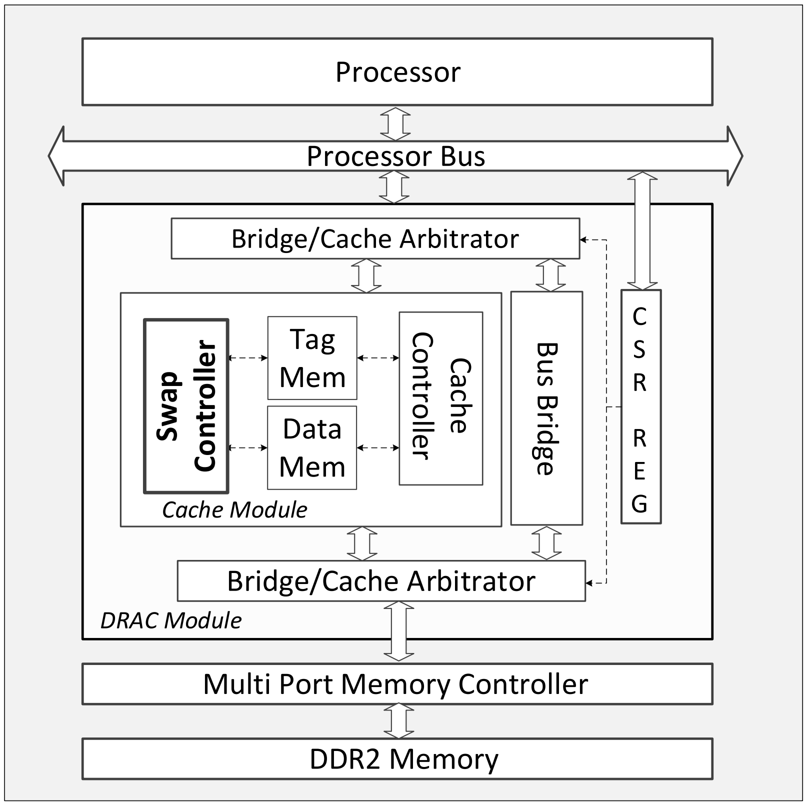


Figure 3.3 Modified design of DRAC including Swap Controller

Figure 3.3 presents the design architecture of DRAC with the Swap Controller added to the design. Similar to the cache controller, swap controller has also access to the tag and data memory. The swap controller is responsible for switching the cache context among different cores. It is also responsible for stalling the processor during the swap transaction. The swap controller is activated by writing the proper value into CSR register. When the swap is activated by the processor, the swap controller locks the data and tag memory and does not allow the processor pollutes the cache contents.

3.2.1 Swap Module

The cache swap feature is the ability of the cache to save a copy of itself on the off-chip DDR memory and to load it later automatically. The swap module is responsible for switching the cache context from one core to another during run-time. Whenever a swap is triggered by the MEK, DRAC stalls the processor and saves the current cache context to the main memory, line by line. The cache context of the next core to be simulated is, subsequently, loaded.

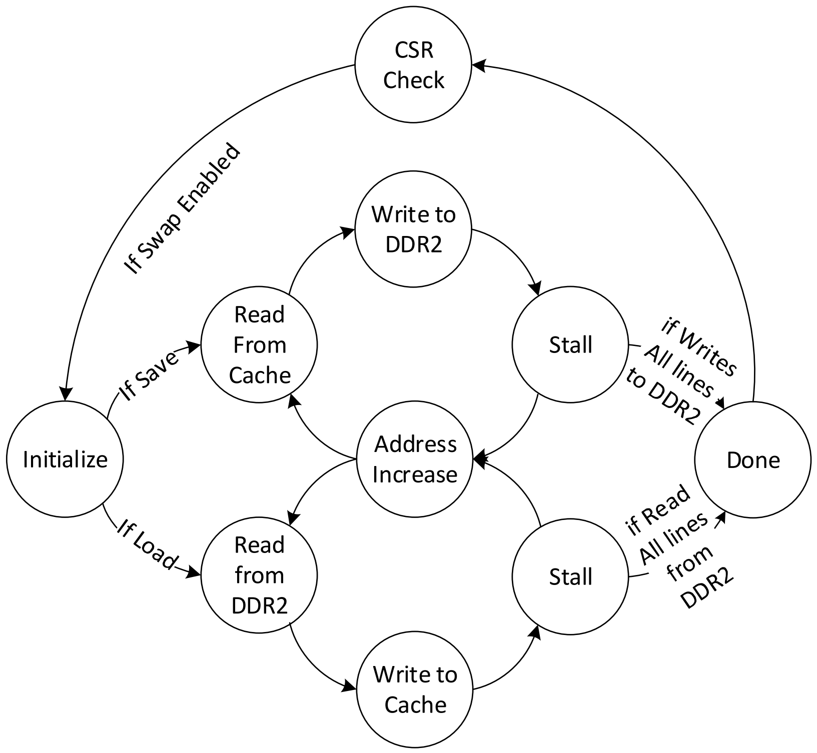


Figure 3.4 FSM of Swap Controller (Swap Mode)

Figure 3.4 illustrates the finite state machine of the swap controller. Similar to the cache controller, the swap controller has a *CSR Check* state as an initial state. The swap trigger is detected in this state. Whenever MEK requires the cache to be swapped, it writes a certain value, which will be explained in the next section, into the CSR to start the process. Depending on the CSR value, the controller will save or load the cache state. As explained earlier, space in the DDR memory has been allocated for each core, depending on the cache size. The. *Initialize* state determines the starting and the ending address locations of each core's cache.

If the processor issues the save cache state command, the controller goes to *Read from Cache* state, reads the first line of the cache, and writes it into main memory. It continues this process until all cache lines are written to the main memory. On the other hand, if the processor's command is load, the controller goes to *Read from DDR2* state, reads all previously saved contents of the cache from main memory, and writes them into the cache. There is a *stall* state in swap's FSM that ensures the data is safely resided in the cache or the main memory.

Some processors, like MicroBlaze, do not support sleep or idle mode. Hence, we stalled the processor in another way. Since DRAC is used as instruction cache and lies between the processor and the main memory, it is possible to give any desired instruction to the processor instead of real instructions. Therefore, whenever swap is enabled and the processor is requesting instructions during swap, we give the machine code of relative branch to the same Program Counter (PC).

$$PC \leftarrow PC + 0$$

As long the cache is operating the swap action, the processor is jumping to the same PC, so it seems the processor is stalled. Once DRAC swap all the current cache contents with the next cache contents, the swap operation is done and emulator goes back to the normal operation. It gives the real instructions to the processor, and the program continues the execution.

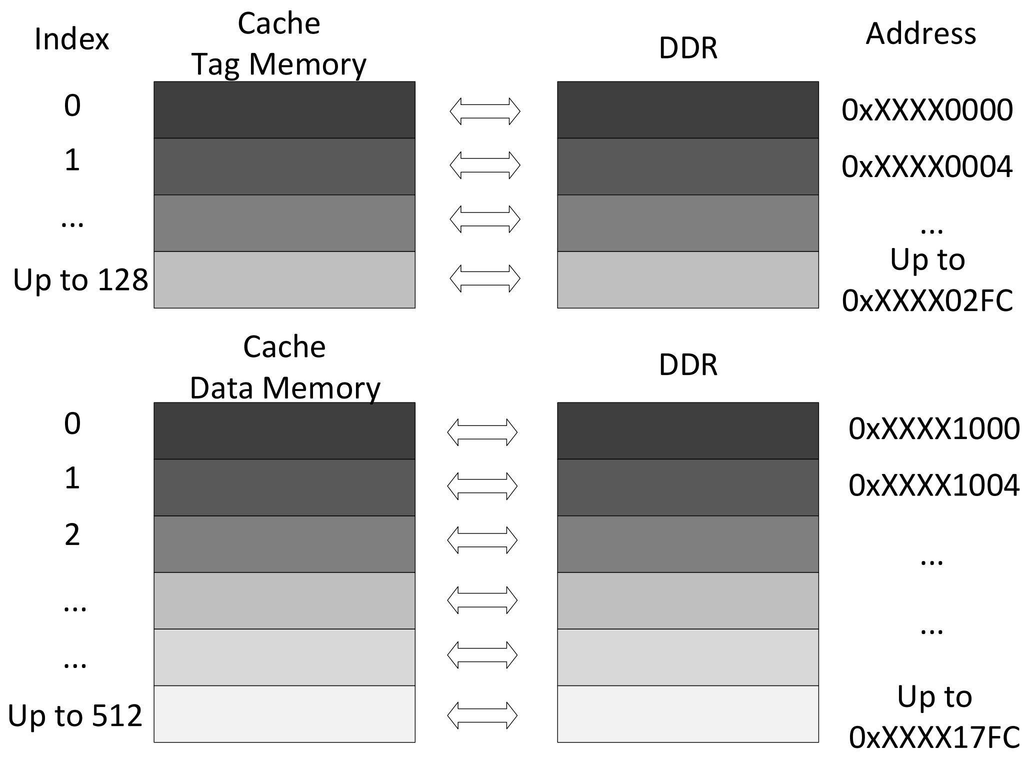


Figure 3.5 Process of a save or load transaction between cache and DDR

Figure 3.5 shows the schematic view of a load or save transaction and their addresses on the DDR. In the save case, first the tag memory will be written to the main memory from the starting address of 0xXXXX0000 to 0xXXXX02FC; subsequently, the data memory from will be saved from the starting address of 0xXXXX1000 to 0xXXXX17FC in the 8kB cache size case. The ending addresses will be changed depending on the cache size. In the load case, the cache contents will be loaded from DDR from the starting to the ending address. DRAC supports different caches of the virtual cores. For each core we dedicated a separate address space. The next core space on the DDR starts from 0xXXXX2000 address to 0xXXXX37FC, and so on.

3.3 DRAC Software Driver

As explained earlier, the MEK platform is a software layer on top a physical processor. Since the MEK should be able to control the cache, we developed a software driver for the DRAC. MEK is able to communicate with DRAC through writing the proper value into the CSR. As a result, we created simple macros for the MEK to control the cache.

Listing 3.1 DRAC Driver

```
#define enable_cache {\
        XIo_Out32(Cache_CSR_Address ,Cache_Enable_Value);\
}
#define disable_cache {\
        XIo_Out32(Cache_ CSR _Address,Cache_Disable_Value);\
}
#define reset_cache {\
        XIo_Out32(Cache_ CSR _Address,Cache_Reset_Value);\
        disable_cache\
}
#define Cache_Swap(c1, c2) {\
        save_cache(c2);\
        load_cache(c1);\
        reset_cache;\
}
```

Listing 3.1 presents the Enable, Disable, Reset, and swap macros that MEK uses for commanding DRAC.

Listing 3.2 Save Macro

```
#define save_cache(c1){
        if(c1 == 0)
                XIo_Out32(Cache_ CSR _Address,Core1_Save_Value);
        else if(c1 == 1)
                XIo_Out32(Cache_ CSR _Address,Core2_Save_Value);
        else if(c1 == 2)
                XIo_Out32(Cache_ CSR _Address,Core3_Save_Value);
        else if(c1 == 3)
                XIo_Out32(Cache_ CSR _Address,Core4_Save_Value);
        else if(c1 == 4)
                XIo_Out32(Cache_ CSR _Address,Core5_Save_Value);
        XIo_Out32(Cache_ CSR _Address,Swap_Trigger _Value);
        asm("nop");
        asm("nop");
}
```

Listing 3.3 Load Macro

```
#define load_cache(c1){
        if(c1 == 0)
                XIo_Out32(Cache_ CSR _Address,Core1_Load_Value);
        else if(c1 == 1)
                XIo_Out32(Cache_ CSR _Address,Core2_Load_Value);
        else if(c1 == 2)
                XIo_Out32(Cache_ CSR _Address,Core3_Load_Value);
        else if(c1 == 3)
                XIo_Out32(Cache_ CSR _Address,Core4_Load_Value);
        else if(c1 == 4)
                XIo_Out32(Cache_ CSR _Address,Core5_Load_Value);
        XIo_Out32(Cache_ CSR _Address,Swap_Trigger_Value);
        asm("nop");
        asm("nop");
}
```

Table 3.1 - Swap save and load variable and values

Variable	Value	Variable	Value
Cache_Enable_Value	*0xaaaaaaaa*	*Cache_Reset_Value*	*0x33333333*
Cache_Disable_Value	*0x00000000*	*Swap_Trigger_Value*	*0x11111111*
Core1_Save_Value	*0x00000000*	*Core1_Load_Value*	*0x00000001*
Core2_Save_Value	*0x00000002*	*Core2_Load_Value*	*0x00000003*
Core3_Save_Value	*0x00000004*	*Core3_Load_Value*	*0x00000005*
Core4_Save_Value	*0x00000006*	*Core4_Load_Value*	*0x00000007*
Core5_Save_Value	*0x00000008*	*Core5_Load_Value*	*0x00000009*

Listing 3.2 and 3.3, present the load and save macros of DRAC. Table 3.1 defines the values that should be written into CSR to set the status of the cache. In each of save or load functions, first the number of virtual core for the cache is defined, after, the swap is triggered by writing the value *Swap_Trigger_Value* into the CSR. There are also two no operation (nop) assembly commands in the last lines of load and save macro. DRAC put the processor in the nop command loop until the swap is done.

CHAPTER 4

Timing Model

This research is an effort to estimate the execution time of a program on an embedded system with built-in instruction and data cache. This execution time is mainly depended on the behavior of the cache, data link interconnections, and off-chip memory. DRAC is designed as an active and parameterized cache model. Since DRAC actively interacts with the modeled system, it is possible to model the behavior of the built-in cache, interconnections, and DRAM as the main memory, all in a cache model. In this chapter, we present timing model of the system, using the example of a MicroBlaze based system [27].

4.1 Bus Characteristic

Bus is the communication system that transfers data between different components of the system. Each bus has certain set of rules, governing how it works; these rules are called bus protocol. The bus protocol includes the specification of the bus, and all attached peripherals must obey the protocol. Two different buses have been used in our emulation system. The target system utilizes XCL bus, and the modeled system uses PLB in order to establish the communication among the processor, the cache, and the main memory. These two buses have different characteristics that affect the execution time of the program. For example, XCL is a point to point connection while PLB is a shared bus that supports multiple master and slave devices. In following subsections, PLB and XCL bus protocols will be introduced. We used ChipScope Pro [28] bus analyzer to observe the memory behavior and obtain the memory parameters in all the experiments.

4.1.1 PLB Bus

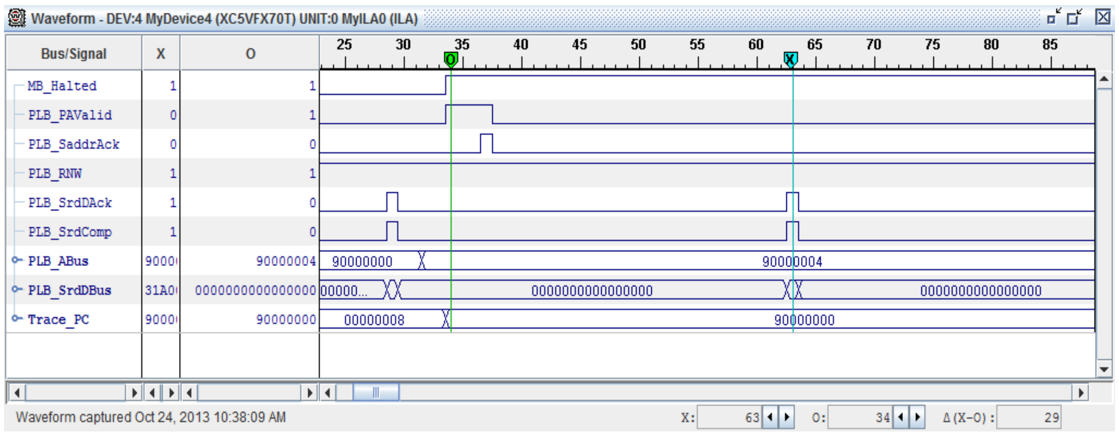


Figure 4.1 Read operation from off-chip DDR memory via PLB

PLB is a high-performance bus interface that is used to access data. Figure 4.1 presents a read request transaction from the processor to the main memory over PLB bus. The address cycle request has three phases: request, transfer, and address acknowledgment. When a master device requests a data transaction from the main memory (point O on Figure 4.1), *PLB_PAValid* signal goes high, showing there is a memory access request. At the same time *PLB_RNW* signal goes high to indicate the request is a read request. In the second phase, the main memory set *PLB_SaddrAck* (slave address acknowledgement) signal high for a single cycle to show it has received the memory request and is processing that request. After certain amount of time, at the point of X of Figure 4.1, the data is delivered to the processor by setting PLB_SrdDAck (slave read acknowledgment) signal high for a single cycle. The duration of a read request over the PLB (the time interval between point O and point X in Figure 4.1) without any cache in between, is 29 cycles. The main reason for this delay is the column address latency that is imposed by the main memory. Similar to the read request the write request has also the same three phase as the read request. This latency for a write transaction over the PLB is 11 cycles.

4.1.2 XCL Bus

XCL bus is a high performance FSL FIFO based point to point data link that provides the direct access of the processor the main memory. This interface is available for the MicroBlaze processor when using built in cache.

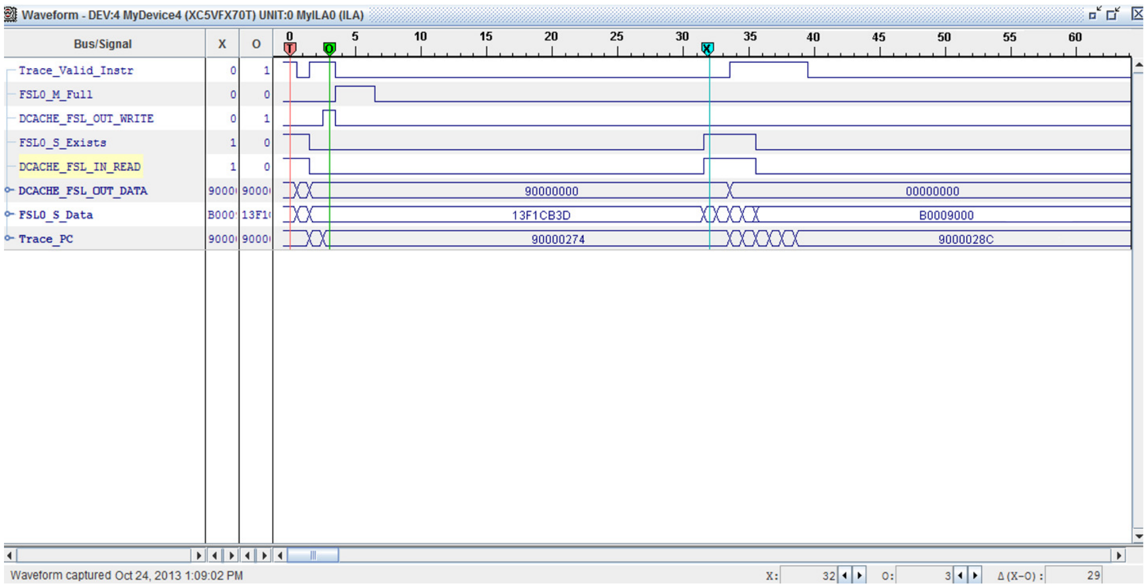


Figure 4.2 Read miss latency with built-in cache via XCL

Figure 4.2 presents a data read request transaction from the cache to the main memory over XCL. XCL can handle 4 or 8-word cache lines during each fetch. In read case, the information is requested by raising *DCACHE_FSL_OUT_WRITE* signal for a single clock cycle. When the data is ready, the main memory raises the *FSL_S_Exists* high to show data exits on the *FSL_S_Data*. In write through policy, the communication protocol over XCL is IXCL and DXCL. According to this protocol, each cache line is expected to start with the critical word first and 3 words follow the first word. Each write in this policy results in a write over cache link regardless of existence of this data in the cache. The write to the main memory is complicated since there is a buffering policy in the main memory. In following sections, the write will be investigated in more detail.

4.2 Cache Modeling

DRAC is designed as an active cache emulator. It actively interacts with the modeled system and estimates the execution time of a program. DRAC saves the data as well as tag on BRAM implemented on the FPGA. This behavior reduces the execution time of programs as compared to the system without cache. Since the built-in cache is using XCL bus for communication and this bus is optimized for the built-in cache, the execution time of the program with built-in cache is faster than the same program with DRAC. It must be noted, our concern is not only the simulation speed, but also the accuracy of the timing estimation.

As discussed in the section 3.2.3, the cache controller has an *add delay time* state that models the timing. In order to model the built-in cache, we add extra cycles to certain DRAC transactions such that all the DRAC delays are a multiple of corresponding built-in cache delays, by the same factor. As a result, the program's execution time, when using DRAC, will be a multiple of the execution time with built-in cache. For example, if a processor with built-in cache executes a program in *x Clock cycles*, the proposed model will execute the same program in $n \times x$ *Clock cycles*, which "n" is the linear scaling factor:

$$\textit{Modeled Clock cycles} = n \times \textit{Real Clock cycle}$$

For hybrid prototyping, we excluded built-in caches, and used DRAC instead. DRAC is designed as a master IP core that can be utilized as data or instruction cache. Since DRAC is simply a peripheral to MicroBlaze, it is connected to processor local bus (PLB). Being a peripheral, enables DRAC to connect to different processor types by some changes in the interface.

