

Quantifying Effective Memory Bandwidth of Platform FPGAs

Andrew G. Schmidt

Submitted to the Department of Electrical Engineering &
Computer Science and the Faculty of the Graduate School
of the University of Kansas in partial fulfillment of
the requirements for the degree of Master's of Science

Thesis Committee:

Dr. Perry Alexander: Chairperson

Dr. Ron Sass

Dr. David Andrews

Date Defended

© 2007 Andrew G. Schmidt

The Thesis Committee for Andrew G. Schmidt certifies
That this is the approved version of the following thesis:

Quantifying Effective Memory Bandwidth of Platform FPGAs

Committee:

Chairperson

Date Approved

Abstract

The Reconfigurable Computing Cluster aims to use Platform FPGAs to determine the feasibility of building a cluster capable of scaling to the PetaFLOPs range. Platform FPGAs were chosen for their ability to host entire systems on a single chip. This thesis investigates several common bus architectures common to Platform FPGAs in order to measure the effective bandwidth between High Performance Computing cores and off-chip memory.

Specifically three memory access patterns are implemented: random, strided, and sequential. Sequential can be further broken down into burst and non-burst mode transaction styles. In addition the effect of multiple cores on, not only the system, but each hardware core is studied. The results show that while some bus architectures are clearly better than others, none approach the theoretical bandwidth of the memory interface. Furthermore, negotiating the bus protocol is a significant source of overhead. So much so that it effectively hides any performance one might gain from trying to access the off-chip SDRAMs using an “intelligent” access pattern. Finally, burst mode (naturally) gives the highest performance, but unless the HPC core knows in advance that several sequential values are needed, non-burst sequential transaction accesses are no faster than strided or random.

Contents

Acceptance Page	i
Abstract	ii
1 Introduction	1
1.1 Motivation	3
1.2 Statement of Problem	4
1.3 Outline	5
1.4 Contributions	6
2 Background	8
2.1 Platform FPGAs and Reconfigurable Computing	8
2.2 IBM CoreConnect	9
2.2.1 IBM's — Processor Local Bus	10
2.2.2 IBM's — On-chip Peripheral Bus	11
2.3 DDR SDRAM	12
2.4 Xilinx Components	15
2.4.1 Xilinx's Virtex-II Pro	15
2.4.2 Xilinx's Intellectual Property Interface	15
2.4.3 Xilinx's DDR SDRAM Memory Controller	16
2.4.4 Xilinx's Environment Development Kit	17
3 Related Works	18
4 Design & Architecture	22
4.1 Memory Access Patterns	23
4.1.1 Sequential Non-Burst Access Pattern	24

4.1.2	Sequential Burst Access Pattern	24
4.1.3	Random Access Pattern	24
4.1.4	Strided Access Pattern	25
4.2	Single Core Design	25
4.2.1	Software Initialization	27
4.2.2	Single core control and data flow	28
4.2.3	Slave finite state machine	30
4.2.4	Master finite state machine	31
4.2.5	Executing the Test	32
4.2.6	Retrieval of Results	33
4.3	Multiple Cores	33
4.3.1	Software Initialization	35
4.3.2	Executing the Test	35
4.3.3	Retrieval of Results	36
5	Results and Analysis	38
5.1	Theoretical Bandwidths	38
5.2	Simulation Results	40
5.3	Single Core Synthesize Results	41
5.3.1	Non-burst transaction results	42
5.3.2	Burst transaction results	44
5.4	Multiple Core Results	45
6	Conclusion	54
6.1	Future Work	56
	References	58

List of Figures

4.1	On-chip organizations for memory and hardware cores	26
4.2	Hardware core signaling and bus connection	29
4.3	Master and slave finite state machines	29
4.4	On-chip organizations for memory and multiple hardware cores . .	34
5.1	Single core simulated vs. synthesized results	41
5.2	Single core effective memory bandwidth results	43
5.3	Multiple cores average bandwidth (non-burst)	48
5.4	Multiple cores average bandwidth (burst)	49
5.5	Multiple cores individual bandwidth (non-burst)	50
5.6	Multiple cores individual bandwidth (burst)	51
5.7	Multiple cores aggregate bandwidth (non-burst)	52
5.8	Multiple cores aggregate bandwidth (burst)	53

Chapter 1

Introduction

The Reconfigurable Computing Cluster (RCC) Project [27] is investigating the use of Platform FPGAs in High Performance Computing (HPC) clusters. Specifically, the RCC Project is exploring the feasibility of building a cost-effective cluster capable of scaling to the PetaFLOPS (10^{15} floating-point operations per second) range.

The benefits of High Performance Computing (HPC) can be seen in a wide range of applications. From science and medicine [23,25,30] to industries as diverse as oil exploration, financial, and entertainment [18], access to cost-effective HPC is becoming a critical part of our national infrastructure.

While current commodity off-the-shelf (cots) machines are providing cost-effective performance it remains to be seen if this technology will be capable of sustaining performance as applications demand petascale performance. Researchers [19] suggest access to cost-effective petascale computing would enable scientists to perform tests which are not possible with today's technology. By increasing the resolution of computer simulations, experiments reveal phenomena that are otherwise unobservable and provide insight into processes that are

otherwise impractical to experiment in the physical world.

For example, Computational Fluid Dynamics scientists want to increase the resolution of their experiments, allowing them to observe phenomenon and make better models. However, simply doubling the resolution increases the computation 16-fold. The increase is due in part to the fact that there is 2^3 more data that needs to be processed and that smaller time-steps are required for mathematical stability.

Among other challenges pertaining to the development of the cluster, one important question is related to the off-chip memory bandwidth of individual nodes. As Underwood and Hemmert [29] have shown, several important scientific applications (such as those using dense matrix computations) are not limited by the computational capability of the FPGA, but rather by the off-chip memory bandwidth. Moreover, it is reasonable to expect that exponential advances in semiconductor technology will continue to improve the rate-of-computation of integrated circuits (including FPGAs). However, without enough memory bandwidth, applications with large data sets will not necessarily solve problems faster. This so called “Memory Wall” was predicted over a decade ago [32] but, thus far, large caches on general-purpose microprocessors have been able to compensate for the growing disparity. Since FPGA designers do not have the luxury of large caches, high-performance designs generally must include custom memory hierarchies and data paths as well as application-specific computational structures. Hence, characterizing the memory bandwidth is crucial to the RCC project. We look to first, evaluate the feasibility of the proposed approach, then second, determining the best on-chip structures to support computational science applications.

1.1 Motivation

While a number of tools and a rich catalog of IP cores exist to assist building on-chip structures to access external memory, many of the key system components — such as buses, bridges, and (off-chip) SDRAM controllers — were intended for use in embedded systems [6, 7]. These components are often used to build the conventional two-bus structures. A processor and primary memory connected by a system bus and a slower, general-purpose I/O or peripheral bus bridged to the system bus which adds secondary storage, network, and other input-output components.

This serves embedded systems well. A peripheral bus has fewer features (making it easy to use), generally allows more cores to be attached, and requires fewer resources. The two-bus structure isolates the high-speed bus from slower transfers. The organization is a good compromise between general communication and bandwidth. Direct connections between cores provide the highest bandwidth, but at the expense of allowing arbitrary cores to communicate. Other interconnection schemes also exist but they are generally more difficult to work with and require more resources.

Platform FPGAs often mimic a two-bus structure due to available IP catalogs and tools. However, it is unclear whether these components are able to sustain the bandwidth needed for HPC computational science designs. Although the performance characteristics of individual components are available, it is not always possible to determine analytically how an arbitrary composition of several components will behave.