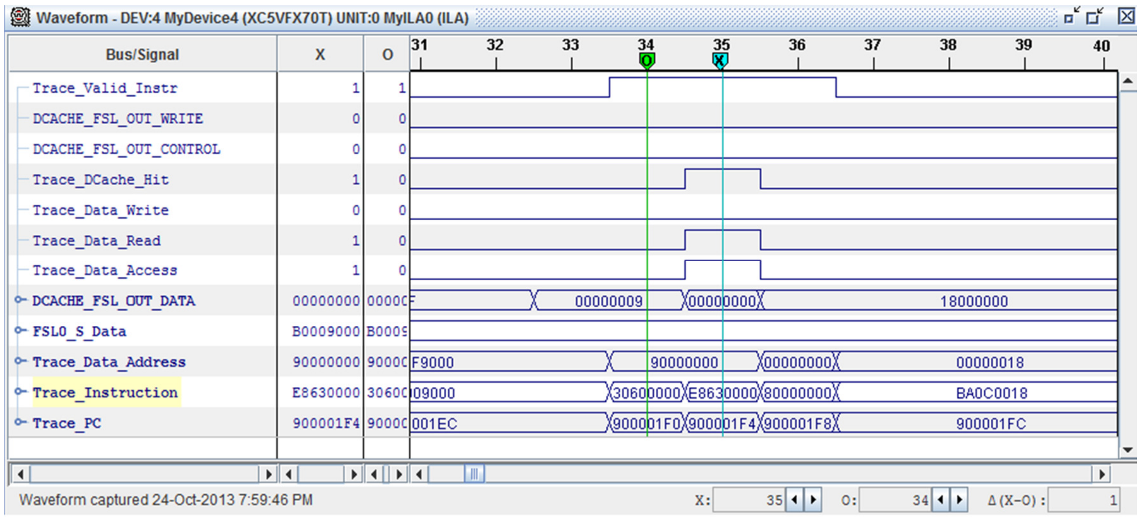


Figure 4.3 Read hit latency with built-in cache

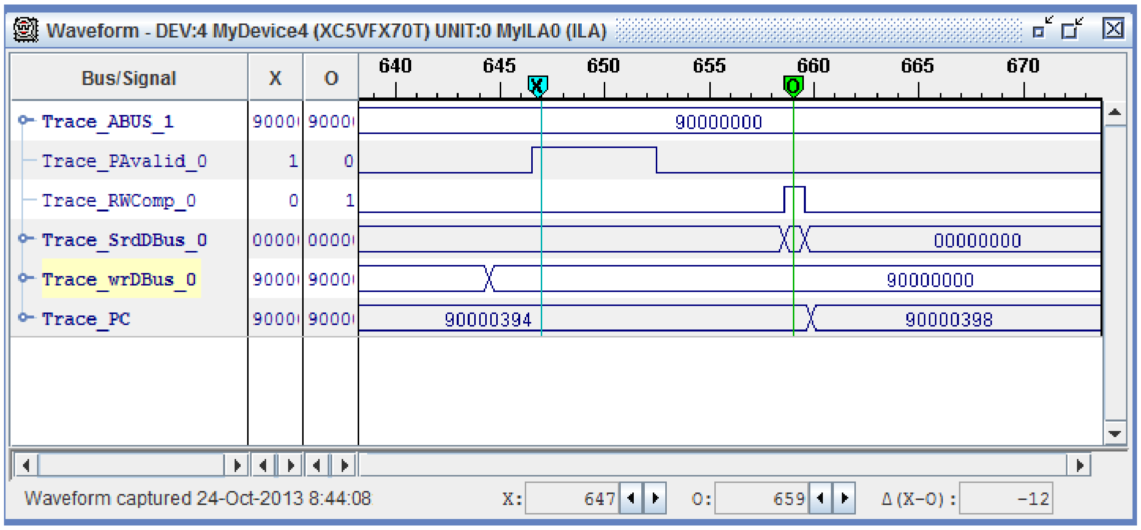
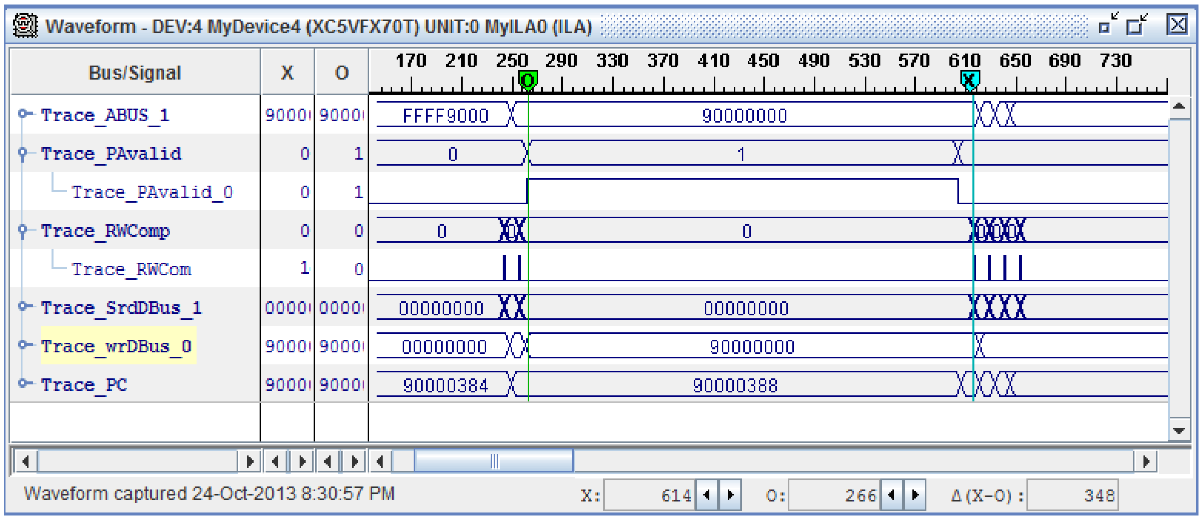


Figure 4.4 Read hit latency with DRAC model

Figure 4.5 and 4.6 present the snap shot of a hit latency in built-in cache and DRAC model. As can be seen, the hit time for the MicroBlaze built-in cache is 1 cycle, while this time is 12 cycles for DRAC over PLB. Therefore, we have defined our scaling factor as 12. It means every memory transaction in DRAC will incur 12 times the delay of the corresponding transactions with the MicroBlaze built-in cache.

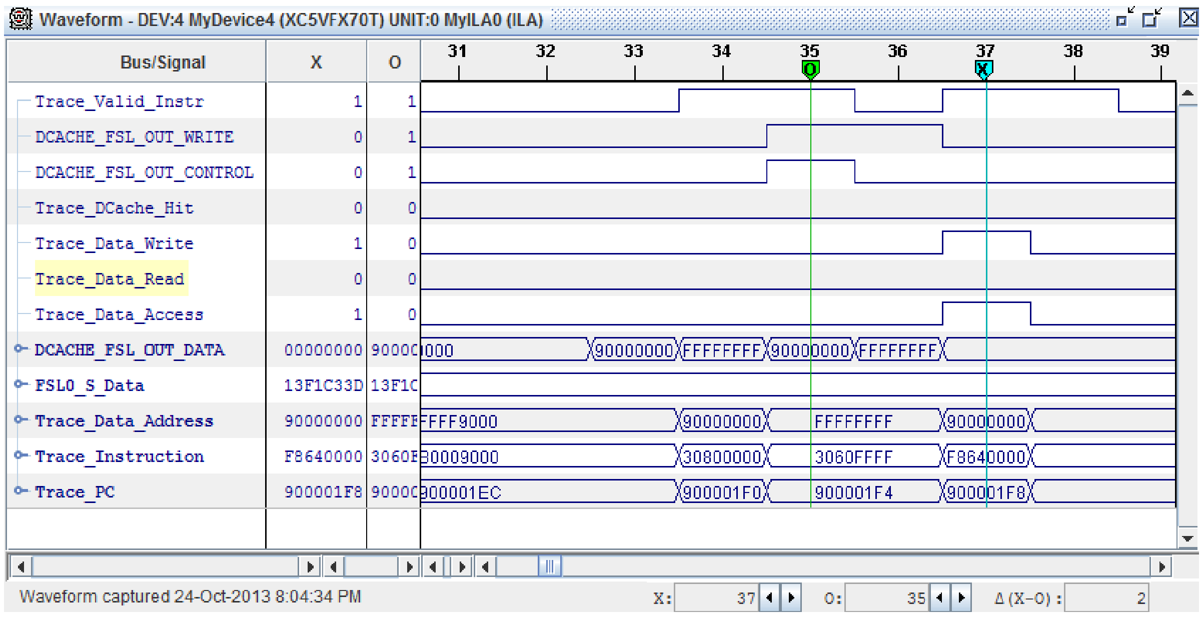

Waveform - DEV:4 MyDevice4 (XC5VFX70T) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
Trace_ABUS_1	9000(	9000(
Trace_PAvalid	0	1
Trace_PAvalid_0	0	1
Trace_RWComp	0	0
Trace_RWCom	1	0
Trace_SrdDBus_1	0000(	0000(
Trace_wrDBus_0	9000(	9000(
Trace_PC	9000(	9000(

Waveform captured 24-Oct-2013 8:30:57 PM X: 614 O: 266 Δ(X-O): 348

Figure 4.5 Read miss latency with DRAC model

The other factor that defines cache performance is read miss time. The average miss time latency for the MicroBlaze built-in cache to bring 4 words of data is 29 cycles, as shown in Figure 4.2. This miss time is 149 cycles in DRAC without adding any extra cycles. In order to model read miss time, the emulator inserts 199 cycles to make read miss latency 12×29 cycles. Figure 4.7 shows the read miss time of DRAC after modeling.


Waveform - DEV:4 MyDevice4 (XC5VFX70T) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
Trace_Valid_Instr	1	1
DCACHE_FSL_OUT_WRITE	0	1
DCACHE_FSL_OUT_CONTROL	0	1
Trace_DCache_Hit	0	0
Trace_Data_Write	1	0
Trace_Data_Read	0	0
Trace_Data_Access	1	0
DCACHE_FSL_OUT_DATA	00000000	9000(
FSL0_S_Data	13F1C33D	13F1C
Trace_Data_Address	90000000	FFFFF
Trace_Instruction	F8640000	3060F
Trace_PC	900001F8	9000(

Waveform captured 24-Oct-2013 8:04:34 PM X: 37 O: 35 Δ(X-O): 2

Figure 4.6 write operation latency with built-in cache

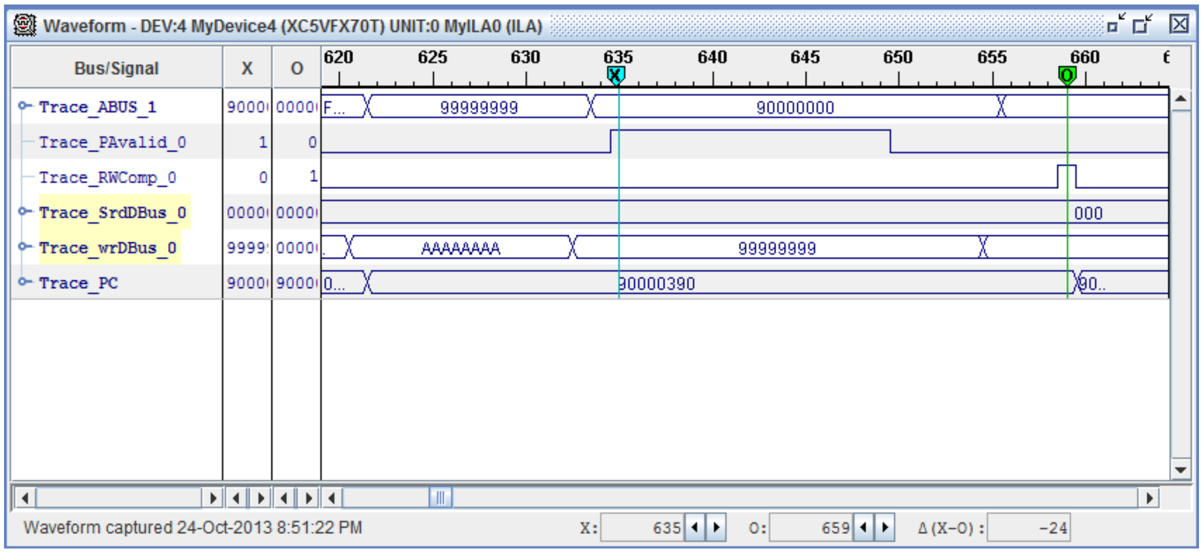


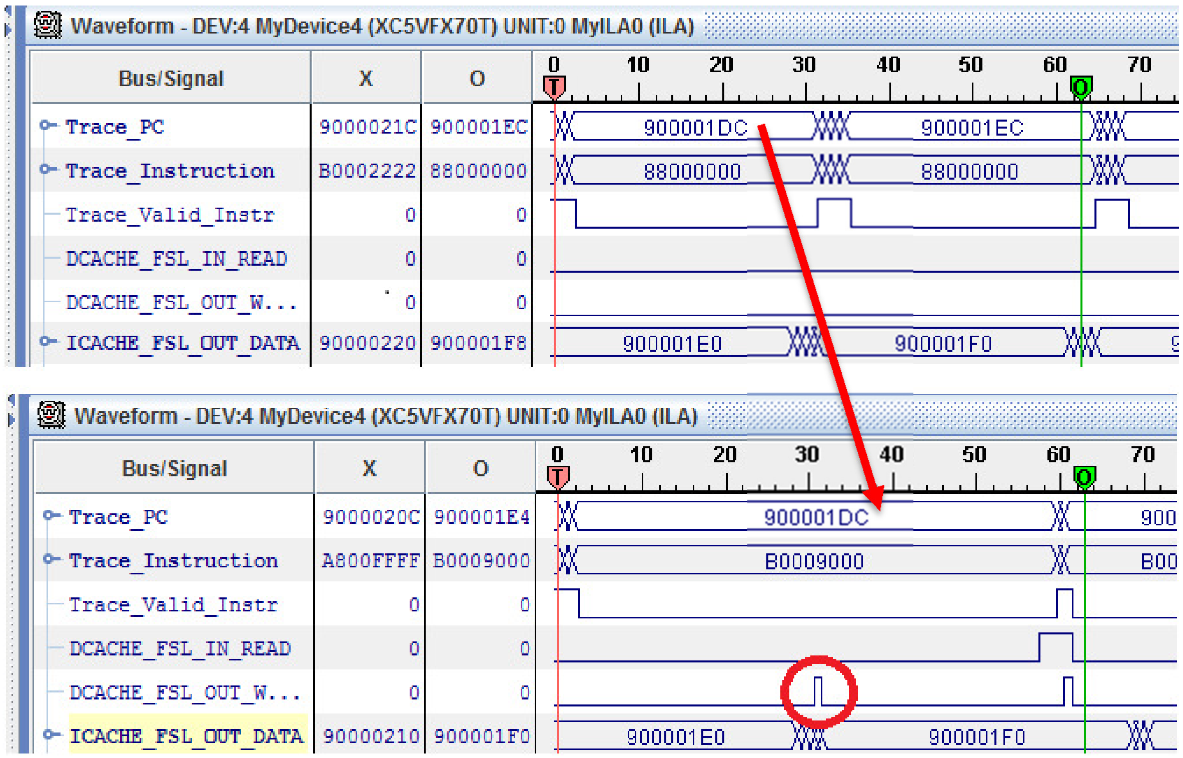
Figure 4.7 write operation latency with built-in cache

The write operation is another factor that impacts the system performance. DRAC models the write-through cache policy. Hence, in every write transaction the main memory will be updated. Figure 4.8 demonstrates a single write operation in the built-in cache, and Figure 4.9 shows the write operation to the same location (0x90000000) in DRAC. A single write operation in the built-in cache takes 2 cycles. This write operation takes 2×12=24 cycles in DRAC after modeling; which is a factor of 12.

Writing into on-chip BRAM is quite simple and predictable; however, the write operation to the main memory, which in our case is DRAM, is quite complex. The complexity comes from the buffers that are implemented in the DRAM memory controller. This complexity is not clear in a single write; however, if there are more than a single write to the main memory, the buffering shows its effect. Therefore, in order to model every write operation scenarios in DRAC, we first need to model DRAM.

4.3 DRAM Modeling

The connection of DDR2 memory to the system is established by Multi-Port Memory Controller (MPMC). MPMC provides separate accesses to the main memory for different modules in the system. It shares single off-chip DDR2 memory between multiple devices. We have two kinds of memory transactions in the system: read and write. The effect of multiple reads from different ports of MPMC is negligible since reading from the memory does not affect the saved data. The write delays behave differently, though. For a write into the main memory, MPMC stalls other memory transactions to make sure that the memory is in a consistent state. Therefore, if there is a write into a port of MPMC, the read or write access time of other ports will increase.


Waveform - DEV:4 MyDevice4 (XC5VFX70T) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
Trace_PC	9000021C	900001EC
Trace_Instruction	B0002222	88000000
Trace_Valid_Instr	0	0
DCACHE_FSL_IN_READ	0	0
DCACHE_FSL_OUT_W...	0	0
ICACHE_FSL_OUT_DATA	90000220	900001F8

Waveform - DEV:4 MyDevice4 (XC5VFX70T) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
Trace_PC	9000020C	900001E4
Trace_Instruction	A800FFFF	B0009000
Trace_Valid_Instr	0	0
DCACHE_FSL_IN_READ	0	0
DCACHE_FSL_OUT_W...	0	0
ICACHE_FSL_OUT_DATA	90000210	900001F0

Figure 4.8 Effect of a single write on the instruction fetch time in built-in cache

Figure 4.10 shows how a single write of the built-in data cache (*DCACHE_FSL_OUT_WRITE*), which is circled in the figure, increases the fetch time of an instruction (*ICACHE_FSL_OUT_DATA*) and the programs execution time (*Trace_PC*). MPMC uses buffering technique in order to reduce the write time latency. In case of a single write, MPMC processes the write transactions in the background while it handles other read accesses. Buffering offers the system a better performance, although it creates irregularity in successive or multiple memory transactions. In case of successive writes into a single port, the write operation time will be different depending on the number of consecutive writes in that port. The first write will take the least, and the last write will take the most operation time. The read time will also be affected by the successive writes of the other port. If the number of consecutive writes increases, the read access time of the other port will also increase.

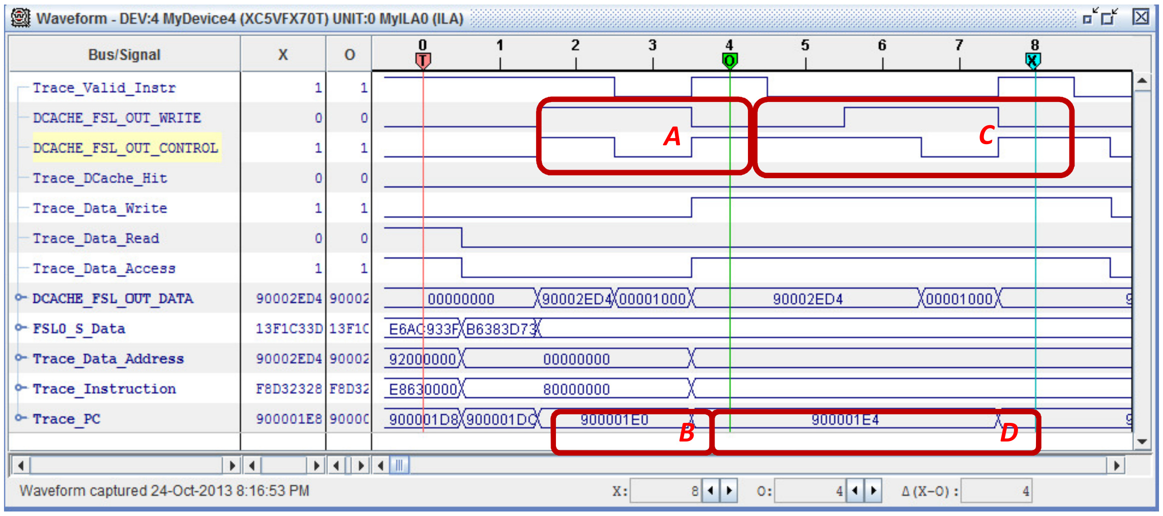


Figure 4.9 Effect of consecetive writes in built-in cache

Figure 4.11 presents two consecutive writes with the built-in cache. Part A and B of the figure indicate the first write operation (part A) and execution (part B) time which takes 2 cycles. Part C and D of the picture shows the write (part C) and exaction (part D) time of the second consecutive write operation, which takes 4 cycles.