Exploiting parallelism in reconfigurable computing has been a motivating reason behind its use in HPC. Unfortunately rather than producing orders of mag-

nitude increases in performance many researchers are left with much less than $10\times$ speedups and in some cases slowdowns! This is partly due to the implicit assumptions made by designers about access to off-chip memory. Currently memory is capable of providing a theoretical bandwidth of up to 3200 MB/s. While it is not common to assume 100% utilization of this bandwidth, designers may estimate obtaining a percentage of this bandwidth. The question then remains, how close are designers to obtaining their estimate? Researchers may simulate their designs to provide a more accurate model. However, as will be discussed in Chapter 5, a large disparity between theoretical and simulation results exist. It may be that any designers using simulation and theoretical bandwidth are grossly overestimating the system's performance.

It is necessary to not only provide simulated results, but to actually synthesis tests, in hardware, to determine from the potential simulated bandwidth what the effective bandwidth actually achieved is. Additionally, as Noseworthy and Leeser [24] have shown, there can be significant variability in memory performance depending on how the components are arranged. Therefore, experimental analysis is necessary.

1.2 Statement of Problem

The fundamental question we aim to answer with this thesis is whether it is feasible to leverage the existing IP cores for High-Performance Computing. These cores were designed to support embedded computing systems by providing several buses and bridges with different performance and operational characteristics. They are flexible and convenient, which allows them to be used in a variety system designs. However, the price we pay for general-purpose is performance. How

much performance we lose will answer the question of feasibility.

This work focuses beyond the simulated and theoretical analysis of individual CoreConnect components, and instead tries to understand the effective performance of all components interacting as a whole. There are several reasons. First, SDRAM having non-uniform access times. Second, off-chip interfacing protocols have variable times. Third, many of the components have ranges of cycles that depend on the actual data (and sequence of data). Fourth, there are many interacting components in memory hierarchy. It is difficult or nearly impossible to accurately model how they will collectively interact.

Hence, our approach is to implement the systems and measure their actual performance. The goal of this work is two-fold. First, we empirically characterize the performance of several CoreConnect on-chip memory subsystem under three common HPC memory access patterns. Second, based on that data, synthesize a set design guidelines for getting the most performance out of the memory subsystem for High-Performance Computing. By addressing these two goals, we can answer the question of feasibility.

1.3 Outline

The rest of the thesis is organized as follows. The specific contributions of this thesis are presented next. The background knowledge and detailed descriptions of components used throughout this thesis are found in Chapter 2. Chapter 3 discusses related works in the subject area. Chapter 4 contains an in-depth description of the design and implementation of the tests within this thesis, beginning with a description of the different access patterns in Section 4.1 and followed by both the single core organization in Section 4.2 and the interconnection of multi-

ple cores in Section 4.3. In Chapter 5 the results and analysis of simulation and synthesis of different access patterns in Section 5.2, single cores in Section 5.3, multiple cores Section 5.4, and varying bus architectures. Finally the conclusion and comments on future work to provide HPC with more effective bandwidth are discussed in Chapter 6.

1.4 Contributions

The main goal of this work is to determine if currently available off-the-shelf components will provide the necessary bandwidth demanded by High Performance Computing. Below is a list of the contributions made by this thesis work.

- Development of a testing methodology for three different access patterns and accurately measures bandwidth:
 - Sequential (burst and non-burst)
 - Strided
 - Random
- Design and sythesis of a set of hardware cores to obtain bandwidth measurements via:
 - software driven testing interface for configurability and reliability
 - slave finite state machine controlling individual tests and collecting timing measurements
 - master finite state machine issuing requests to off-chip memory through Xilinx interface protocol

- Design and synthesis of four different on-chip memory interfacing organizations using commodity off-the-shelf IP cores to thoroughly test all combinations of configurations of memory controllers and test cores
- Measuring single core effective bandwidth to off-chip memory
- Measuring multiple cores effective bandwidth to off-chip memory and the impact multiple cores has not only on the individual cores, but the system and other cores as well
- Illustrating the difference between theoretical, simulation, and synthesized bandwidth
- Providing evidence of the lack of off-chip memory bandwidth current components are capable of offering High Performance Computing

Chapter 2

Background

This thesis relies on a firm understanding of components and tools commonly used within reconfigurable computing. For those familiar with these components it is advised to proceed onto Chapter 3; otherwise, it is recommended to review this material prior to continuing on with the remainder of the thesis.