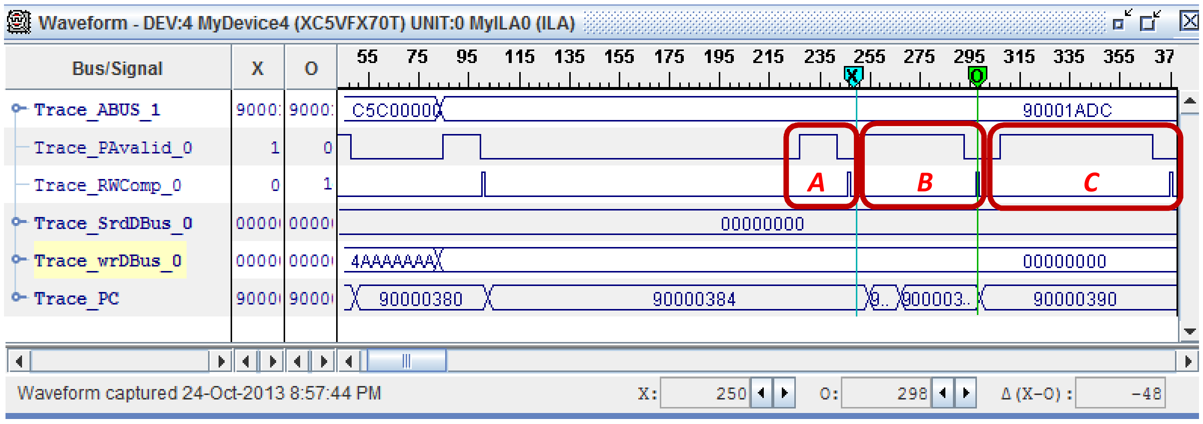


Figure 4.10 Effect of three consecutive writes in DRAC model

Figure 4.12 shows the screen shot of three consecutive writes in DRAC model. The first write (part A) takes 2×12 cycles, the second consecutive write takes 4×12 cycles, and the third consecutive write takes 5×12 cycles. DRAC detects all these parameter and applies the effect.

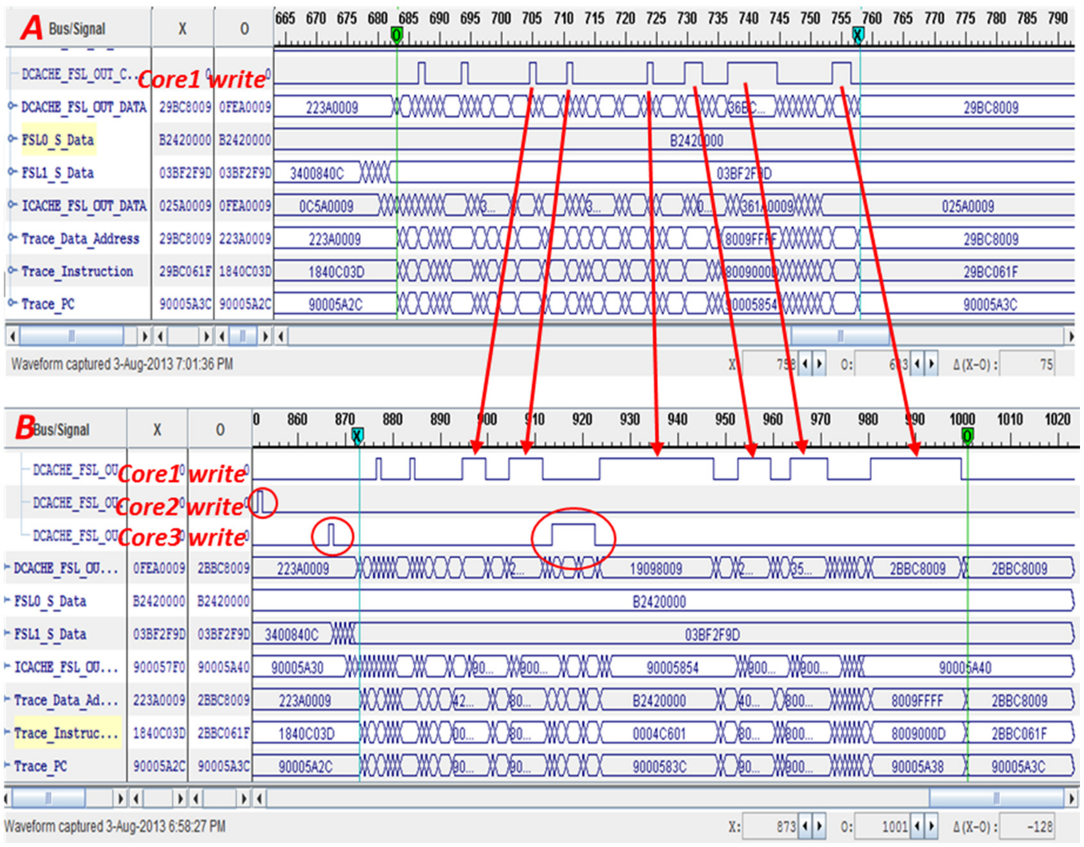


Figure 4.11 Effect of multiple writes in different port of MPMC with built-in cache

Figure 4.13 shows how the write time latency of a MicroBlaze core increases when other cores trying to write into MPMC. Figure 4.13 (A) presents memory transactions of a particular program with certain amount of writes on a single core design, while Figure 4.13 (B) presents the same program running on the first core of a multicore design and other cores have writes into the MPMC (the write transactions are circled in the figure). It should be noted, the cores are independent to each other and do not communicate to each other.

In multicore emulation with hybrid prototyping, only one core is simulated at a time. Hence, it is not possible to predict the exact behavior of the other cores during simulation. This effect causes the predicted execution time to be less than what is expected. In order to decrease this effect, we introduce a multiplication factor f_m, which models the multiple write effect. To determine f_m, we tested different multicore designs with all the cores running in parallel. We observed that the multiple write effect depends on the number, density, and distribution of writes over different cores. As much as number of writes, and number of cores increases the effect gets more severe. As a result, we executed a sample software code with different write distributions on multiple cores running in parallel.

Listing 4.1 Sample code for multiple port write test

```
Task1                      Task2:
for i=0 to 1000            for i=0 to n (n is variable)
nop;                       nop;
DDR write                  DDR write
nop;                       nop;
DDR write                  DDR write
end for                    nop;
                           DDR write
                           end for
```

Table 4.1 Multiple write factor for different number of cores

Number of cores	**2**	**3**	**4**
f_m	4	9	12

Listing 4.1 shows this sample code for a 2core design. Both task1 and task2 have a normal distribution of writes. In each experiment, we kept the write density of core 1 constant, and changed the write density of the other core, then observed how the core 1 execution time is changing. We test different write densities from the best case, which there is no write on the second core, to the worst case, which there is almost 100% write density. We examined different number of cores, and found an average f_m value for each core. Table 4.1 presents the values of f_m for different number of cores.

Table 4.2: Effect of multiple writes to MPMC (Numbers in clock cycles)

Number of consecutive writes to port 0	**0**	**1**	**2**	**3**	**>=4**
Number of cycles to write to MPMC port 0 in BIC	0	2	4	5	11
Number of cycles to write to MPMC port 0 in DRAC	0	$2*12*f_m$	$4*12*f_m$	$5*12*f_m$	$11*12*f_m$
Number of cycles to read from MPMC port 1 in BIC	29	42	53	65	79
Number of cycles to read from MPMC port 1 in DRAC	$29*12*f_m$	$42*12*f_m$	$53*12*f_m$	$65*12*f_m$	$79*12*f_m$

Previously, it was mentioned the DRAC model scales its delays to be a multiple of built-in cache delays. Besides hit and miss time latencies, DRAC also models the successive and multiple write delays. Table 4.2 presents the write and read access parameters of the built-n cache, and the modeled parameter values of DRAC. In the single core design, the instruction and the data cache are utilizing separate ports of MPMC. Since there is no write into MPMC in the instruction cache, the read access time of the instruction cache is only effected by data cache writes. In the multicore design, there are more than one data caches that write into MPMC. Hence, the effect of multiple writes will be more severe in higher number of cores.

CHAPTER 5

Experimental Result

We developed DRAC model for the MicroBlaze soft processor implemented on a Virtex5 FPGA. Since DRAC is designed as a cache model in single core design, and a cache emulator in multicore design, we tested DRAC in both single and hybrid multicore designs. We also created full-FPGA single and multicore designs using MicroBlaze built-in cache as the reference for our hybrid estimation timings. As explained in the introduction, the target cache is set to direct mapped, 4 word cache line size, and write through writing policy. DRAC design is coded in VHDL language and evaluated by Xilinx EDK software [29] on ML507 Evaluation board. VHDL code of DRAC design is presented in the Appendix. The system clock is operating at frequency of 125MHz for all different designs.

5.1 Standalone Accuracy

Prior to use DRAC in hybrid prototyping, we evaluated the standalone model in a single core design. In order to check the functionality and timing accuracy of the standalone instruction and data DRAC model, we ran JPEG Encoder, Quicksort, and Dhrystone benchmarks for different cache sizes in a single core design. The closest model to DRAC is pCache [20], that is a data cache emulator implemented on FPGA. We used the same benchmarks as pCache and observed the average estimation accuracy improved 8.96% in Dhrystone and 2.57% in JPEG Encoder benchmark.

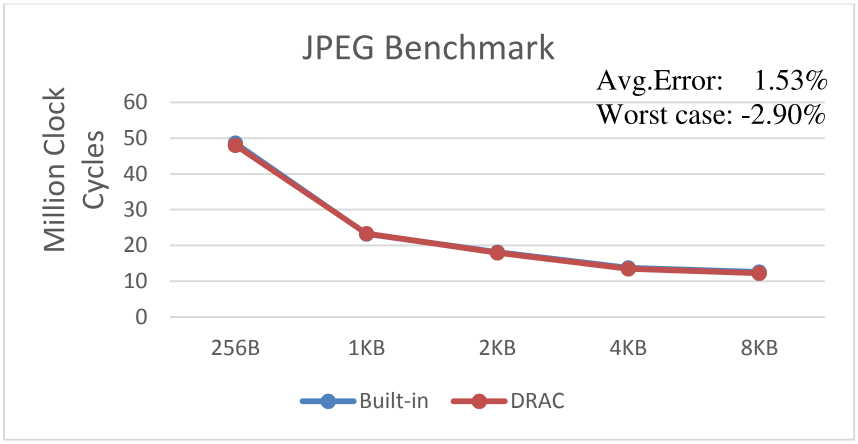


Figure 5.1 Execution time estimation of JPEG encoder benchmark in single core design

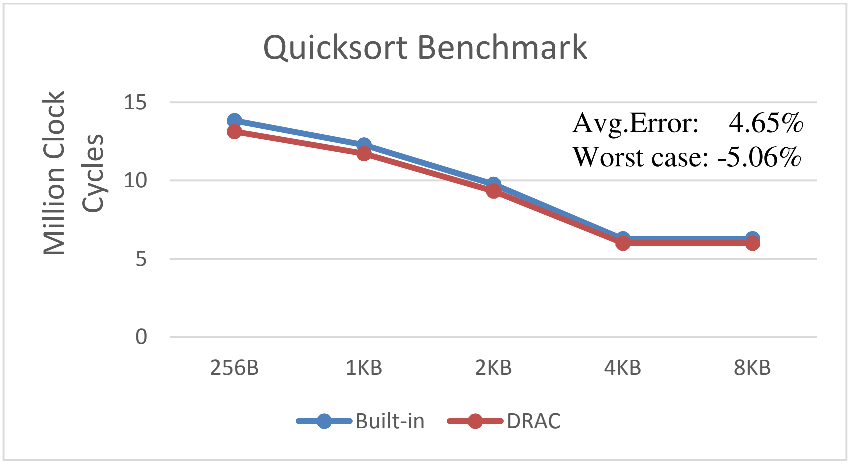


Figure 5.2 Execution time estimation of quicksort benchmark in single core design

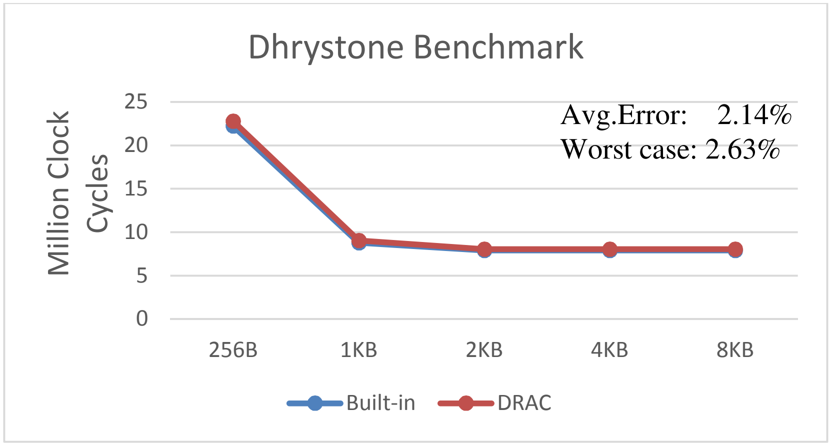


Figure 5.3 Execution time estimation of dhrystone benchmark in single core design

Figure 5.1, 5.2, and 5.3 demonstrates the execution time of different benchmarks running on the system with built-in cache, and with DRAC model. For all the experiments five different cache sizes have been chosen.

Table 5.1 Execution time estimation of different benchmarks in single core design

Benchmark	**Cache Size**	T_{BIC} (Million Cycles)	T_{DRAC} (Million Cycles)	**Error %**
JPEG	256B	48.63	48.05	-1.18
	1KB	23.19	23.31	0.49
	2KB	18.11	17.91	-1.10
	4KB	13.72	13.45	-1.98
	8KB	12.55	12.18	-2.90
Quicksort	256B	13.83	13.13	-5.06
	1KB	12.27	11.72	-4.48
	2KB	9.76	9.32	-4.59
	4KB	6.28	5.99	-4.61
	8KB	6.28	5.99	-4.61
Dhrystone	256B	22.25	22.79	2.41
	1KB	8.79	9.02	2.63
	2KB	7.90	8.05	1.90
	4KB	7.90	8.05	1.90
	8KB	7.90	8.05	1.90

The result values for different cases is shown in Table 5.1. We observed an average estimation time error of 2.78% and the worst-case estimation error is only 5.06%, thereby demonstrating the accuracy of DRAC as a standalone cache model.

5.2 Accuracy in Hybrid

In order to evaluate and verify the accuracy of DRAC emulator in the hybrid design, we created different multicore in the full FPGA design and the hybrid prototype, ranging from 1 to 4 cores. Each core is running in parallel with different tasks. These tasks can have communication to each other. In the full FPGA design, cores are connected to each other with FIFOs. FIFOs are FSL based channel that provide the communication link for the cores running in the parallel.

5.2.1 JPEG Encoder Benchmark

The benchmark used for testing the prototype is JPEG encoder. JPEG is a popular image compression technique that fits well into multi-processing system. The JPEG encoder divides the image into 8 by 8 pixel blocks. To compress the image, the encoder applies number of operations on these blocks. These operation includes ReadBMP, DCT, Quantization, ZigZag, and Huffman Encoding.

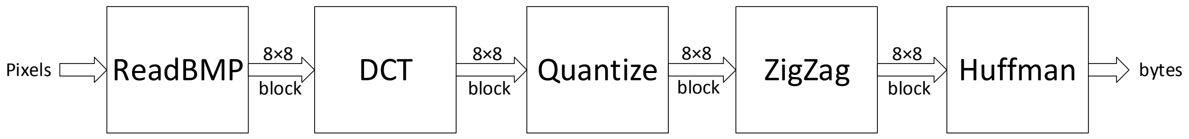


Figure 5.4 JPEG encoding process and its tasks

Figure 5.4 shows the JPEG encoding process and the order of the tasks. All the tasks are independent to each other and the communication is through 64bit FIFO channels. The operations are done one after the other. This makes the JPEG encoder a suitable program as benchmark for multicore processing system. Each task can be mapped to a core, and the FIFO channel can be realized by FSL links in full FPGA prototyping. In hybrid prototyping, the MEK platform emulates the cores and the channels, and runs the same JPEG encoder benchmark.

5.2.2 Timing Results

There are two timers implemented on each core in full FPGA prototype. The first timer calculates the actual busy-time of a core regardless of that core's waiting time on blocking reads or writes. The second measures the total execution time including program execution time and the processor's waiting times on FSL. In hybrid design, there are also two timers. One timer is used by the MEK, to simulate the busy-time and the total execution time of each core; the other timer calculates the total simulation time, including the swap time, the total execution time of the tasks, and the MEK software.

We created 15 different multicore designs for different JPEG encoder mappings. There are four possible of five JPEG task mappings for a 2 core design, six possible JPEG task mappings for a 3core design, and four possible JPEG task mappings in a 4core design. Since MPMC ports are limited to eight ports and each core consumes two ports, we cannot have more than 4 cores running in parallel in full FPGA Prototyping. We also obtained the estimation time of the same mapping in Hybrid Prototyping and calculated the error percentage.

The mapping values represent number of JPEG encoder tasks that have mapped to each core. For example, in the 2 core design, mapping 4-1 means that the first four tasks of JPEG encoder (ReadBMP, DCT, Quantization, ZigZag) have been mapped to the first core, and the last task (Huffman) of JPEG encoder to the second core.

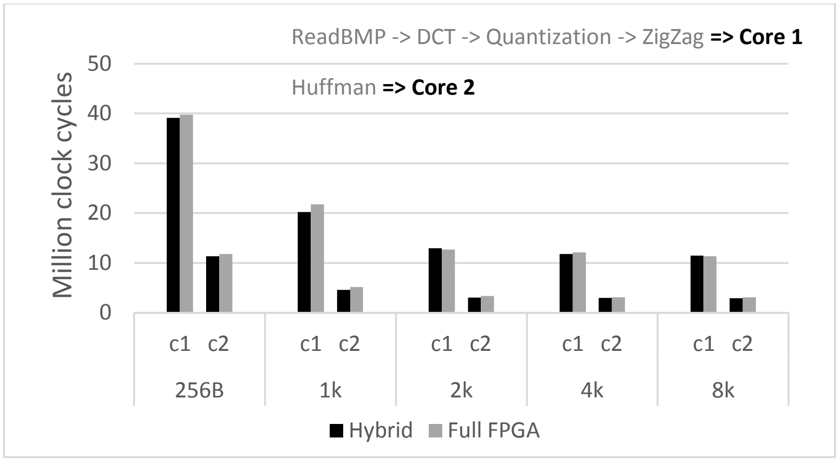


Figure 5.5 Busy-time estimation of a 2 core design with 4-1 JPEG mapping

Figure 5.5 presents the busy-time of a 2 core design for different cache sizes. Four tasks have been mapped to the first core, and one task to the second core. The relative accuracy with cache size increment is 100%. The average estimation error is 5.14% and worst case busy-time estimation error is for the second core in 1k design, which is 11.60%.

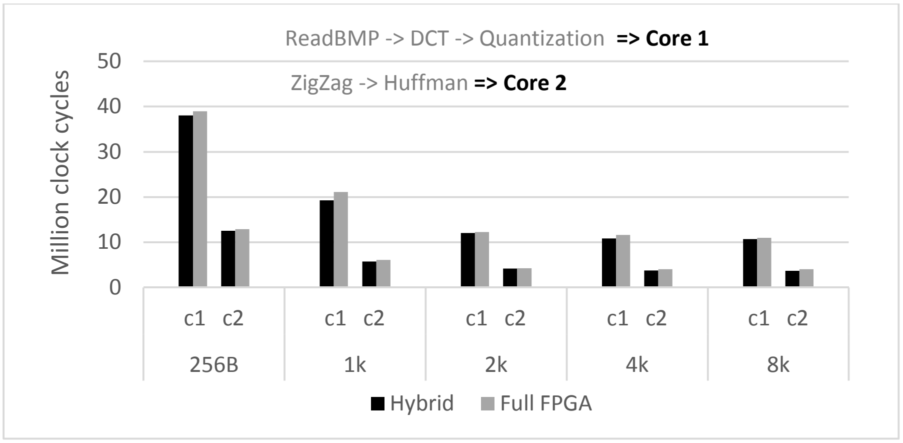


Figure 5.6 Busy-time estimation of a 2 core design with 3-2 JPEG mapping

Figure 5.6 demonstrates the busy-time of a 2 core design for different cache sizes. Three tasks have been mapped to the first core, and two tasks to the second core. The relative accuracy with cache size increment is 100%. The average estimation error is 4.86% and worst case busy-time estimation error is for the second core in 1k design, which is 8.61%.

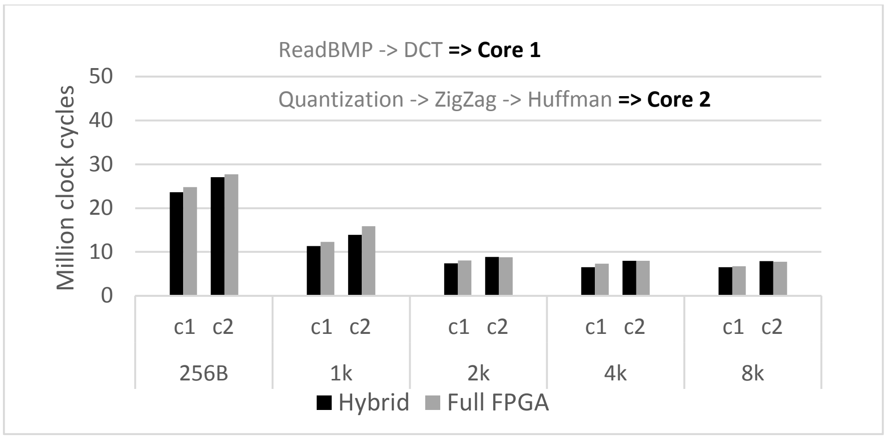


Figure 5.7 Busy-time estimation of a 2 core design with 2-3 JPEG mapping

Figure 5.7 shows the busy-time of a 2 core design for different cache sizes. Two tasks have been mapped to the first core, and three tasks to the second core. The relative accuracy with cache size increment is 100%. The average estimation error is 5.10% and worst case busy-time estimation error is for the second core in 1k cache size design, which is 12.08%.

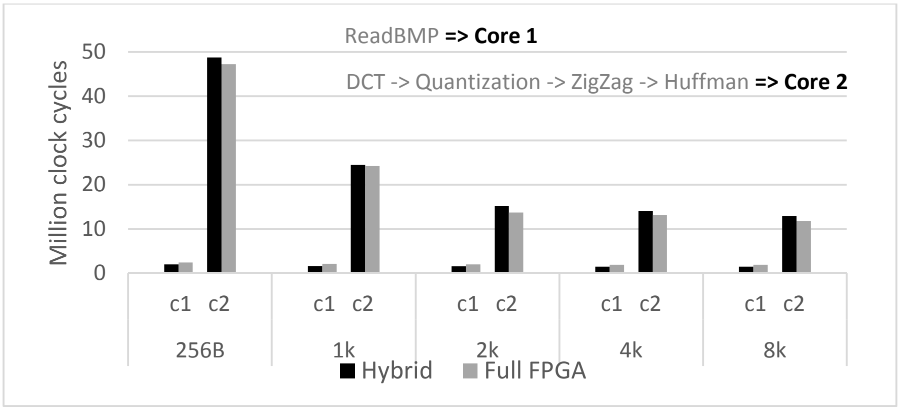


Figure 5.8 Busy-time estimation of a 2 core design with 1-4 JPEG mapping

Figure 5.8 illustrates the busy time of a 2 core design for different cache sizes. One task has been mapped to the first core, and four tasks to the second core. The relative accuracy with cache size increment is 100%. The average estimation error is 14.20% and worst case busy-time estimation error is for the first core in 2k cache size design, which is 23.77%.

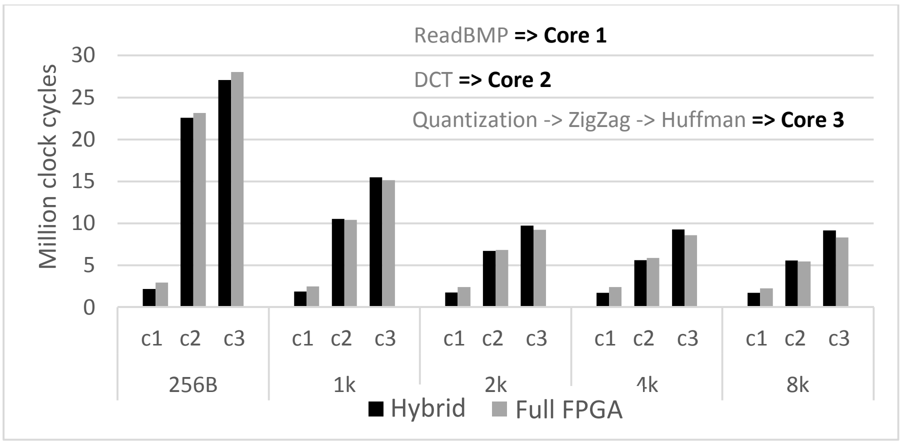


Figure 5.9 Busy-time estimation of a 3 core design with 1-1-3 JPEG mapping

Figure 5.9 shows the busy time of a 3 core design for different cache sizes. One task has been mapped to the first core, one task to the second core, and three tasks to the third core. The relative accuracy with cache size increment is 100%. The average estimation error is 11.27% and worst case estimation error is for the first core in 4k cache size design, which is 28.11%.

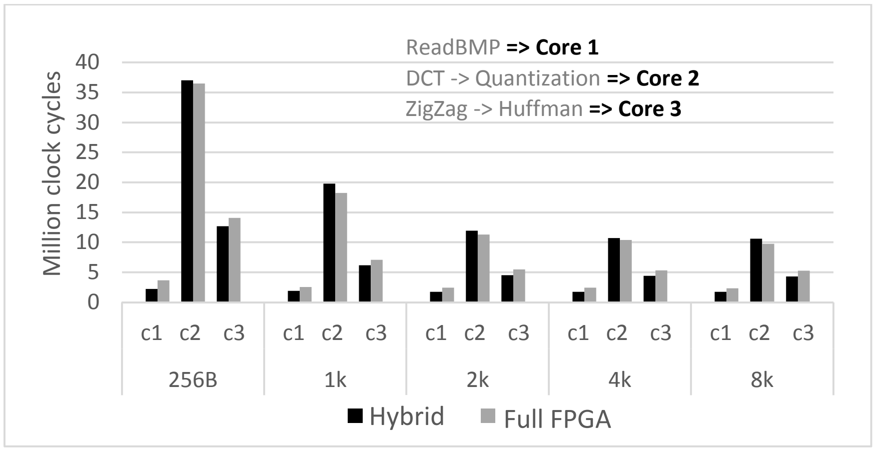


Figure 5.10 Busy-time estimation of a 3 core design with 1-2-2 JPEG mapping

Figure 5.10 presents the busy time of a 3 core design for different cache sizes. One task has been mapped to the first core, two tasks to the second core, and two tasks to the third core. The relative accuracy with cache size increment is 100%. The average estimation error is 16.67% and worst case estimation error is for the first core in 256B cache size design, which is 37.88%.

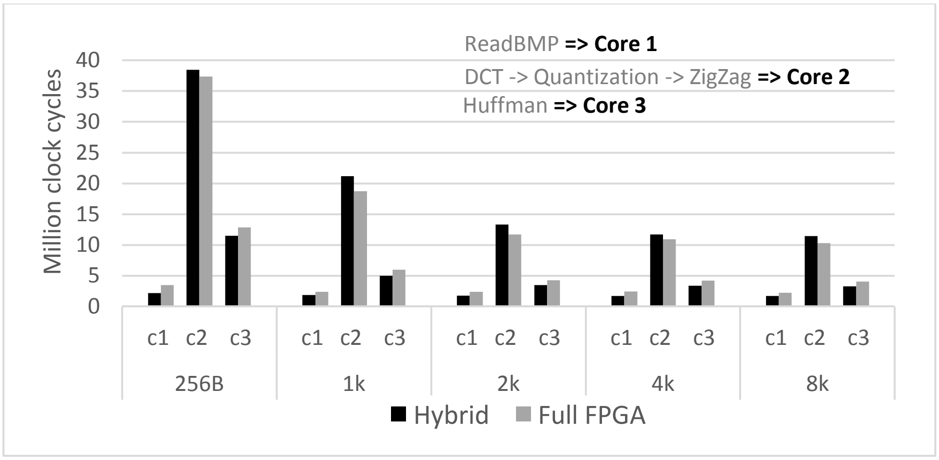


Figure 5.11 Busy-time estimation of a 3 core design with 1-3-1 JPEG mapping

Figure 5.11 presents the busy time of a 3 core design for different cache sizes. One task has been mapped to the first core, three tasks to the second core, and one task to the third core. The relative accuracy with cache size increment is 100%. The average estimation error is 17.69% and worst case estimation error is for the first core in 256B cache size design, which is 36.88%.

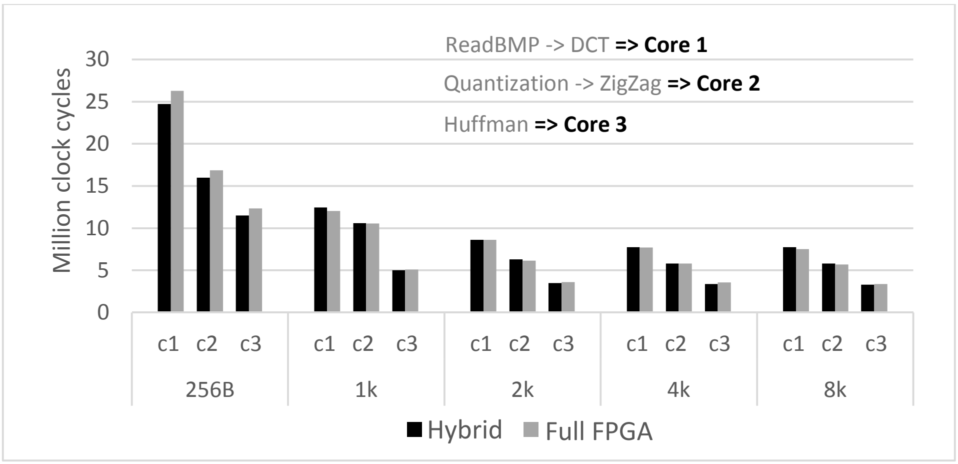


Figure 5.12 Busy-time estimation of a 3 core design with 2-2-1 JPEG mapping

Figure 5.12 shows the busy time of a 3 core design for different cache sizes. Two tasks have been mapped to the first core, two tasks to the second core, and one task to the third core. The relative accuracy with cache size increment is 100%. The average estimation error is 2.79% and worst case estimation error is for the third core in 256B cache size design, which is 6.71%.

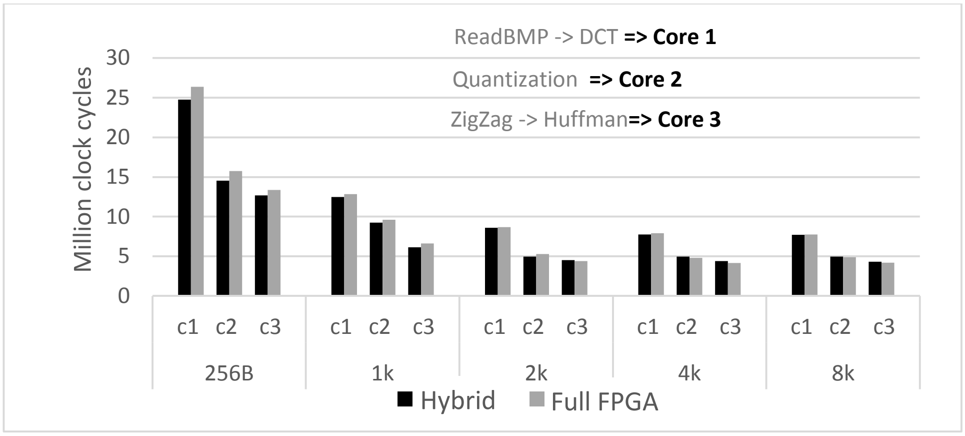


Figure 5.13 Busy-time estimation of a 3 core design with 2-1-2 JPEG mapping

Figure 5.13 demonstrates the busy time of a 3 core design for different cache sizes. Two tasks have been mapped to the 1st core, one task to the 2nd core, and two tasks to the 3rd core. The relative accuracy with cache size increment is 100%. The average estimation error is 3.89% and worst case estimation error is for the 2nd core in 256B cache size design, which is 7.49%.

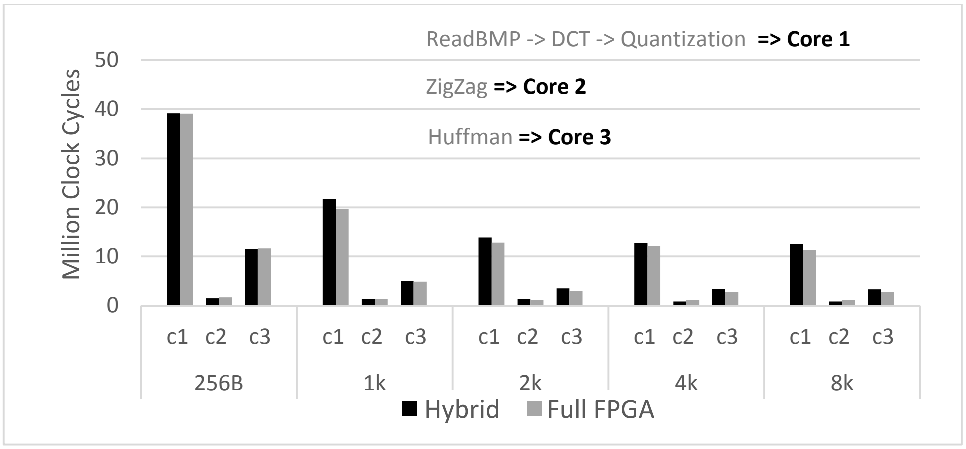


Figure 5.14 Busy-time estimation of a 3 core design with 3-1-1 JPEG mapping

Figure 5.14 illustrates the busy time of a 3 core design for different cache sizes. Three tasks have been mapped to the first core, one task to the second core, and one task to the third core. The relative accuracy with cache size increment is 100%. The average estimation error is 12.48% and worst case estimation error is for the 2nd core in 4KB cache size, which is 25.45%.

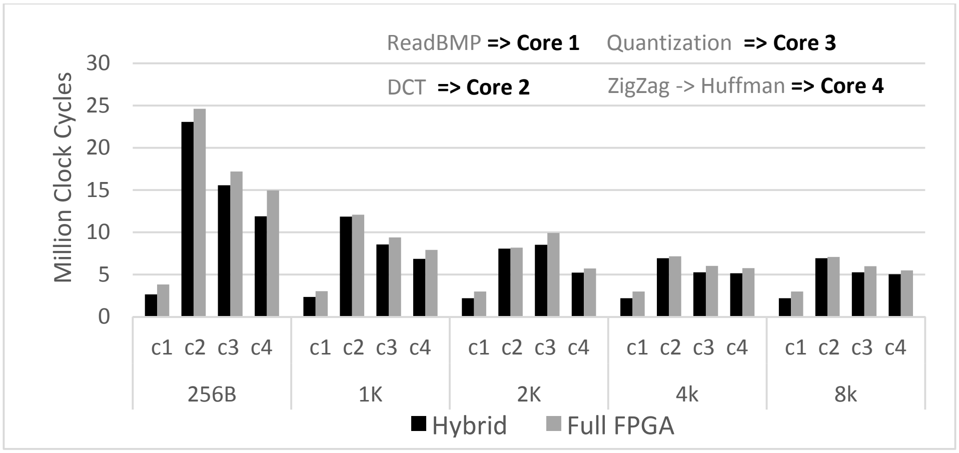


Figure 5.15 Busy-time estimation of a 4 core design with 1-1-1-2 JPEG mapping

Figure 5.15 shows the busy time of a 4 core design for different cache sizes. One task has been mapped to the 1st core, one task to the 2nd core, one to the 3rd core, and two tasks to the 4th core. The relative accuracy with cache size increment is 100%. The average estimation error is 13.34% and worst case estimation error is for the 1st core in 256B cache size, which is 31.09%.

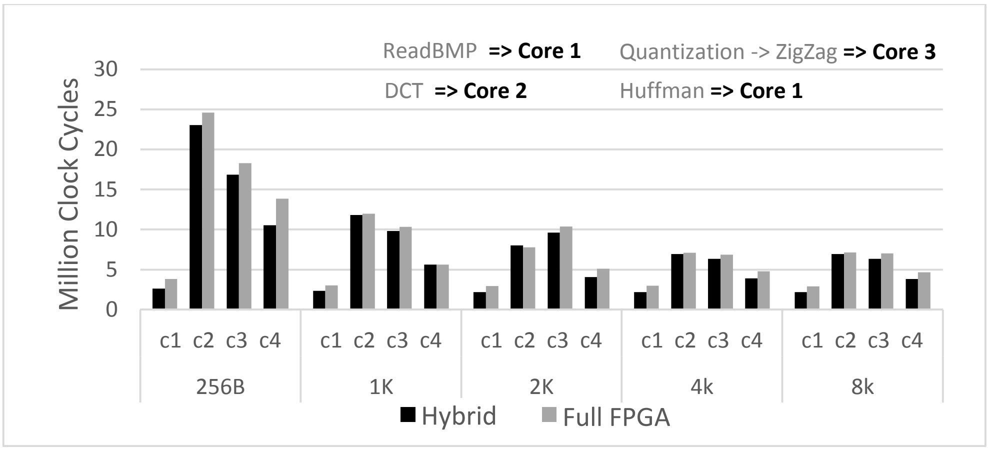


Figure 5.16 Busy-time estimation of a 4 core design with 1-1-2-1 JPEG mapping

Figure 5.16 presents the busy time of a 4 core design for different cache sizes. One task has been mapped to the 1st core, one task to the 2nd core, two tasks to the 3rd core, and one task to the 4th core. The relative accuracy with cache size increment is 100%. The average estimation error is 13.28% and worst case error is for the 1st core in 1KB cache size, which is 31.05%.

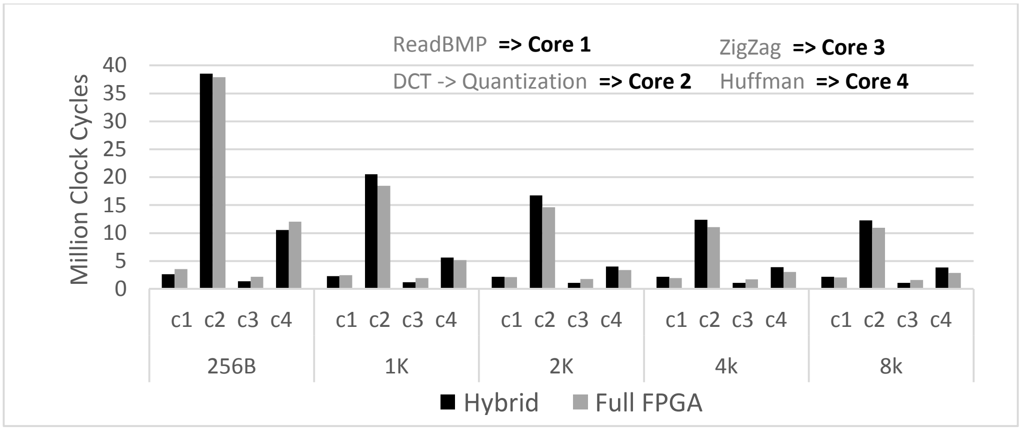


Figure 5.17 Busy-time estimation of a 4 core design with 1-2-1-1 JPEG mapping

Figure 5.17 illustrates the busy time of a 4 core design for different cache sizes. One task has been mapped to the 1st core, two tasks to the 2nd core, one task to the 3rd core, and one task to the 4th core. The relative accuracy with cache size increment is 100%. The average estimation error is 19.40% and worst case error is for the 3rd core in 1KB cache size, which is 38.18%.

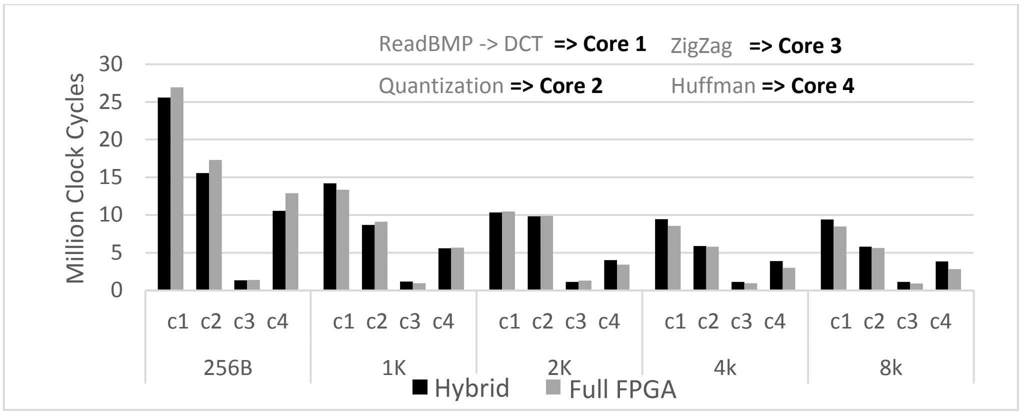


Figure 5.18 Busy-time estimation of a 4 core design with 2-1-1-1 JPEG mapping

Figure 5.18 demonstrates the busy-time of a 4 core design for different cache sizes. Two tasks have been mapped to the 1st core, one task to the 2nd core, one task to the 3rd core, and one task to the 4th core. The relative accuracy with cache size increment is 100%. The average estimation error is 11.71% and worst case error is for the 4th core in 8KB cache size, which is 34.48%.

Table 5.2 Busy-time error for different JPEG mappings

Number of cores	Mapping	Average Error	Worst-case		
			Error	Core	Cache Size
2core	4-1	5.14%	11.60%	2nd	1kB
	3-2	4.86%	8.61%	1st	1kB
	2-3	5.10%	12.08%	2nd	1kB
	1-4	14.20%	23.77%	1st	2kB
3core	1-1-3	11.27%	28.22%	1st	4kB
	1-2-2	16.67%	39.88%	1st	256B
	1-3-1	17.69%	36.88%	1st	256B
	2-2-1	2.79%	6.71%	3rd	256B
	2-1-2	3.89%	7.49%	2nd	256B
	3-1-1	12.48%	25.45%	2nd	4kB
4core	1-1-1-2	13.34%	31.09%	1st	256B
	1-1-2-1	13.28%	31.05%	1st	1kB
	1-2-1-1	19.40%	38.18%	3rd	1kB
	2-1-1-1	11.71%	34.48%	4th	8kB

Table 5.2 summarizes the busy-time estimation error for different number of cores, JPEG task mappings, and five different cache sizes. The average busy-time estimation error is 10.84% for all the experiment and the worst case estimation time error is 39.88% which is for the first core of the 3 core design with mapping of 1-2-2 and the cache size of 256B.