2.1 Platform FPGAs and Reconfigurable Computing

High performance computing uses multiple processing elements connected together via a network in order to create fast supercomputers capable of meeting the performance demands of industry, research, and education. While many approaches have been taken to create HPC systems, our approach investigates the feasibility of using platform FPGAs rather than conventional processors.

Platform FPGAs are a system-on-chip (SoC) which is entirely contained within the FPGA [22]. The difference of between an FPGA and an application specific integrated circuit (ASIC) is the FPGA is capable being reprogrammed, changing the SoC organization or adding and removing components. An FPGA is typically

comprised of hard IP which is embedded into the FPGA fabric and soft IP which can be added or removed from the design. The ability to create these new soft IP cores and later modify or remove them give the designer greater flexibility over ASICs which require significant design time, are more costly, and result in slower development time due to fabrication. Compton and Hauck further discuss reconfigurable computing in a survey paper [4] aimed at introducing many concepts of reconfigurable computing and its similarities and differences to commodity computing components.

Platform FPGAs are now capable of hosting entire systems, from embedded processors, buses, bridges, to networking, on-chip memory and off-chip memory controllers. In addition user specific intellectual property (IP) hardware cores can be designed and connected into the system. As such, it is possible to use a Platform FPGA as a node in a HPC message-passing [11] cluster [27]. The following sections discuss in more detail the components which typically compose a SoC Platform FPGA implementations.

2.2 IBM CoreConnect

A bus organization commonly is comprised of a system and peripheral bus. IBM's CoreConnect bus architecture [16] is designed to connect embedded processors, buses, bridges, on-chip memory, off-chip and memory controllers. While these cores were designed by IBM, Xilinx provides specific implementations suitable for use with their FPGAs. Discussed below are the Processor Local Bus and On-chip Peripheral Bus. These two buses are used to connect multiple cores together; however, each have their own characteristics and functionality.

2.2.1 IBM's — Processor Local Bus

IBM's Processor Local Bus (PLB) [7] operates on a synchronized 100 MHz clock and supports 16, 32, and 64-bit data transfers. This work will focus on utilizing the 64-bit data buses and 32-bit address bus. Embedded processors, PowerPC405 in the case of the Virtex-II Pro, along with other cores with high performance demands traditionally connect to the PLB. Off-chip memory is also commonly interfaced through an on-chip memory controller connected to the PLB. When components on the PLB need access to off-chip memory a request is made through the memory controller.

The PLB supports up to sixteen masters and any number of slaves to be connected at one time. Each master is attached through separate address, write data, and read data buses. Each slave is attached through a shared, but decoupled address, read data, and write data buses. This separation of buses allows transfers to be pipelined, meaning separate master requests can be overlapped with an ongoing request in the opposite direction. For example, if one core performs a read request a second core could perform a write request and due to the separate buses the transactions can occur simultaneously.

In addition, the PLB supports primary and secondary buffering of requests. If two cores issues requests in the same direction the first core's request will be directly followed by the second request rather than the second core's request being blocked.

The PLB also offers bus parking where a requesting core can retain ownership of the bus as long as no other core's request access to the bus. The result is the core no longer needs to communicate with the bus arbiter for ownership, reducing transaction times and increasing performance.

An alternative method to retaining ownership of the bus is for the core to issue a request with a bus lock signal asserted. The PLB arbiter will wait for both the write data bus and read data bus to be available prior to granting the PLB to a core, but then the core has ownership of the bus until the bus is unlocked by the core.

Xilinx's implementation of IBM's PLB contains all of the necessary signals in order to connect none, one or both of the embedded processors. While off-chip memory is typically connected to the PLB via an on-chip memory controller, it is possible to connect the memory through the OPB and instantiate a bridge to allow communication between both buses.