The busy-time error is worse when the load of a core is significantly low in compare to the others cores. This means the cores with the highest loads affect severely on low load cores. The source of this error is the multiple write effect discussed in section 4.3. Before applying our compensation algorithm for multiple write effects of DDR2, our worst case error was 66% in 3th core of 4 core emulation, with 1-2-1-1 mapping, and 1kB of data and instruction cache. Through the proposed compensation algorithm, we decreased the error to 38.18% in that case.

Total execution time of the tasks are an important factor to compere different designs. As explained earlier, the total execution time includes the FSL waiting due to FIFO's blocking read or write effect as well as program's execution time.

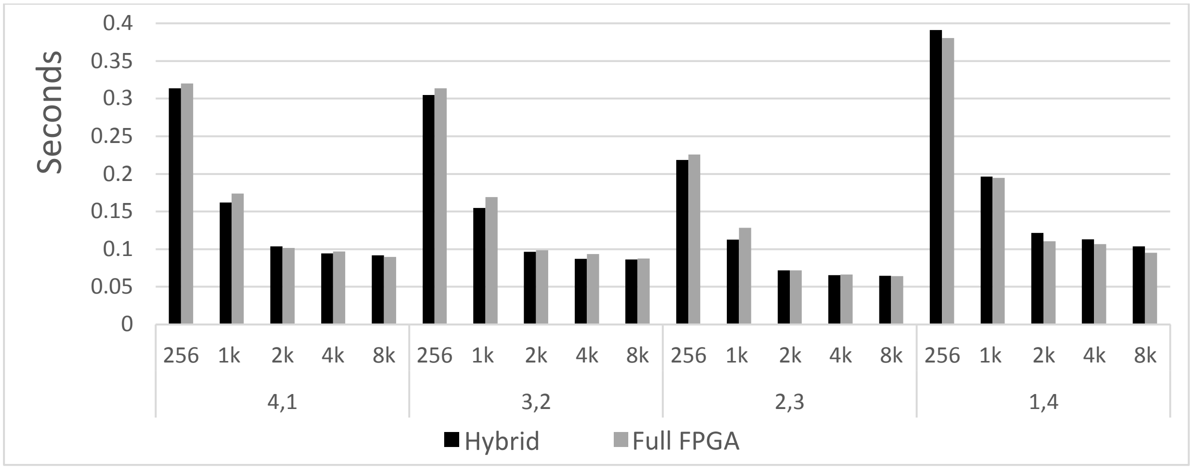


Figure 5.19 Total execution time estimation of a 2core design for different JPEG mapping

Figure 5.19 presents the total execution time of 2core design for all possible JPEG encoder mapping and cache sizes. The average total execution time error for 2core design is 4.20% and the worst case estimation error is 12.3% which belongs to 2-3 JPEG mapping and 4K cache size design.

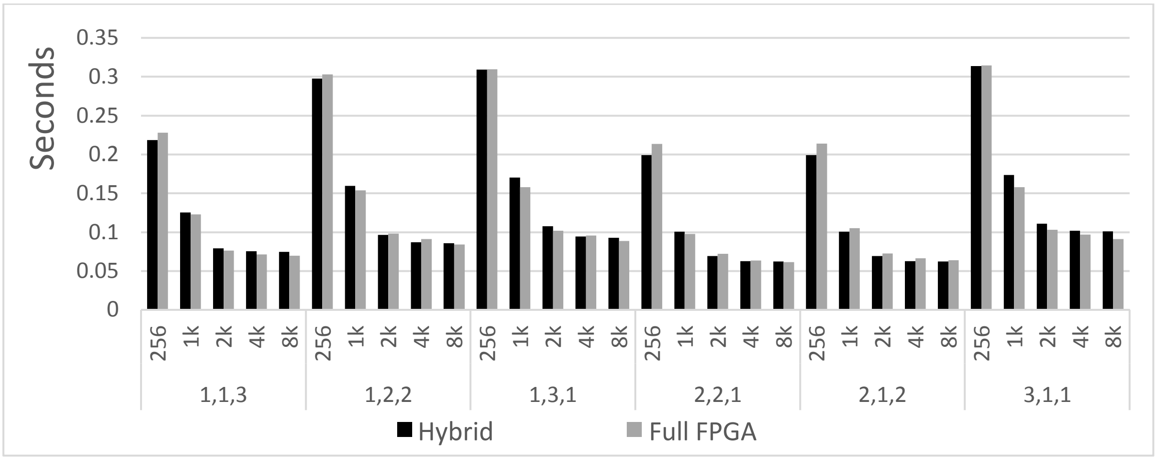


Figure 5.20 Total execution time estimation of a 3core design for different JPEG mapping

Figure 5.20 shows the total execution time of 3core design for all possible JPEG encoder mapping and cache sizes. The average total execution time error for 3core design is 4.29% and the worst case estimation error is 10.8% which belongs to 3-1-1 JPEG mapping and 8K cache size design.

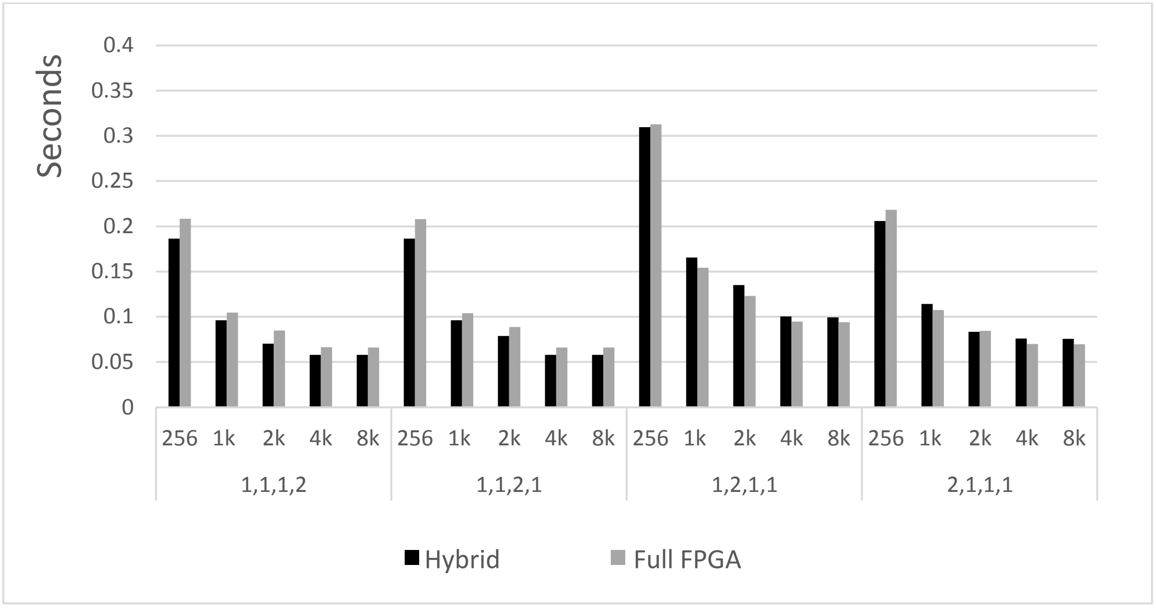


Figure 5.21 Total execution time estimation of a 4core design for different JPEG mapping

Figure 5.21 illustrates the total execution time of 4core design for all possible JPEG encoder mapping and cache sizes. The average total execution time error for 4core design is 8.82% and the worst case estimation error is 12.98% which belongs to 1-1-1-2 JPEG mapping and 4K cache size design.

Table 5.4 shows the average and worst case total execution estimation error for different JPEG mapping and cache sizes. The total average execution time error is 5.56% and the worst case error is 12.98 in 4core design, 1-1-1-2 JPEG mapping, and the cache size of 4KB.

Table 5.3 Total execution time error for different JPEG mappings

Number of cores	Mapping	Average Error	Worst-case	
			Error	Cache Size
2core	4-1	3.17%	6.76%	1 KB
	3-2	4.24%	8.56%	1 KB
	2-3	3.50%	12.3%	4 KB
	1-4	5.88%	10.2%	1 KB
3core	1-1-3	4.50%	7.24%	8 KB
	1-2-2	2.73%	4.34%	4 KB
	1-3-1	3.76%	7.81%	1 KB
	2-2-1	3.24%	6.73%	256 B
	2-1-2	4.76%	6.92%	256 B
	3-1-1	6.76%	10.8%	8 KB
4core	1-1-1-2	12.24%	12.98%	4 KB
	1-1-2-1	10.80%	12.55%	4 KB
	1-2-1-1	5.96%	9.78%	2 KB
	2-1-1-1	6.27%	9.09%	8 KB

5.3 Simulation Speed

As mentioned earlier, there is a timer for calculating the total simulation time. It starts at the first of the simulation, and stops at the end of the program. The total simulation time of the MEK can be seen in Figure 5.22.

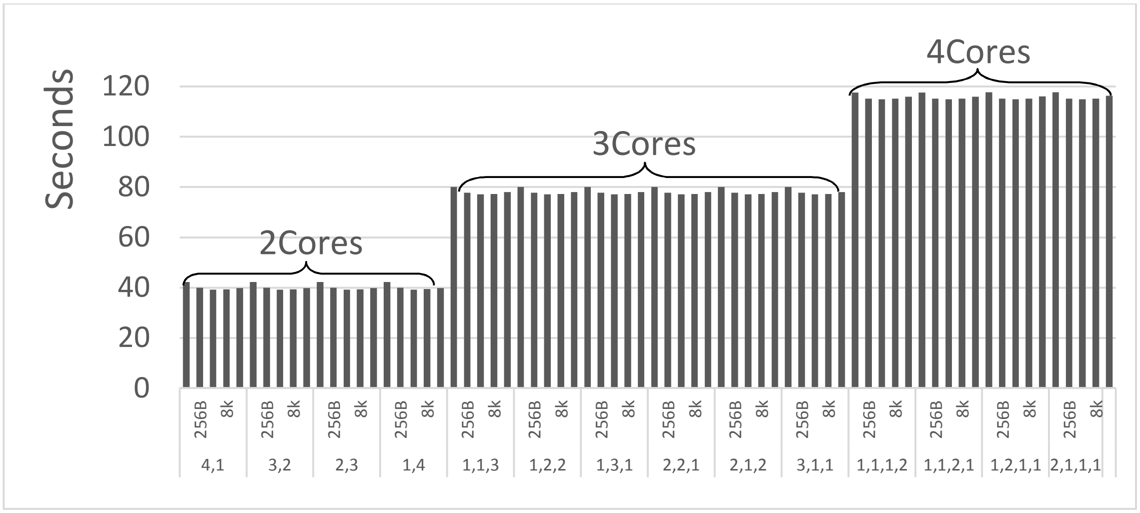


Figure 5.22 Total Simulation time in Seconds for different number of cores in Hybrid design

Table 5.4 Swap time consumption for different sizes

Cache Size	**Save (Cycles)**	**Load (Cycles)**	**Total Swap Time (Cycles)**
256B	2467	3831	6298
1k	8499	13725	22224
2k	16477	26957	43434
4k	32513	53343	85856
8k	64521	106123	170644

The values are obtained for all task mappings and all 5 different cache sizes ranging from 256B to 8KB. Because the MEK is running on off-chip DDR2 SDRAM memory, and the swap is also running during the simulation, the timing is quite high in comparison to MEK running on BRAM. The simulation time increases by increasing the number of cores, since number of cache swaps increase. During simulation, both instruction and data cache is disabled, and the cache is enabled only when a task is running. Because of this, cache size increment effect is not significant in total simulation time. Even in some cases, the cache size increment results in higher simulation timing. The reason is that if the cache size increases, the swap time increases as well. Table 5.4 reports the time consumption for a load/save from/to DDR2 to/from the cache, and total swap (Load + Save).

5.4 DRAC Resource Usage

Each design consumes a certain amount IO, and occupies a portion of FPGA area during implementation. For different cache sizes, we have obtained resource usage.

Table 5.5 Resources usage of Built-in Cache (BIC) and DRAC for different cache sizes

Design	**Resources Usage Percentage**				
	LUT	**Reg.**	**BRAM**	**Slice**	**Bonded IO**
I&D 256B Built-in	12%	14%	14%	29%	18%
I&D 1KB Built-in	12%	14%	14%	29%	18%
I&D 2KB Built-in	12%	14%	16%	29%	18%
I&D 4KB Built-in	12%	14%	18%	29%	18%
I&D 8KB Built-in	12%	14%	21%	29%	18%
I&D Variable Size DRAC	22%	33%	23%	53%	18%

Table 5.5 presents resources usage of single core design with different sizes of the built-in cache and DRAC cache emulator. As can be observed, as much the size of the cache increases the amount of BRAM used on the FPGA increases as well; however, other logic resources remains the same. Although DRAC consumes more resources on FPAG than built-in cache, it promotes run-time configurability. The design with built-in cache must be re-synthesized once for any configuration change like cache size, but the emulation system with DRAC can simulate any configuration with only one time synthesis.

5.5 Power and Energy Analysis

Beside the speed of the system, power consumption is the other main factor for the designer to choose the best design in multicore processing. The main components that consume the most of the power are the processor, built-in cache, and off-chip main memory.

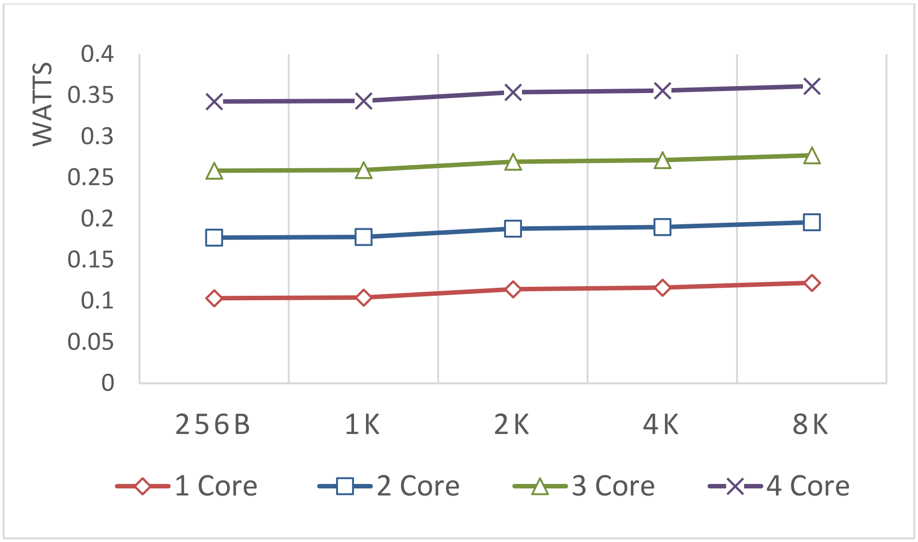


Figure 5.23 Power consumption for different cache sizes in Watts

Figure 5.23 demonstrates the total power consumption of the system for different number of cores and cache sizes. As can be seen, the cache size increment results in more BRAM utilization and more power consumption. On the other hand, adding more cores to the system and using more ports of MPMC increases the power consumption as well.

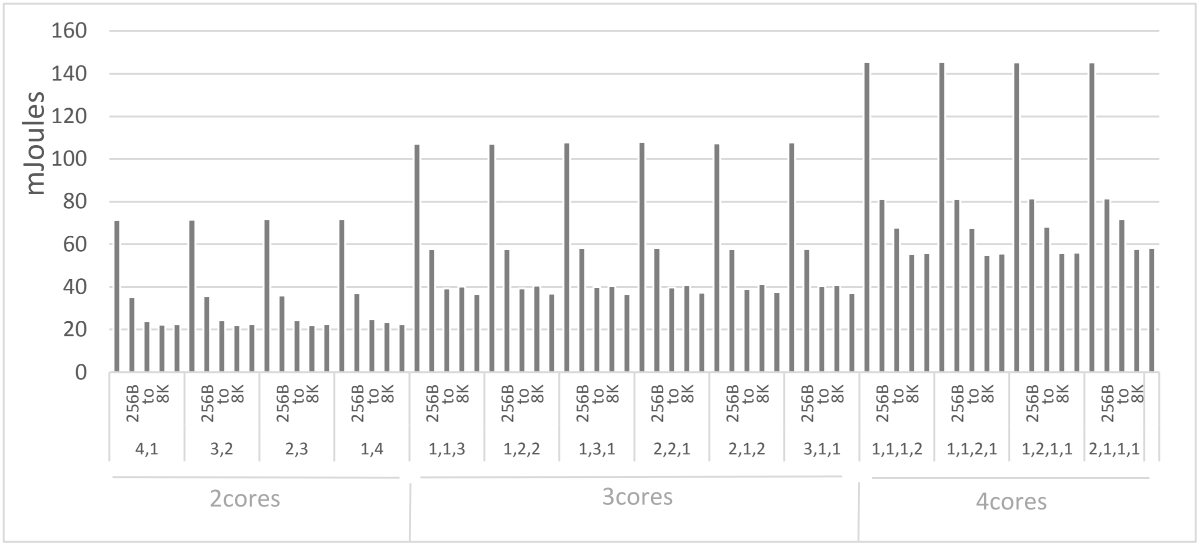


Figure 5.24 Energy consumption for different cache sizes and JPEG mappings in mJoules

In multicore systems, power consumption is different core by core, depending on the task running on each core. Energy is the best way to measure the system performance in terms of power and time. The busy-time of a task is the time for a core to execute a task without considering blocking data transfer among different cores. The processor is on idle during blocking reads or writes, hence it consumes negligible amount of energy. Because of this fact, we multiplied the total power consumption of each core to the total busy-time of all cores and obtained the energy consumption for different task mappings. Figure 5.24 demonstrates the energy consumption for all possible JPEG encoder designs and cache sizes.

5.6 Design Exploration

Two of the most important factors that define system efficiency, are the speed and energy consumption of the system.

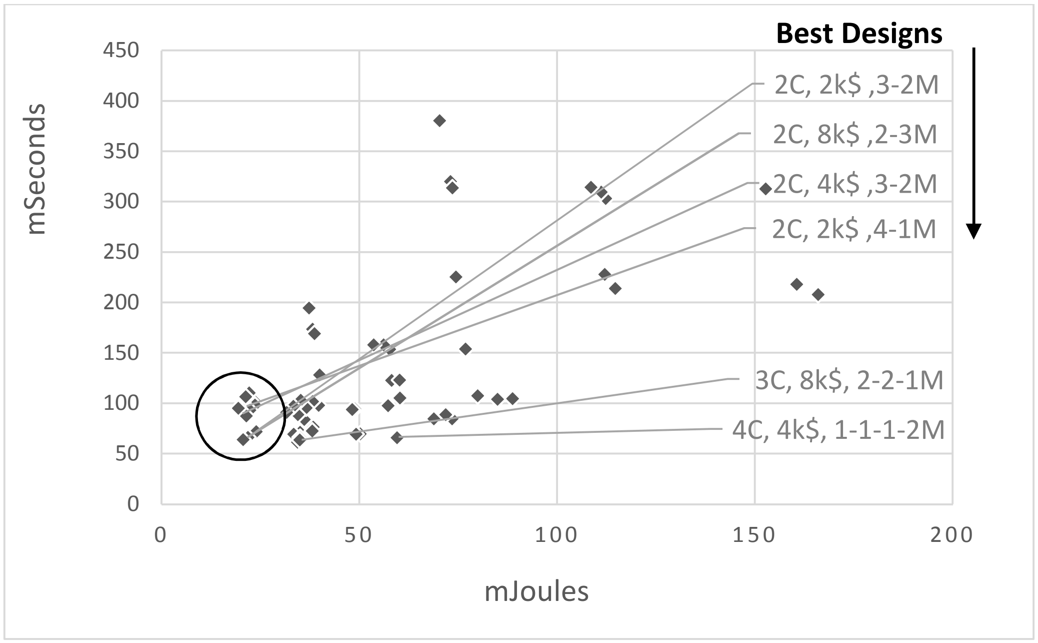


Figure 5.25 Design exploration of JPEG for different task mapping and cache sizes in terms of energy & speed in Full FPGA prototyping

Figure 5.25 plots all the full FPGA multicore designs from 2 to 4 cores with all possible JPEG Encoder mappings, and five different cache sizes execution time versus energy consumption. Each point is a design with certain mapping and the cache size. As it is circled on the figure, the best designs are the one that consume less energy and execute the program in the shortest time. For example, the best design in JPEG Encoder is a 2 core design with 2k cache size and the mapping of 2 tasks in the first core and 3 tasks in the second core.

The MEK provides a simple environment for the designer to choose the best design among the others, without having the full FPGA multicore prototype. The consistency of the results, 100% relative accuracy among different cache sizes and different task mapping, make the MEK a powerful tool to compare different designs.

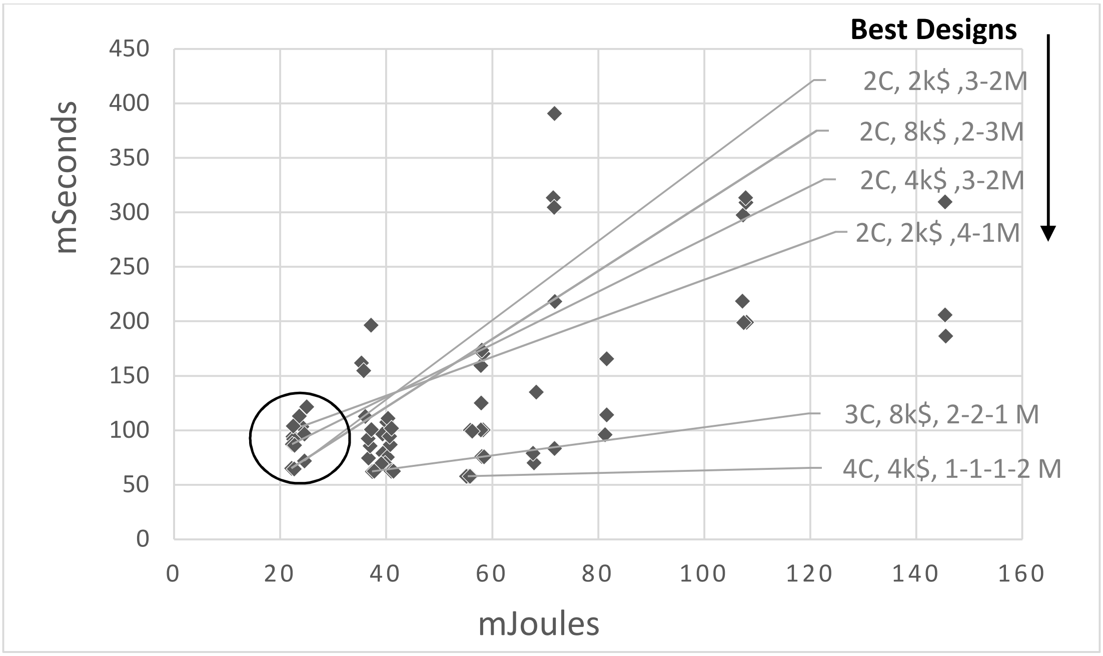


Figure 5.26 Design exploration of JPEG for different task mapping and cache sizes in terms of energy & speed in Hybrid prototyping

Figure 5.26 presents energy versus execution time for all JPEG Encoder possible mapping and the cache sizes, predicted by the MEK. The correlation of the MEK results and the full FPGA results is clear. In both Figure 5.25 and 5.26, the best design is the 2core design with 2k instruction and data cache with 3-2 JPEG mapping. This confirms the accuracy and reliabality of the MEK with cache.

CHAPTER 6

Conclusion and Future work

In this thesis, we presented DRAC instruction and data cache model that can be used in hybrid prototyping. Standalone accuracy, run-time configurability, and multiple cache context support of DRAC, make it an ideal cache emulator in multicore emulation systems. DRAC model is capable of emulating instruction and data caches of many virtual cores, as well as modeling cache in a single core design. Designing DRAC as an active cache emulator increases the accuracy and detail of the model. Additionally, parametric design of DRAC allows the model to be integrated with different processors.

Adopting hybrid prototyping idea and utilizing DRAC model, embedded designers are able to analyze, verify, and optimize their multicore design with cache design without the need for full system prototyping. Such full system prototyping can be complex to be designed and time consuming for the designer to explore different design options. Therefore, hybrid prototypes increase the productivity for both software embedded designer and hardware multicore designer.

In the future, we will extend DRAC model to support the following,

- Write back writing policy – This writing policy increases the performance of data cache by decreasing the cache refereeing to main memory. In order to implement the write back policy, the cache controller should be extended.
- Set Associativity – Set associativity determines the replacement policy of a particular entry of the cache. Now, DRAC is one-way associative or direct mapped

cache, but in the future it will support different associativity. Although the model becomes more complex, the state of the art in computer architecture is multi-way associativity.

- Different cache levels – Designing the cache in multiple levels is another way to optimize system performance; however, managing different cache levels is a great challenge in multicore processing. In future, DRAC model will be extended to support L2 and L3 caches.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine,* vol. 38, p. 114 ff, April 1965.

[2] R. Hiremane, "From Moore's Law to Intel Innovation—Prediction to," *Technology@Intel,* pp. 1-9, April 2005.

[3] D. Geer, "Chip makers turn to multicore processors," *IEEE Computer,* vol. 38, p. 11–13, May 2005.

[4] Y. Hwang, S. Abdi and D. Gajski, "Cycle approximate retargettable performance estimation at the transaction level," in *DATE*, Munich, Germany, Mar 2008.

[5] Z. Wang and A. Herkersdorf, "An Efficient Approach for System-Level Timing Simulation of Compiler-Optimized Embedded Software," in *Proceedings of the 46th Annual Design Automation Conference (DAC'09)*, San Francisco, California, July 2009.

[6] E. Saboori and S. Abdi, "Hybrid Prototyping of Multicore Embedded Systems," in *DATE*, Grenoble, France, 2013.

[7] P. Denning and S. Schwartz, "Properties of the Working-Set Model," *Communications of the ACM,* vol. 15, no. 3, pp. 191-198, March 1972.

[8] J. Edler and M. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," 2012. [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/.

[9] B. Atanasovski, S. Ristov, M. Gusev and N. Anchev, "MMCacheSim: A Highly Configurable Matrix Multiplication Cache Simulator," in *ICT Innovations*, 2012.

[10] T. Kempf, k. Karuri, G. Ascheid, R. Leupers and H. Meyr, "A SW performance estimation framework for early system-," in *DATE*, 2006.

[11] Z. Wang and A. Herkersdorf, "An Efficient Approach for System-Level Timing Simulation of Compiler-Optimized Embedded Software," in *Proceedings of the 46th Annual Design Automation Conference (DAC'09)*, San Francisco, California, July 2009.

[12] E. Cheung, H. Hsieh and F. Balarin, "Memory subsystem simulation in software tlm/t models," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference , ser. ASP-DAC '09*, 2009.

[13] S. Stattelmann, G. Gebhard, C. Cullmann, O. Bringmann and W. Rosenstiel, "Hybrid source-level simulation of data caches using abstract cache models," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012.

[14] Z. Wang and A. Herkersdorf, "An efficient approach for system-level timing simulation of compiler-optimized embedded software," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, San Francisco, CA, July 2009.

[15] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation techniques for storage hierarchies," in *IBM Systems Journal*, 1970.

[16] M. D. Hill and A. J. Smith, "Evaluating associativity in CPU caches," in *IEEE Trans. Comput*, 1989.

[17] R. Sugumar and S. Abraham, "Efficient simulation of multiple cache configurations using binomial trees," in *Tech. Report*, 1991.

[18] W. Zang and A. Gordon-Ross, "T-SPaCS – a Two-level Single-pass Cache Simulation Methodology," in *Proc. 16th Asia and South Pacific Design Automation Conference* , Jan. 2011.

[19] A. Janapsatya, A. Ignjatovic and S. Parameswaran, "Finding optimal L1 cache configuration for embedded systems," in *Design Automation, 2006. Asia and South Pacific Conference on*, 10.1109/ASPDAC.2006.1594753, Jan 2006.

[20] H. Yoon , G. Park, K. Lee , T. Han, S. Kim and S. Yang, "Reconfigurable Address Collector and Flying Cache Simulator," in *High Performance Computing on the Information Superhighway, 1997. HPC Asia '97*, Seoul, 1997.

[21] A. Nanda, K. Mak, K. Sugavanam, R. K. Sahoo, V. Soundararajan and T. B. Smith, "MemorIES: A programmable, real-time hardware emulation tool for multiprocessor server design," in *Proc. Conf. Arch. Support Program. Lang. Operat. Syst.*, 2000.

[22] P. Ravishankar and A. Samar, "pCache: An Observable L1 Data Cache Model for FPGA Prototyping of Embedded Systems," in *Digital System Design (DSD), 2013 Euromicro Conference on*, 2013.

[23] N. Chalainanont, E. Nurvitadhi, R. Morrison, L. Su, K. Chow, S. L. Lu and K. Lai, "Real-time L3 cache simulations using the programmable hardware-assisted cache emulator (PHA$E)," in *the 6th Ann. Workshop Workload Characterization*, Madrid, Spain, 2003.

[24] L. A. Barroso, S. Iman, J. Jeong, K. Oner, K. Ramamurthy and M. Dubois, "RPM: A rapid prototyping engine for multiprocessor systems," in *IEEE Comput. Mag.*, Feb. 1995.

[25] J. Wawrzynek and et al., "RAMP: Research Accelerator for Multiple Processors," in *Micro, IEEE*, 2007.

[26] E. Nurvitadhi, J. Hong and S. Lu, "Active Cache Emulator," vol. 16, no. 3, pp. 229 - 240, March 2008.

[27] X. Inc., "MicroBlaze Processor Reference Guide," [Online]. Available: http://www.xilinx.com.

[28] X. Inc., "ChipScope Pro and the Serial I/O Toolkit," [Online]. Available: http://www.xilinx.com/tools/cspro.htm.

[29] X. Inc., "Platform Studio and the Embedded Development Kit (EDK)," [Online]. Available: http://www.xilinx.com/tools/platform.htm.