2.2.2 IBM's — On-chip Peripheral Bus

The tests also utilize an instantiation of IBM's On-chip Peripheral Bus [6]. The OPB operates on a synchronous 100 MHz clock and supports 16 and 32 bit data transfers. The OPB is used in this thesis for completeness to determine the effect of connecting hardware cores to a secondary bus while accessing memory across a bridge to a primary bus. The OPB is not designed to be connected to embedded processors; however, it is possible to interface off-chip memory through a on-chip memory controller connected to the OPB. Unlike the PLB, the OPB does not offer separate data and address buses nor does it offer address pipelining. The OPB does provide bus parking which allows a single master direct access to the bus without any arbitration under the condition that no other requests are pending. As with the PLB, the OPB supports up to sixteen masters and any number of slaves to be connected at one time.

2.3 DDR SDRAM

These tests utilize Double Data Rate Synchronous Dynamic Random Access Memory, DDR SDRAM, as off-chip memory (a limitation of the Xilinx MI-310 board). In Chapter 4 and Chapter 5 references are made back to this section to explain in more detail the characteristics of off-chip memory in the designs.

The Joint Electron Device Engineering Council (JEDEC) Solid State Technology Association released the Double Data Rate (DDR) SDRAM Specification [1]. The specification covers all aspects of 64 Mb through 1 GB DDR SDRAM memories. This section will cover the necessary knowledge to understand the functionality DDR SDRAM has in this thesis.

Patterson and Hennessy [14,15] go into detail describing Random Access Memory (RAM) and its various characteristics and functionality. Modern memory provides a large collection of addressable memory cells. Dynamic Random Access Memory (DRAM) stores data in capacitors and must be refreshed periodically (64 ms) to retain the data. A single transistor is used to then access the stored data within the capacitor. This is unlike Static Random Access Memory (SRAM) which uses multiple transistors (typically between four and six) to store data and does not require a refresh. The result is SRAM is significantly faster than DRAM while DRAM is less expensive and capable of economically obtaining higher densities than SRAM.

DRAM works by using a pass transistor to allow either reading or writing of data from the capacitor. This transistor, operating much like a switch, connects the capacitor to the memory cell's bit line and during a write the capacitor will either be charged or dispersed. During a read the bit line is supplied a partial voltage, then when reading the value from the capacitor a sense amplifier detects

the change in voltage on the bit line.

When addressing DRAM a two-level decoder, row and column, is used. Row Address Strobe (RAS) and Column Address Strobe (CAS) are used to simplify the DRAM design by reducing the number of addressing pins. The strobe signals whether the address supplied to DRAM is either for a row or column. First the row is activated (opened) and stores the data from the columns in latches. The column access selects the latched data from the corresponding column. As stated above, DRAM requires a refresh signal to retain data. A refresh command is used to read data from a row into the column latches, then writing the data back into the columns. This dramatically effects performance by a reduction of as much as five to ten times as compared to SRAM.

Synchronous Dynamic Random Access Memory (SDRAM) evolved from DRAM. While still based on the underlying physical and electrical characteristics of DRAM one significant difference is SDRAM is synchronized to the bus's clock frequency. DRAM operates asynchronously, allowing memory to operate at one frequency and the bus it connects to to operate on a separate frequency.

A significant advantage of SDRAM is its ability to provide all the data in a particular row during a burst transaction. Rather than issuing single column addresses, a row address is used to open the row from which data will be read. This increases performance under situations where data is typically sequentially accessed.

Double Data Rate (DDR) SDRAM [1] differs from Single Data Rate (SDR) with the ability to transfer two data words per clock cycle, one on the rising edge and the second on the falling edge. Read and writes are designed for burst transactions to improve efficiency. A request begins with an *ACTIVE* command

followed by either a *READ* or *WRITE* command. A bank within the memory is selected based on the supplied address, typically A0 and A1, followed by the row select, typically A0-A13. Requests to different rows require the row to be closed before opening the new row.