Appendix

VHDL CODE of DRAC

```
use ieee.std_logic_1164.all;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
use proc_common_v3_00_a.ipif_pkg.all;

library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity DRAC is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
          C_BRIDGE_BASEADDR      : std_logic_vector(0 TO 31) := X"FFFFFFFF";
     C_BRIDGE_HIGHADDR      : std_logic_vector(0 TO 31) := X"00000000";

                    C_SPLB_AWIDTH              : integer          := 32;
                    C_SPLB_DWIDTH              : integer          := 32;
                    C_MPLB_AWIDTH              : integer          := 32;
                    C_MPLB_DWIDTH              : integer          := 32;
                    C_SPLB_NUM_MASTERS         : integer          := 84;
                    C_MPLB_NATIVE_DWIDTH       : integer          := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
        port (
                     CR_out                     : out std_logic_vector (0 to 15);
                     CW_out                     : out std_logic_vector (0 to 3);
                     write_out                  : out std_logic;
                     miss_in                    : in std_logic;
                     miss_out                   : out std_logic;
                     RData_valid_in             : in std_logic;
                     RData_valid_out            : out std_logic;
                     Trace_PC                   : in std_logic_vector (0 to 31);
                     Swap_in                    : in std_logic;
                     Swap_out                   : out std_logic;
                     MPLB_Clk                   : in std_logic;
                     SPLB_Rst                   : in std_logic;
                    -- PLBv46 Bus Slave signals
                    PLB_ABus                    : in std_logic_vector(0 to C_SPLB_AWIDTH-1);
                    PLB_PAValid                 : in std_logic;
                    PLB_RNW                     : in std_logic;
                    PLB_wrDBus                  : in std_logic_vector(0 to C_SPLB_DWIDTH-1);
                   PLB_BE                       : in std_logic_vector(0 to 3);

                    -- Slave Response Signals
                    Sl_addrAck                  : out std_logic;
                    Sl_SSize                    : out std_logic_vector(0 to 1);
                    Sl_wait                     : out std_logic;
                    Sl_rearbitrate              : out std_logic;
                    Sl_wrDAck                   : out std_logic;
                    Sl_wrComp                   : out std_logic;
                    Sl_wrBTerm                  : out std_logic;
```

```
                      Sl_rdDBus                    : out std_logic_vector(0 to C_SPLB_DWIDTH-1);
                      Sl_rdWdAddr                  : out std_logic_vector(0 to 3);
                      Sl_rdDAck                    : out std_logic;
                      Sl_rdComp                    : out std_logic;
                      Sl_rdBTerm                   : out std_logic;
                      Sl_MBusy                     : out std_logic_vector (0 to C_SPLB_NUM_MASTERS-1);
                      Sl_MWrErr                    : out std_logic_vector (0 to C_SPLB_NUM_MASTERS-1);
                      Sl_MRdErr                    : out std_logic_vector (0 to C_SPLB_NUM_MASTERS-1);
                      Sl_MIRQ                      : out std_logic_vector (0 to C_SPLB_NUM_MASTERS-1);
                     IP2Bus_MstRd_Req              : out std_logic;
                     IP2Bus_MstWr_Req              : out std_logic;
                     IP2Bus_Mst_Addr               : out std_logic_vector(0 to C_MPLB_AWIDTH-1);
                     IP2Bus_Mst_BE                 : out std_logic_vector(0 to (C_MPLB_NATIVE_DWIDTH/8) -1);
                     IP2Bus_Mst_Lock               : out std_logic;
                     IP2Bus_Mst_Reset              : out std_logic;

                     -- IP Request Status Reply
                     Bus2IP_Mst_CmdAck             : In std_logic;
                     Bus2IP_Mst_Cmplt              : In std_logic;
                     Bus2IP_Mst_Error              : In std_logic;
                     Bus2IP_Mst_Rearbitrate        : In std_logic;
                     Bus2IP_Mst_Cmd_Timeout        : In std_logic;

                    -- IPIC Read data
                     Bus2IP_MstRd_d               : In std_logic_vector(0 to C_MPLB_NATIVE_DWIDTH-1);
                     Bus2IP_MstRd_src_rdy_n       : In std_logic;

                    -- IPIC Write data
                     IP2Bus_MstWr_d               : out std_logic_vector(0 to C_MPLB_NATIVE_DWIDTH-1);
                     Bus2IP_MstWr_dst_rdy_n       : In  std_logic;

            ---------------------------
                    IP2Bus1_MstRd_Req             : out std_logic;
                    IP2Bus1_MstWr_Req             : out std_logic;
                    IP2Bus1_Mst_Addr              : out std_logic_vector(0 to C_MPLB_AWIDTH-1);
                    IP2Bus1_Mst_BE                : out std_logic_vector(0 to (C_MPLB_NATIVE_DWIDTH/8) -1);
                    IP2Bus1_Mst_Lock              : out std_logic;
                    IP2Bus1_Mst_Reset             : out std_logic;

                    -- IP Request Status Reply
                    Bus2IP1_Mst_CmdAck            : In std_logic;
                    Bus2IP1_Mst_Cmplt             : In std_logic;
                    Bus2IP1_Mst_Error             : In std_logic;
                    Bus2IP1_Mst_Rearbitrate       : In std_logic;
                    Bus2IP1_Mst_Cmd_Timeout       : In std_logic;

                   -- IPIC Read data
                    Bus2IP1_MstRd_d               : In std_logic_vector(0 to C_MPLB_NATIVE_DWIDTH-1);
                    Bus2IP1_MstRd_src_rdy_n       : In std_logic;

                   -- IPIC Write data
                    IP2Bus1_MstWr_d               : out std_logic_vector(0 to C_MPLB_NATIVE_DWIDTH-1);
                    Bus2IP1_MstWr_dst_rdy_n       : In  std_logic
            );

end entity DRAC;

architecture Behavioral of PLB_Bridge is

            signal PLB_BE_temp :std_logic_vector(0 to 3);
            type state_type is (stall, First, Second_Bridge, Third_Bridge, forth_Bridge,f ifth_Bridge, Second_Reg, Third_Reg, forth_Reg,
             fifth_Reg, Cache_Write, Cache_Read, Second_Cache, Cache_DDR_Read1,Cache_DDR_Read2,Cache_DDR_Read3);

            type swap_type is (init, run, time_to_stable, time_to_stable2, cache_to_ram1_1, cache_to_ram1_2, cache_to_ram2_1,
            cache_to_ram2_2, cache_to_ram3, ram_to_cache1_1, ram_to_cache1_2, ram_to_cache2, ram_to_cache3);

            type rst_type is (start,stop,idle,incr);
            signal rst_state :rst_type :=idle;
            signal swap : swap_type :=init;
            signal current_state : state_type := First;
```

```
        signal Sl_rdDBus_i                                              :std_logic_vector(0 to C_SPLB_DWIDTH-1);
        signal reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,Address,tag_image_temp,write_coef   :std_logic_vector(0 to 31);
        signal RNW,write_en                                             :std_logic;
        signal DO_i1,DO_i2 :  STD_LOGIC_VECTOR (0 to 31):=(others=> '0');
        signal ADDR_i1,ADDR_i2,adr_swap : STD_LOGIC_VECTOR (0 to 10):=(others=> '0');
        signal DI_i1,DI_i2,data_temp,PLB_wrDBus_temp,PLB_ABus_temp :  STD_LOGIC_VECTOR (0 to 31):=(others=> '0');
        signal EN_i1,EN_i2 :  STD_ULOGIC:='0';
        signal SSR_i : STD_ULOGIC:='0';
        signal WE_i1,WE_i2                                              :  STD_ULOGIC:='0';
        signal SRM_flag, SRM_Timeflag                                   :  STD_LOGIC:='0';
        signal SM_Cnt                                                   : STD_LOGIC_VECTOR (0 to 3):=(others=> '0');
        signal image_hit_flag                                           : STD_LOGIC_VECTOR (0 to 4):=(others=> '0');
        signal Trace_PC_temp                                            : STD_LOGIC_VECTOR (0 to 31):=(others=> '0');
        signal write_enable,write_end,read_end,miss,InstAcc,dataAcc,dmissoverhead_out         : std_logic := '0';
        signal write_flag                                                                     : std_logic:= '0';
        signal swap_en,swap_done                                                              : std_logic := '0';

component  BRAM8 is
  port (
          DO : out STD_LOGIC_VECTOR (0 to 31);
          ADDR : in STD_LOGIC_VECTOR (0 to 10);
          CLK : in STD_ULOGIC;
          DI : in STD_LOGIC_VECTOR (0 to 31);
          EN : in STD_ULOGIC;
          WE : in STD_ULOGIC
  );
end component;


begin
  RAMB8_0 :  BRAM8
        port map (
                DO          => DO_i1,
                ADDR        => Addr_i1,
                CLK         => MPLB_Clk,
                DI          => DI_i1,
                EN          => EN_i1,
                WE          => WE_i1
        );

        RAMB8_1 :  BRAM8
                port map (
                        DO          => DO_i2,
                        ADDR        => Addr_i2,
                        CLK         => MPLB_Clk,
                        DI          => DI_i2,
                        EN          => EN_i2,
                        WE          => WE_i2
        );



IP2BUS_DATA_MUX_PROC : process( MPLB_Clk,SPLB_Rst,PLB_PAValid ) is

        variable adr                       : STD_LOGIC_VECTOR (0 to 10) :=(others => '0') ;
        variable cache_read_reg            :std_logic_vector(0 to 31):=(others => '0') ;
        variable swap_run                  :std_logic:='0' ;
        variable delay_time,delay_time2,delay_time3,cnt,Wcnt,cnt2,image_cnt_stable              :integer range 0 to 4095 := 0;
        variable XC :integer range 0 to 127 := 1;
        variable XC150, XC220, XC270, XC370, XC470, XC600, XC5, XCIhit, XC20, XCR30, XC50, XCR50, XC30, XC40, XC80,
                  XCR1, XCR5, XC100, XC200, XCDhit, XC_nextwrite, XCD220, XCD270        :integer range 0 to 2047 := 1;
        variable cnt1,mem_temp,tag_temp,image_address,temp              :std_logic_vector(0 to 31) :=(others=> '0');
        variable tag                                                    :std_logic_vector(0 to 23) :=(others=> '0');
        variable RMC,RHC,WMC,WHC                                        :std_logic_vector(0 to 31):= (others=> '0');
        variable Chit,CW,BC,CWS,CR                                      : integer range 0 to 4095;
        variable int_index1,int_index2                                  : integer range 0 to 63;
        variable address_temp3                                          : STD_LOGIC_VECTOR (0 to 27);
        variable address_temp2                                          : STD_LOGIC_VECTOR (0 to 1);
        variable address_temp1                                          : STD_LOGIC_VECTOR (0 to 6);
        variable Adr_swap1,Adr_swap2,Adr_swap1_cnt,Adr_swap2_cnt        : STD_LOGIC_VECTOR (0 to 11) := (others=>'0');
variable Adr_swap3,image_address_temp,image_address_tag          : STD_LOGIC_VECTOR (0 to 3) := (others=>'0');
variable branch_flag,write1,writeflag,missflag                   : STD_LOGIC := '0';
variable writetime,misstime                                      : STD_LOGIC_VECTOR (0 to 31);

begin
        if SPLB_Rst = '1' then
            delay_time := 0;
            delay_time2 := 0;
            delay_time3 := 0;
            cnt_image_update := 0;
            image_hit_flag <= (others => '0');
            image_address_tag := (others => '0');
            image_address := (others => '0');
            image_address_temp := "0001";
            image_cnt_stable := 0;
            cnt_image := 0;
            PLB_wrDBus_temp <= (others => '0');
            PLB_ABus_temp <= (others => '0');
            swap_run := '0';
            cache_read_reg := (others => '0');
            PLB_BE_temp <=  (others => '0');
            mem_temp := (others => '0');
            tag_temp := (others => '0');
            data_temp <= (others => '0');
            cnt2 := 0;
            swap_en <= '0';
            Adr_swap1 := (others => '0');
            Adr_swap1_cnt := (others => '0');
            Adr_swap2 := (others => '0');
            Adr_swap2_cnt := (others => '0');
            swap <= init;
            swap_done <= '0';
            address <= (others => '0');
            int_index1 := 0;
            int_index2 := 0;
            RMC := (others=> '0');
            RHC := (others=> '0');
            WMC := (others=> '0');
            WHC := (others=> '0');
            write_en <= '0';
            Addr_i1 <= (others=> '0');
            Addr_i2 <= (others=> '0');
            DI_i1 <= (others=> '0');
            DI_i2 <= (others=> '0');
            SSR_i  <= '0';
            cnt1 := (others =>'0');
            WE_i1 <= '0';
            EN_i1 <= '0';
            WE_i2 <= '0';
            EN_i2 <= '0';
            DI_i2 <= (others=> '0');
            Address_temp1 := (others => '0');
            Address_temp2 := (others => '0');
            Address_temp3 := (others => '0');
            RNW <= '0';
            cnt := 0;
            Sl_addrAck <= '0';
            Sl_SSize <= (others => '0');
            Sl_wait      <= '0';
            Sl_rearbitrate <= '0';
            Sl_wrDAck <= '0';
            Sl_wrComp <= '0';
            Sl_wrBTerm <= '0';
            Sl_rdDBus <= (others => '0');
            Sl_rdWdAddr <= (others => '0');
            Sl_rdDAck  <= '0';
            Sl_rdComp <= '0';
            Sl_rdBTerm <= '0';
            Sl_MBusy <= (others => '0');
            Sl_MWrErr <= (others => '0');
```

```
                Sl_MRdErr <= (others => '0');
                Sl_MIRQ <= (others => '0');
                IP2Bus_MstRd_Req <= '0';
                IP2Bus_MstWr_Req <= '0';
                IP2Bus_Mst_Addr <= (others => '0');
                IP2Bus_Mst_BE <= (others => '0');
                IP2Bus_Mst_Lock <= '0';
                IP2Bus1_MstRd_Req <= '0';
                IP2Bus1_MstWr_Req <= '0';
                IP2Bus1_Mst_Addr <= (others => '0');
                IP2Bus1_Mst_BE <= (others => '0');
                IP2Bus1_Mst_Lock <= '0';
                Sl_rdDBus_i <= (others => '0');
                reg1 <= (others => '0');
                reg2 <= (others => '0');
                reg3 <= (others => '0');
                reg4 <= (others => '0');
                reg5 <= (others => '0');
                reg6 <= (others => '0');
                reg7 <= (others => '0');
                reg8 <= (others => '0');
                reg9 <= (others => '0');
                rst_state <= idle;
                SM_Cnt <= "0000";
                SRMC := (others=> '0');
                SRM_Timeflag <= '0';
                SRM_flag <= '0';
                Trace_PC_temp <= (others => '0');
                WCC := (others => '0');
                Trace_PC_temp <= (others => '0');

        elsif rising_edge(MPLB_Clk) then
                write_out <= PLB_PAValid and (not PLB_RNW);
                miss_out <= miss;
                RData_valid_out <= (PLB_PAValid);
                if writeflag = '1' then
                        writetime := writetime + 1;
                end if;

                if missflag = '1' then
                        misstime := misstime + 1;
                end if;

                if ((C_BRIDGE_BASEADDR (0 to 11) = x"c1c")or (C_BRIDGE_BASEADDR (0 to 11) = x"c3c") or
                (C_BRIDGE_BASEADDR (0 to 11) = x"c5c")) then
                        InstAcc <= '1';
                        DataAcc <= '0';
                        RData_valid_out <= dmissoverhead_out;
                elsif ((C_BRIDGE_BASEADDR (0 to 11) = x"c0c")or (C_BRIDGE_BASEADDR (0 to 11) = x"c2c") or
                (C_BRIDGE_BASEADDR (0 to 11) = x"c4c")) then
                        InstAcc <= '0';
                        DataAcc <= '1';
                end if;

                if (reg1(4 to 31) = x"3333333") then
                        RMC := (others=> '0');
                        RHC := (others=> '0');
                        WHC := (others=> '0');
                        WMC := (others=> '0');
                elsif reg1(4 to 31) = x"1111111" then
                        swap_en <= '1';
                        if DataAcc = '1' then
                                Swap_out <= '1';
                                swap_run := '0';
                        else
                                Swap_out <= '0';
                                swap_run := '1';
                        end if;
                        case reg1 (0 to 3) is
                        when "0001" =>
```

```
                                        Adr_swap1_cnt := "000000010000";
                                        Adr_swap2_cnt := "000001000000";


                        when "0010" =>
                                        Adr_swap1_cnt := "000001000000";
                        Adr_swap2_cnt := "000100000000";

                        when "0011" =>
                                        Adr_swap1_cnt := "000010000000";
                                        Adr_swap2_cnt := "001000000000";

                        when "0100" =>
                                        Adr_swap1_cnt := "000100000000";
                                        Adr_swap2_cnt := "010000000000";

                        when "0101" =>
                                        Adr_swap1_cnt := "001000000000";
                                        Adr_swap2_cnt := "100000000000";

                        when others =>
                                        Adr_swap1_cnt := "001000000000";
                                        Adr_swap2_cnt := "100000000000";

                        end case;
                        IP2Bus1_Mst_BE <= "1111";
                        EN_i1 <= '1';
                        EN_i2 <= '1';
                        Addr_i1 <= (others => '0');
                        Addr_i2 <= (others => '0');
        end if;
        case rst_state is
        when idle =>
                        rst_state <= start;
                        adr := (others => '0');

        when start =>
                        rst_state <= incr;
                        DI_i1 <= (others => '0');
                        DI_i2 <= (others =>'0');
--                      SSR_i <= '1';
                        Addr_i1 <= adr;
                        Addr_i2 <= adr;
                        EN_i1 <= '1';
                        EN_i2 <= '1';
                        WE_i1 <= '1';
                        WE_i2 <= '1';

        when incr =>
                        if cnt = 1 then
                                        cnt := 0;
                                        if adr = "11111111111" then
                                                        rst_state <= stop;
                                        else
                                                        adr := adr+1;
                                                        rst_state <= start;
                                        end if;
                        else
                                        cnt := cnt +1;
                        end if;

        when stop =>
                        rst_state <= stop;

        when others =>
                        rst_state <= stop;
        end case;

        case current_state is
        when First =>
        SSR_i <= '0';
        Sl_rdDAck <= '0';
        Sl_rdComp <= '0';
        Sl_rdDBus <= (others => '0');
        Sl_wrDAck <= '0';
        Sl_wrComp <= '0';
        Sl_MBusy <= (others => '0');
        Sl_rdDBus <= (others => '0');
        if ((PLB_PAValid = '1') and (PLB_ABus (0 to 19) = C_BRIDGE_BASEADDR (0 to 19))) then
                Sl_wait <= '1';
                current_state <= Second_Reg;
                if (PLB_RNW = '0') then
                RNW <= '0';
                case PLB_ABus (16 to 31) is
                when x"0000" =>
                        reg1 <= PLB_wrDBus;
                when x"0004" =>
                        reg2 <= C_BRIDGE_BASEADDR (0 to 27) & PLB_wrDBus(28 to 31);
                when x"00F0" =>
                        write_coef <= PLB_wrDBus;
                when x"0080" =>
                        XCR1 := conv_integer(PLB_wrDBus (20 to 31));
                when x"0084" =>
                        XCR30 := conv_integer(PLB_wrDBus (20 to 31));
                when x"0088" =>
                        XCR5 := conv_integer(PLB_wrDBus (20 to 31));
                when x"008C" =>
                        XC80 := conv_integer(PLB_wrDBus (20 to 31));
                when x"0090" =>
                        XC20 := conv_integer(PLB_wrDBus (20 to 31));
                when x"0094" =>
                        XC50 := conv_integer(PLB_wrDBus (20 to 31));
                when x"0098" =>
                        XC200 := conv_integer(PLB_wrDBus (20 to 31));
                when x"009C" =>
                        XCR50 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00A0" =>
                        XC5 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00A4" =>
                        XCIhit := conv_integer(PLB_wrDBus (20 to 31));
                when x"00A8" =>
                        XC30 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00AC" =>
                        XC40 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00B0" =>
                        XC100 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00B4" =>
                        XC150 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00B8" =>
                        XC220 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00BC" =>
                        XC270 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00C0" =>
                        XC370 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00C4" =>
                        XC470 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00C8" =>
                        XC600 := conv_integer(PLB_wrDBus (20 to 31));
                when x"00d8" =>
                        XCD220 := conv_integer(PLB_wrDBus (20 to 31));
                when others =>
                        null;
                end case;
        else
                RNW <= '1';
        end if;
elsif ((PLB_PAValid = '1') and (PLB_ABus (0 to 3) = "1001")) then
        Sl_wait <= '1';
        if (swap_run = '1' or Swap_in = '1') then
                En_i1 <= '1';
                En_i2 <= '1';
                if cnt = 10 then
                        Sl_rdDBus_i <= x"b8000000";

                        current_state <= forth_Bridge;
                        Sl_addrAck <= '1';
                        cnt := 0;
                else
                        cnt := cnt +1;
                end if;
        else
                case reg1 (0 to 3) is
                when "0001" =>
                        Addr_i1 <=  "0000000" & PLB_ABus(24 to 27);
                        Addr_i2 <= "00000" & PLB_ABus(24 to 29);
                        address(4 to 27) <= PLB_ABus(4 to 27);
                        address(0 to 3) <= (others => '0');

                when "0010" =>
                        Addr_i1 <=  "00000" & PLB_ABus(22 to 27);
                        Addr_i2 <= "000" & PLB_ABus(22 to 29);
                        address(4 to 27) <= PLB_ABus(4 to 27);
                        address(0 to 3) <= (others => '0');

                when "0011" =>
                        Addr_i1 <=  "0000" & PLB_ABus(21 to 27);
                        Addr_i2 <= "00" & PLB_ABus(21 to 29);
                        address(4 to 27) <= PLB_ABus(4 to 27);
                        address(0 to 3) <= (others => '0');

                when "0100" =>
                        Addr_i1 <=  "000" & PLB_ABus(20 to 27);
                        Addr_i2 <= "0" & PLB_ABus(20 to 29);
                        address(4 to 27) <= PLB_ABus(4 to 27);
                        address(0 to 3) <= (others => '0');

                when "0101" =>
                        Addr_i1 <=  "00" & PLB_ABus(19 to 27);
                        Addr_i2 <= PLB_ABus(19 to 29);
                        address(4 to 27) <= PLB_ABus(4 to 27);
                        address(0 to 3) <= (others => '0');

                when others =>
                        Addr_i1 <=  "00" & PLB_ABus(19 to 27);
                        Addr_i2 <= PLB_ABus(19 to 29);
                        address(4 to 27) <= PLB_ABus(4 to 27);
                        address(0 to 3) <= (others => '0');
                end case;
                if ((reg1(4 to 31) = x"aaaaaaa") and (PLB_RNW = '1'))then
                        if ((InstAcc = '1')) then
                                delay_time3 := 0;
                        else
                                delay_time3 := 0;
                                delay_time2 := XCD220;
                        end if;
                                delay_time := 1;
                                WE_i1 <= '0';
                                EN_i1 <= '1';
                                WE_i2 <= '0';
                                EN_i2 <= '1';
                                current_state <= Second_cache;
                else
                        if PLB_RNW = '1' then
                                IP2Bus_MstRd_Req <= '1';
                                current_state <= Second_Bridge;
                                IP2Bus_Mst_Addr <= PLB_ABus;
                                IP2Bus_Mst_BE <= PLB_BE;
                                IP2Bus_Mst_Lock <= '0';
                        else
                                PLB_BE_temp <= PLB_BE;
```

```
                                                PLB_wrDBus_temp <= PLB_wrDBus;
                                                PLB_ABus_temp <= PLB_ABus;
                                                address(4 to 27) <= PLB_ABus(4 to 27);
                                                address(0 to 3) <= (others => '0');
                                                cnt_image := 0;
                                                EN_i1 <= '1';
                                                EN_i2 <= '1';
                                                WE_i1 <= '0';
                                                WE_i2 <= '0';
                                                current_state <= Second_cache;
                                        end if;
                                end if;
                        end if;

                else
                        current_state <= First;
                end if;

                when Second_Cache =>
                        if cnt = delay_time then
                                tag_temp := DO_i1;
                                mem_temp := DO_i2;
                                cnt :=0;
                                if PLB_RNW = '1' then
                                        RNW <= '1';
                                        current_state <= Cache_Read;
                                else
                                        RNW <= '0';
                                        current_state <= Cache_Write;
                                end if;
                        else
                                cnt := cnt +1;
                                current_state <= Second_Cache;
                        end if;
                when Cache_Read =>
                        if ((tag_temp(0 to 3)="1111") and (address(4 to 27) = tag_temp(4 to 27))) then
                                RHC := RHC +1;
                                if (InstAcc = '1') then
                                        Chit := Chit + 1;
                                        if BC = 0 then
                                                if ((Chit = 1) and ((mem_temp (0 to 5)  = "110100") or
                                                (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
                                                (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
                                                (mem_temp (0 to 5)  = "110110"))) then
                                                        CWS := 0;
                                                        delay_time2 := (XC270) ;
                                                        CW := 1;
                                                        CW_out <= x"1";
                                                elsif ((CW = 0) and ((mem_temp (0 to 5)  = "110100") or
                                                (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
                                                (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
                                                (mem_temp (0 to 5)  = "110110"))) then
                                                        CWS := 0;
                                                        CW := 1;
                                                        CW_out <= x"1";
                                                elsif ((CW = 1) and ((mem_temp (0 to 5)  = "110100") or
                                                (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
                                                (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
                                                (mem_temp (0 to 5)  = "110110"))) then
                                                        if (CWS = 0) then
                                                                delay_time2 := (XC370);
                                                        else
                                                                delay_time2 := (XC270) ;
                                                        end if;
                                                        CW := 2;
                                                        CW_out <= x"2";
                                                elsif ((CW = 2) and ((mem_temp (0 to 5)  = "110100") or
                                                (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
                                                (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
                                                (mem_temp (0 to 5)  = "110110"))) then
                if (CWS = 0) then
                        writeflag :='1';
                        delay_time3 := XC40;
                        delay_time2 := (XC470) ;
                else
                        delay_time2 := (XC370);
                        CWS := 2;
                end if;
                CW := 3;
                CW_out <= x"3";
        elsif ((CW = 3) and ((mem_temp (0 to 5)  = "110100") or
        (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
        (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
        (mem_temp (0 to 5)  = "110110"))) then
                if (CWS = 0) then
                        writeflag :='1';
                        delay_time3 := XC50;
                        delay_time2 := (XC600) ;
                elsif (CWS = 2) then
                        writeflag :='1';
                        delay_time3 := XC40;
                        delay_time2 := (XC470) ;
                else
                        delay_time2 := (XC470) ;
                        CWS := 2;
                end if;
                        CW := 4;
                        CW_out <= x"4";
        elsif ((CW = 4) and ((mem_temp (0 to 5)  = "110100") or
        (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
        (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
        (mem_temp (0 to 5)  = "110110"))) then
                if (CWS = 0) then
                        delay_time3 := XC100;
                        elsif (CWS = 2) then
                        delay_time3 := XC50;
                end if;
                        writeflag :='1';
                        CWS := 0;
                        delay_time2 := (XC370) ;
                        CW := 4;
                        CW_out <= x"4";
        elsif ((CW = 5) and ((mem_temp (0 to 5)  = "110100") or
        (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
        (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
        (mem_temp (0 to 5)  = "110110"))) then
                CW := 6;
                CW_out <= x"6";
                CWS := 0;
                delay_time2 := (XC220) ;
        elsif ((CW = 6) and ((mem_temp (0 to 5)  = "110100") or
        (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
        (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
        (mem_temp (0 to 5)  = "110110"))) then
                CW := 7;
                CW_out <= x"7";
                if (CWS = 0) then
                        delay_time3 := (XC80) ;
                        writeflag :='1';
                end if;
                delay_time2 := (XC370) ;
        elsif ((CW = 7) and ((mem_temp (0 to 5)  = "110100") or
        (mem_temp (0 to 5)  = "111100") or (mem_temp (0 to 5)  = "110101") or
        (mem_temp (0 to 5)  = "111101") or (mem_temp (0 to 5)  = "111110") or