With the progression of technology newer DDR standards have been released, such as DDR2. While the current prototype node in the cluster utilizes Xilin's ML310 development boards only DDR memory can be used. Future releases will be able to utilize DDR2. DDR2 consists of SDRAM just as DDR; however, in DDR2 the bus is clocked at twice the speed of the memory which results in four words transferred per memory clock cycle. This does introduce additional latency as the bus is capable of higher speeds while the memory itself is only operating at half the frequency. In DDR the latency is typically between two and three clock cycles whereas DDR2 is commonly between three and nine clock cycles.

Dual Inline Memory Modules (DIMM) contain multiple SDRAM modules in order to provide higher densities. However, this also adds a higher capacitive load on address and control signals as compared to lower density DIMMs. One approach is to use buffers on the DIMM to increase signal strength and reduce the system load. This memory is commonly known as registered memory compared to memory without this technique known as unbuffered memory. Registered memory is capable of obtaining higher densities at the cost of a small delay in the electrical signal.

Many applications require the use of external memory storage since on-chip memory, especially with FPGAs, can be very limiting. While the design of SDRAM and its operation characteristics may seem too detailed, it is necessary to understand its complexities in order to design better memory interconnects in

the future.

2.4 Xilinx Components

This thesis relies on designs created through the use of multiple Xilinx [33] components. These components are discussed below in more detail.

2.4.1 Xilinx's Virtex-II Pro

The Virtex-II Pro XC2VP30 [40] is an FPGA created by Xilinx which contains 30,816 logic cells, 13,696 slices, 136 18 Kb Block RAM and 2 embedded IBM PowerPC 405 processor blocks. In these designs the ML-310 development board contains a single Virtex-II Pro FPGA and is configured with Xilinx provided software. The IBM PowerPC 405 is a RISC based processor with a maximum frequency of 300 MHz and connects through IBM's CoreConnect bus architecture to off-chip memory and other cores in this work. Currently the logic limitation of the Virtex-II Pro is a limiting factor to the number of test cores we are able to connect in our tests. While the upper limit of sixteen cores is specified by the CoreConnect documentation, we are currently only capable of connecting a maximum of eight cores.

2.4.2 Xilinx's Intellectual Property Interface

Xilinx provides two interfaces for connecting hardware cores to the bus, depending on the bus, OPB or PLB [35, 38]. In principle a designer could create a hardware core intended to connect to the PLB and then move the hardware core to the OPB. This process works through the use of the Intellectual Property Interface (IPIF) which connects to the IP's user logic via an Intellectual Property

Interconnect (IPIIC).

The user logic issues requests through the IPIIC which propagates the signal through the IPIIF. Two different IPIIFs exist, both designed to signal their respective bus, OPB or PLB. The advantage of such a system is portability with a sacrifice of resources and performance. In high performance computing this may not be a sacrifice designers are willing to make. The result is constructing bus specific cores and if necessary modifying the design if necessary to move the core to the opposite bus.

2.4.3 Xilinx's DDR SDRAM Memory Controller

Xilinx provides two memory controllers for connecting the FPGA to the pins of the DDR SDRAM. The memory controller is an on-chip hardware core which either connects to the OPB [41] or PLB [37]. The controllers consist of a corresponding bus interface (IPIIF) to communicate with the bus and initialization, data, and command state machines for initialization, data transfers, clocking and control of off-chip memory. Through the use of separate state machines it is possible for both data and control transfers to occur in parallel in order to provide more efficient utilization of the memory.

Addresses and command signals are issued by the command state machine and signals the data state machine to send or receive data. The command state machine is responsible for issuing DDR SDRAM specific commands to activate rows, open rows and columns, read or write data and issue refresh commands. The timing constraints are specified by the off-chip memory and must be strictly adhered to in order to achieve optimal performance.

2.4.4 Xilinx's Environment Development Kit

Xilinx provides a tool through the EDK called Base System Builder used to create the base systems which this thesis rely on. In this base system certain components, such as embedded processors, buses, bridges, and memory controllers are connected. After the creation of the base system the user can connect hardware cores to the system.