        (mem_temp (0 to 5)  = "110110"))) then
                CW := 7;
                CW_out <= x"7";
                if (CWS = 0) then
                        delay_time3 := (XC100) ;
                else
                        delay_time3 := (XC80) ;
                end if;
                writeflag :='1';
                delay_time2 := (XC370) ;
                CWS := 0;
        else
                if (mem_temp (0 to 5)  = "101110") then
                        BC := 1;
                end if;
                CWS := CWS +1;
                if (CWS > 15) then
                        CW := 0;
                        CW_out <= x"0";
                elsif ((CWS /= 1) and (CW = 4 or CW = 5 or CW = 7)) then
                        CW := 5;
                        CW_out <= x"5";
                elsif ((CWS /= 1)) then
                        CW := 0;
                        CW_out <= x"0";
                elsif (CW = 1) then
                        writeflag :='1';
                        write1 :='1';
                        CR := 0;
                        CW := 2;
                        CW_out <= x"2";
                elsif ((CW = 2)) then
                        writeflag :='1';
                        delay_time3 := XC40;
                        CW := 3;
                        CR := 0;
                        CW_out <= x"3";
                elsif ((CW = 3)) then
                        writeflag :='1';
                        delay_time3 := XC50;
                        CW := 4;
                        CR := 0;
                        CW_out <= x"4";
                elsif ((CW = 4)) then
                        writeflag :='1';
                        CR := 0;
                        CW := 4;
                        CW_out <= x"4";
                        delay_time3 := (XC100) ;
                elsif ((CW = 6)) then
                        writeflag :='1';
                        write1 :='1';
                        CW := 7;
                        CR := 3;
                        CW_out <= x"7";
                elsif ((CW = 7)) then
                        writeflag :='1';
                        CW := 7;
                        CW_out <= x"7";
                        CR := 3;
                        delay_time3 := (XC100) ;
                end if;
        elsif BC = 1 then
                BC := 2;
        else
                BC := 0;
        end if;
    end if;
        current_state <= data_stall;
        Sl_rdDBus_i <= mem_temp;
else
    if DataAcc = '1' then
        miss <= '1';
        delay_time2 := XCD220;
        missflag := '0';
    else
```

```
                                        missflag := '1';
                                        miss <= '0';
                                end if;

                                EN_i1 <= '1';
                                WE_i1 <= '1';
                                DI_i1 <=  "1111"& address(4 to 27)& "0000";
                                RMC := RMC +1;
                                current_state <= Cache_DDR_Read1 ;
                                IP2Bus_MstRd_Req <= '1';
                                IP2Bus_Mst_Addr <= PLB_ABus(0 to 27) &"0000";
                                IP2Bus_Mst_BE <= PLB_BE;
                                IP2Bus_Mst_Lock <= '0';
                                address_temp3 := PLB_ABus(0 to 27);
                                address_temp2 := "00";
                        end if;
                end if;

        when stall =>
                if cnt = delay_time2 then
                        cnt := 0;
                        current_state <= Cache_DDR3;
                        Sl_addrAck <= '1';
                        missflag := '0';
                        Chit := 0;
                        if ((InstACC = '1') and ((Sl_rdDBus_i (0 to 5)  = "110100") or (Sl_rdDBus_i (0 to 5)  = "111100")
                        or (Sl_rdDBus_i (0 to 5)  = "110101") or (Sl_rdDBus_i (0 to 5)  = "111101") or
                        (Sl_rdDBus_i (0 to 5)  = "111110") or (Sl_rdDBus_i (0 to 5)  = "110110"))) then
                                delay_time2 := (XC270);
                                CW := 1;
                                CR := 0;
                                CW_out <= x"1";
                                CWS := 0;
                        else
                                delay_time2 := (XC150);
                                CW := 0;
                                CW_out <= x"0";
                                BC := 0;
                                CWS := 0;
                        end if;
                else
                        cnt := cnt+1;
                        current_state <= stall;
                end if;

        when Cache_DDR_Read1 =>
                if ( Bus2IP_Mst_CmdAck = '1') then
                        current_state <= Cache_DDR_Read2;
                elsif ( Bus2IP_Mst_Cmplt = '1' ) then
                        EN_i2 <= '1';
                        WE_i2 <= '1';
                        IP2Bus_MstRd_Req <= '0';
                        Addr_i2 <= Addr_i1(2 to 10) & address_temp2;
                        DI_i2 <= Bus2IP_MstRd_d;
                        if address_temp2 = PLB_ABus(28 to 29) then
                                Sl_rdDBus_i <= Bus2IP_MstRd_d;
                        end if;
                        if (address_temp2 = "11" ) then
                                current_state <= stall;
                        else
                                current_state <= Cache_DDR_Read3;
                        end if;
                else
                        WE_i2 <= '0';
                        current_state <= Cache_DDR_Read1;
                end if;

        when Cache_DDR_Read2 =>
                if ( Bus2IP_Mst_Cmplt = '1' ) then
                        EN_i2 <= '1';
                        WE_i2 <= '1';
                        IP2Bus_MstRd_Req <= '0';
                        Addr_i2 <= Addr_i1(2 to 10) & address_temp2;
                        DI_i2       <= Bus2IP_MstRd_d;
                        if address_temp2 = PLB_ABus(28 to 29) then
                                Sl_rdDBus_i <= Bus2IP_MstRd_d;
                        end if;
                        if (address_temp2 = "11") then
                                current_state <= stall;
                        else
                                current_state <= Cache_DDR_Read3;
                        end if;
                else
                        WE_i2 <= '0';
                        current_state <= Cache_DDR_Read2;
                end if;

        when data_stall =>
                if write1 = '1' then
                        delay_time3 := XCIhit;
                end if;
                if (RData_valid_in = '0' and miss_in = '0') or (Chit < 4) then
                        if cnt = delay_time3 then
                                writeflag := '0';
                                missflag := '0';
                                write1 := '0';
                                cnt := 0;
                                current_state <= Cache_DDR3;
                                Sl_addrAck <= '1';
                                branch_flag := '0';
                        else
                                cnt := cnt +1;
                                current_state <= data_stall;
                        end if;
                elsif miss_in = '1' then
                        missflag := '1';
                        CW := 0;
                        CWS := 0;
                        delay_time3 := XCR1;
                        current_state <= data_stall;
                else
                        current_state <= data_stall;
                end if;
        when Cache_Write =>
                if ((tag_temp(0 to 3)="1111") and (address(4 to 27) = tag_temp(4 to 27))) then
                        case PLB_BE_temp is
                        when "0001" =>
                                EN_i2 <= '1';
                                WE_i2 <= '1';
                                DI_i2 (0 to 23)<= mem_temp(0 to 23);
                                DI_i2(24 to 31) <= PLB_wrDBus_temp(24 to 31);

                        when "0010" =>
                                EN_i2 <= '1';
                                WE_i2 <= '1';
                                DI_i2 (0 to 15)<= mem_temp(0 to 15);
                                DI_i2(16 to 23) <= PLB_wrDBus_temp(16 to 23);
                                DI_i2 (24 to 31)<= mem_temp(24 to 31);

                        when "0011"=>
                                EN_i2 <= '1';
                                WE_i2 <= '1';
                                DI_i2 (0 to 15)<= mem_temp(0 to 15);
                                DI_i2(16 to 31) <= PLB_wrDBus_temp(16 to 31);

                        when"0100" =>
                                EN_i2 <= '1';
                                WE_i2 <= '1';
                                DI_i2 (0 to 7)<= mem_temp(0 to 7);
                                DI_i2(8 to 15) <= PLB_wrDBus_temp(8 to 15);
```

```
            DI_i2 (16 to 31)<= mem_temp(16 to 31);

when "0101" =>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2 (0 to 7)<= mem_temp(0 to 7);
            DI_i2(8 to 15) <= PLB_wrDBus_temp(8 to 15);
            DI_i2 (16 to 23)<= mem_temp(16 to 23);
            DI_i2(24 to 31) <= PLB_wrDBus_temp(24 to 31);

when "0110"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2 (0 to 7)<= mem_temp(0 to 7);
            DI_i2(8 to 23) <= PLB_wrDBus_temp(8 to 23);
            DI_i2(24 to 31) <= mem_temp(24 to 31);

when "0111"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2 (0 to 7)<= mem_temp(0 to 7);
            DI_i2(8 to 31) <= PLB_wrDBus_temp(8 to 31);

when "1000"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 7) <= PLB_wrDBus_temp(0 to 7);
            DI_i2(8 to 31) <= mem_temp(8 to 31);

when "1001"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 7) <= PLB_wrDBus_temp(0 to 7);
            DI_i2(8 to 23) <= mem_temp(8 to 23);
            DI_i2(24 to 31) <= PLB_wrDBus_temp(24 to 31);

when "1010"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 7) <= PLB_wrDBus_temp(0 to 7);
            DI_i2(8 to 15) <= mem_temp(8 to 15);
            DI_i2(16 to 23) <= PLB_wrDBus_temp(16 to 23);
            DI_i2(23 to 31) <= mem_temp(23 to 31);

when "1011"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 7) <= PLB_wrDBus_temp(0 to 7);
            DI_i2(8 to 15) <= mem_temp(8 to 15);
            DI_i2(16 to 31) <= PLB_wrDBus_temp(16 to 31);

when "1100"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 15) <= PLB_wrDBus_temp(0 to 15);
            DI_i2(16 to 31) <= mem_temp(16 to 31);

when "1101"=>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 15) <= PLB_wrDBus_temp(0 to 15);
            DI_i2(16 to 23) <= mem_temp(16 to 23);
            DI_i2(24 to 31) <= PLB_wrDBus_temp(24 to 31);

when "1110" =>
            EN_i2 <= '1';
            WE_i2 <= '1';
            DI_i2(0 to 23) <= PLB_wrDBus_temp(0 to 23);
            DI_i2(24 to 31) <= mem_temp(24 to 31);
```

```
                                        when "1111" =>
                                                EN_i2 <= '1';
                                                WE_i2 <= '1';
                                                DI_i2 <= PLB_wrDBus_temp;
                                        when others =>
                                                EN_i2 <= '0';
                                                WE_i2 <= '0';
                                        end case;
                                end if;
                                IP2Bus_MstWr_d <= PLB_wrDBus_temp;
                                IP2Bus_Mst_Addr <= PLB_ABus_temp;
                                IP2Bus_Mst_BE <= PLB_BE_temp;
                                IP2Bus_Mst_Lock <= '0';
                                IP2Bus_MstWr_Req <= '1';
                                if reg1(4 to 31) = x"aaaaaaa" then
                                        WHC := WHC +1;
                                end if;
                                current_state <= Second_Bridge;

                        when Cache_DDR3 =>
                                WE_i1 <= '0';
                                WE_i2 <= '0';
                                En_i1 <= '0';
                                En_i2 <= '0';
                                Sl_MBusy(0) <= '1';
                                Sl_addrAck <= '0';
                                Sl_wait <= '0';
                                current_state <= Cache_DDR4;

                        when Cache_DDR4 =>
                                read_end <= '1';
                                miss <= '0';
                                if PLB_RNW = '1' then
                                        Sl_rdDAck <= '1';
                                        Sl_rdComp <= '1';
                                        Sl_rdDBus <= Sl_rdDBus_i;
                                        if SRM_flag ='1' then
                                                SM_Cnt <= "0000";
                                        end if;

                                else
                                        Sl_wrDAck <= '1';
                                        _wrComp <= '1';
                                end if;
                                current_state <= first;
---------------------------------------------------------------------------------
-------------------------------End---Cache-----------------------------------
---------------------------------------------------------------------------------



 ---------------------------------------------------------------------------------
-----------------------------Slave Register-----------------------------------
 ---------------------------------------------------------------------------------
                                when Second_Reg =>
                                        if cnt = 12 then
                                                current_state <= third_Reg;
                                                cnt := 0;
                                                else
                                                current_state <= Second_Reg;
                                                cnt := cnt +1;
                                                end if;

                                when third_Reg =>
                                        if (RNW = '1') then
                                                if  PLB_ABus(24 to 31) = x"00" then
                                                        Sl_rdDBus_i <= reg1;
                                                elsif PLB_ABus(24 to 31) = x"04" then
                                                        Sl_rdDBus_i <= reg2;
                                                elsif PLB_ABus(24 to 31) = x"08" then
                                                        Sl_rdDBus_i <= reg3;
                                                elsif PLB_ABus(24 to 31) = x"0c" then
                                                        Sl_rdDBus_i <= reg4;
                                                elsif PLB_ABus(24 to 31) = x"30" then
                                                        Sl_rdDBus_i <= reg5;
                                                elsif PLB_ABus(24 to 31) = x"34" then
                                                        Sl_rdDBus_i <= reg6;
                                                elsif PLB_ABus(24 to 31) = x"10" then
                                                        Sl_rdDBus_i <= tag_temp;
                                                elsif PLB_ABus(24 to 31) = x"14" then
                                                        Sl_rdDBus_i <= mem_temp;
                                                elsif PLB_ABus(24 to 31) = x"20" then
                                                        Sl_rdDBus_i <= reg5;
                                                elsif PLB_ABus(24 to 31) = x"24" then
                                                        Sl_rdDBus_i <= reg6;
                                                elsif PLB_ABus(24 to 31) = x"28" then
                                                        Sl_rdDBus_i <= reg7;
                                                elsif PLB_ABus(24 to 31) = x"40" then
                                                        Sl_rdDBus_i <= reg8;
                                                elsif PLB_ABus(24 to 31) = x"44" then
                                                        Sl_rdDBus_i <= reg9;
                                                elsif PLB_ABus(24 to 31) = x"48" then
                                                        Sl_rdDBus_i <= reg10;
                                                elsif PLB_ABus(24 to 31) = x"4c" then
                                                        Sl_rdDBus_i <= reg11;
                                                elsif PLB_ABus(24 to 31) = x"50" then
                                                        Sl_rdDBus_i <= reg21;
                                                elsif PLB_ABus(24 to 31) = x"54" then
                                                        Sl_rdDBus_i <= reg22;
                                                elsif PLB_ABus(24 to 31) = x"58" then
                                                        Sl_rdDBus_i <= reg23;
                                                elsif PLB_ABus(24 to 31) = x"5c" then
                                                        Sl_rdDBus_i <= reg24;
                                                elsif PLB_ABus(24 to 31) = x"f0" then
                                                        Sl_rdDBus_i <= write_coef;
                                                elsif PLB_ABus(24 to 31) = x"60" then
                                                        Sl_rdDBus_i <= misstime;
                                                elsif PLB_ABus(24 to 31) = x"64" then
                                                        Sl_rdDBus_i <= writetime;
                                                end if;
                                        end if;
                                        Sl_addrAck <= '1';
                                        current_state <= Forth_Reg;

                                when Forth_Reg =>
                                        Sl_MBusy(0) <= '1';
                                        Sl_addrAck <= '0';
                                        Sl_wait <= '0';
                                        current_state <= Fifth_Reg;

                                when Fifth_Reg =>
                                        if RNW = '1' then
                                                Sl_rdDAck <= '1';
                                                Sl_rdComp <= '1';
                                                Sl_rdDBus <= Sl_rdDBus_i;
                                        else
                                                Sl_wrDAck <= '1';
                                                Sl_wrComp <= '1';
                                        end if;
                                        current_state <= first;


 --------------------------------------------------------------------------------
 ----------------------------------End--Slave Register-----------------------
 --------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-----------------------------------Bridge-------------------------------------
 --------------------------------------------------------------------------------
                                when Second_Bridge =>
                                        if ( Bus2IP_Mst_CmdAck = '1' and Bus2IP_Mst_Cmplt = '0' ) then
```

```
                                                  current_state <= third_Bridge;
                                                elsif ( Bus2IP_Mst_Cmplt = '1' ) then
                                                          if PLB_RNW ='1' then
                                                                  Sl_rdDBus_i <= Bus2IP_MstRd_d;
                                                          end if;
                                                          Sl_addrAck <= '1';
                                                          current_state <= forth_Bridge;
                                                          IP2Bus_MstWr_Req <= '0';
                                                          IP2Bus_MstRd_Req <= '0';
                                                else
                                                  current_state <= Second_Bridge;
                                                end if;

                                        when third_Bridge =>
                                                if ( Bus2IP_Mst_Cmplt = '1' ) then
                                                        if PLB_RNW ='1' then
                                                                Sl_rdDBus_i <= Bus2IP_MstRd_d;
                                                        end if;
                                                        current_state <= forth_Bridge;
                                                        Sl_addrAck <= '1';
                                                        IP2Bus_MstWr_Req <= '0';
                                                        IP2Bus_MstRd_Req <= '0';
                                                else
                                                        current_state <= third_Bridge;
                                                end if;

                                        when forth_Bridge =>
                                                Sl_MBusy(0) <= '1';
                                                Sl_addrAck <= '0';
                                                Sl_wait <= '0';
                                                current_state <= fifth_Bridge;


                                        when fifth_Bridge =>
                                                WE_i1 <= '0';
                                                WE_i2 <= '0';
                                                if PLB_RNW = '1' then
                                                        Sl_rdDAck <= '1';
                                                        Sl_rdComp <= '1';
                                                        Sl_rdDBus <= Sl_rdDBus_i;
                                                else
                                                        Sl_wrDAck <= '1';
                                                        Sl_wrComp <= '1';
                                                end if;
                                                current_state <= first;
--------------------------------------------------------------------------------
------------------------------End  Bridge-------------------------------------
--------------------------------------------------------------------------------
                                        when others =>
                                                current_state <= First;
                                        end case;


--------------------------------------------------------------------------------
------------------------------Swap Madule----------------------------------
--------------------------------------------------------------------------------
                case swap is
                        when init =>
                                if swap_en = '1' then
                                        Adr_swap1 := (others => '0');
                                        Adr_swap2 := (others => '0');
                                        Addr_i1 <= (others => '0');
                                        Addr_i2 <= (others => '0');
                                        swap <= run;
                                else
                                        swap_run := '0';
                                        Swap_out <= '0';
                                        swap <= init;
                                        IP2Bus1_MstWr_Req <= '0';
                                        IP2Bus1_MstRd_Req <= '0';
```

```
                    Adr_swap1 := (others => '0');
                    Adr_swap2 := (others => '0');
            end if;

            when run =>
                    if reg2 (0 to 11) = x"c4c" then
                            adr_swap3 (0) := '0';
                    else
                            adr_swap3 (0) := '1';
                    end if;
                    if reg2(28 to 31) = "0000" then
                            swap <= cache_to_ram1_1;
                            adr_swap3 (1 to 3) := "000";
                    elsif reg2 (28 to 31) = "0001" then
                            swap <= ram_to_cache1_1;
                            adr_swap3 (1 to 3) := "000";
                    elsif reg2 (28 to 31) = "0010" then
                            swap <= cache_to_ram1_1;
                            adr_swap3 (1 to 3) := "001";
                    elsif reg2 (28 to 31) = "0011" then
                            swap <= ram_to_cache1_1;
                            adr_swap3 (1 to 3) := "001";
                    elsif reg2 (28 to 31) = "0100" then
                            swap <= cache_to_ram1_1;
                            adr_swap3 (1 to 3) := "010";
                    elsif reg2 (28 to 31) = "0101" then
                            swap <= ram_to_cache1_1;
                            adr_swap3 (1 to 3) := "010";
                    elsif reg2 (28 to 31) = "0110" then
                            swap <= cache_to_ram1_1;
                            adr_swap3 (1 to 3) := "011";
                    elsif reg2 (28 to 31) = "0111" then
                            swap <= ram_to_cache1_1;
                            adr_swap3 (1 to 3) := "011";
                    elsif reg2(28 to 31) = "1000" then
                            swap <= cache_to_ram1_1;
                            adr_swap3 (1 to 3) := "100";
                    elsif reg2 (28 to 31) = "1001" then
                            swap <= ram_to_cache1_1;
                            adr_swap3 (1 to 3) := "100";
                    else
                            swap <= init;
                    end if;

    when cache_to_ram1_1 =>
            reg1 <= (others => '0');
            swap_en <= '0';
            IP2Bus1_MstWr_Req <= '1';
            IP2Bus1_MstWr_d <= DO_i2;
            IP2Bus1_Mst_Addr <= x"92" &"000" & adr_swap3 &"1000" & Adr_swap2 (1 to 11)& "00";
            swap <= cache_to_ram2_1;

    when cache_to_ram1_2 =>
            if Adr_swap1 = Adr_swap1_cnt then
                    swap <= init;
                    swap_done <= '1';
            else
                    IP2Bus1_MstWr_Req <= '1';
                    IP2Bus1_MstWr_d <= DO_i1;
                    IP2Bus1_Mst_Addr <=x"92" &"000" & adr_swap3 &"0000" & Adr_swap1 (1 to 11)& "00";
                    swap <= cache_to_ram2_2;
            end if;


    when cache_to_ram2_1 =>
            if ( Bus2IP1_Mst_CmdAck = '1' and Bus2IP1_Mst_Cmplt= '0') then
                    swap <= cache_to_ram3;
                    Adr_swap2 := Adr_swap2 +1;
                    Addr_i2 <= Adr_swap2(1 to 11);
            elsif Bus2IP1_Mst_Cmplt = '1' then
```

```
                        IP2Bus1_MstWr_Req <= '0';
                        swap <= time_to_stable;
                else
                        swap <= cache_to_ram2_1;

                end if;

        when cache_to_ram2_2 =>
                if ( Bus2IP1_Mst_CmdAck = '1' and Bus2IP1_Mst_Cmplt= '0') then
                        swap <= cache_to_ram3;
                        Adr_swap1 := Adr_swap1 +1;
                        Addr_i1 <= Adr_swap1(1 to 11);
                elsif Bus2IP1_Mst_Cmplt = '1' then
                        IP2Bus1_MstWr_Req <= '0';
                        swap <= time_to_stable;
                else
                        swap <= cache_to_ram2_2;

                end if;

        when cache_to_ram3 =>
                if ( Bus2IP1_Mst_Cmplt = '1' ) then
                        IP2Bus1_MstWr_Req <= '0';
                        swap <= time_to_stable;
                else
                        swap <= cache_to_ram3;
                end if;

        when time_to_stable =>
                if cnt2 = 12 then
                        cnt2 := 0;
                        if Adr_swap2 = Adr_swap2_cnt then
                                swap <= cache_to_ram1_2;
                        else
                                swap <= cache_to_ram1_1;
                        end if;
                else
                        cnt2 := cnt2+1;
                        swap<= time_to_stable;
                end if;

        when ram_to_cache1_1 =>
                reg1 <= (others => '0');
                swap_en <= '0';
                if Adr_swap2 = Adr_swap2_cnt then
                        swap <= ram_to_cache1_2;
                        swap_done <= '1';
                        WE_i1<= '0';
                        WE_i2<= '0';
                else
                        IP2Bus1_MstRd_Req <= '1';
                        IP2Bus1_Mst_Addr <= x"92" &"000" & adr_swap3 &"1000" & Adr_swap2 (1 to 11)& "00";
                        swap <= ram_to_cache2;
                        Addr_i2 <= Adr_swap2(1 to 11);
                end if;

        when ram_to_cache1_2 =>
                if Adr_swap1 = Adr_swap1_cnt then
                        swap <= init;
                        swap_done <= '1';
                        WE_i1<= '0';
                        WE_i2<= '0';
                else
                        IP2Bus1_MstRd_Req <= '1';
                        IP2Bus1_Mst_Addr <= x"92" &"000" & adr_swap3 &"0000" & Adr_swap1 (1 to 11)& "00";
                        swap <= ram_to_cache2;
                        Addr_i1 <= Adr_swap1(1 to 11);
                end if;
```

```
                        when ram_to_cache2 =>
                                WE_i1<= '0';
                                WE_i2<= '0';
                                if ( Bus2IP1_Mst_CmdAck = '1' and Bus2IP1_Mst_Cmplt = '0') then

                                        swap <= ram_to_cache3;
                                elsif ( Bus2IP1_Mst_Cmplt = '1' ) then
                                        IP2Bus1_MstRd_Req <= '0';
                                        swap <= time_to_stable2;
                                        data_temp <= Bus2IP1_MstRd_d;
                                        swap <= time_to_stable2;
                                else
                                        swap <= ram_to_cache2;

                                end if;


                        when ram_to_cache3 =>
                                if ( Bus2IP1_Mst_Cmplt = '1' ) then
                                        IP2Bus1_MstRd_Req <= '0';
                                        swap <= time_to_stable2;
                                        data_temp <= Bus2IP1_MstRd_d;
                                        swap <= time_to_stable2;
                                else
                                        swap <= ram_to_cache3;
                                end if;


                        when time_to_stable2 =>
                                if cnt2 = 5 then
                                        cnt2 := 0;
                                        if Adr_swap2 = Adr_swap2_cnt then
                                                WE_i2<= '0';
                                                swap <= ram_to_cache1_2;
                                                DI_i1        <= data_temp;
                                                WE_i1 <= '1';
                                                Adr_swap1 := Adr_swap1 +1;
                                        else
                                                swap <= ram_to_cache1_1;
                                                DI_i2        <= data_temp;
                                                WE_i1<= '0';
                                                WE_i2 <= '1';
                                                Adr_swap2 := Adr_swap2 +1;
                                        end if;
                                else
                                        swap <= time_to_stable2;
                                        cnt2 := cnt2 +1;
                                end if;

                        when others =>
                                swap <= init;
                end case;

        end if;

        end process IP2BUS_DATA_MUX_PROC;


                Sl_SSize <= "00";
                Sl_rearbitrate <= '0';
                Sl_wrBTerm <= '0';
                Sl_rdWdAddr <= (others => '0');
                Sl_rdBTerm <= '0';
                Sl_MBusy(1 to C_SPLB_NUM_MASTERS-1) <= (others => '0');
                Sl_MWrErr (0 to C_SPLB_NUM_MASTERS-1)<= (others => '0');
                Sl_MIRQ(0 to C_SPLB_NUM_MASTERS-1) <= (others => '0');
                IP2Bus_Mst_Reset <= SPLB_Rst;

end Behavioral;
```