Using Xilinx's Base System Builder wizard from within the EDK the Xilinx Virtex-II Pro ML310 Evaluation Platform (revision D) board is selected. In the design a single PowerPC is used to control the program and load test specific data into the hardware core. As a result the operating frequency is left at 100 MHz. The Virtex-II Pro provides varying bus frequencies of 25 MHz, 33.33 MHz, 50 MHz, and 100 MHz. To maximize the theoretical bandwidth, the bus operating frequency is set to 100 MHz.

Next the external memory peripheral is selected. As will be explained in Chapter 4 the memory is either connected via the PLB or the OPB. Therefore two base systems are created to support both configurations. The DDR used is W4F232726HA-5Q is a 32MX72 industry standard 184-pin low profile registered DDR400B-333 SDRAM DIMM [17]. Upon completion of the Base System Builder wizard, additional hardware cores can be connected to the system.

To minimize the number of base systems and simplify future testing bridges were used to allow communication between the PLB and OPB. The wizard adds the PLB to OPB bridge [39] and allows cores on the PLB to issue requests to the OPB. Additionally, an OPB to PLB bridge [36] allows cores on the OPB to issue requests to the PLB.

Chapter 3

Related Works

Many research projects are investigating memory subsystems and in particular, HPC memory bandwidth. With the advent of multi-core chips a number of industrial efforts are looking at interconnection schemes such as dual-channel [8], fully-buffered [13], and direct access [34].

The Merrimac project at Stanford is investigating computer architectures and novel languages in attempt to use bandwidth more efficiently [9, 12] by organizing the computation into streams in order to exploit locality. Each node within the system contains sixteen DRAMs totaling two GBytes of memory with simulation estimations expecting the ability to sustain half of the theoretical bandwidth on multiple scientific applications.

Numerous memory buffering [26], re-timing [5], buffers for sliding window computations, and specialized caches have also been extensively covered in the FPGA literature. However, most of this work has been in the context of the FPGA being a co-processor in the system — not hosting the entire system. This is relevant because, as a co-processor, the FPGA usually has exclusive access to local memory and does not require any of the general-purpose structures discussed here.

Work more closely related to this effort done by Donchev et al. [10] has focused on the improvement of efficiency of communications between the PowerPC and off-chip memory. The work takes advantage of the principle functionality of the On-Chip Memory (OCM) bus which gives the PowerPC a low latency, high performance connection with on-chip memory (BRAM) and combines it with the Processor Local Bus (PLB). Named OCM2DDR, this approach is novel in that it allows the PowerPC dedicated access to off-chip memory under the on-chip memory communication mechanism. The performance increases in both reads and writes from the PowerPC by 2.45 and 4.25 respectively.

While this approach does provide the PowerPC faster access to memory, it does so at the cost of other core's access to off-chip memory. The memory controller is traditionally connected to the PLB so all cores needing access to off-chip memory have access. However, by directly connecting the OCM interface from the PowerPC through the PLB's memory controller those additional cores can no longer request access to off-chip memory. This may benefit the original work, but High Performance Computing looks to the reconfigurable fabric for a majority of the processing. Under this requirement the OCM2DDR interface will not be sufficient.

Noseworthy and Lesser [24] use a Software Defined Radio (SDR) as a vehicle for investigating effective communications between the PowerPC and the surrounding FPGA fabric. The work investigates On-Chip Memory (OCM), Processor Local Bus (PLB) and On-Chip Peripheral Bus (OPB) interfaces between the PowerPC and on-chip memory. In addition to different buses, they studied the effect the cache has on performance. While OCM provides a fixed latency to to Block RAM which is advantageous for applications that must guarantee a specific rate

of communication, the amount of memory is limited in size. The PLB provides efficient on-chip communication, but is a shared resource which results in possible bus contention and degradation of performance.

The results show a 60% increase in performance for the SDR implemented when using OCM. However, caching works best when data exhibits temporal locality characteristics. When applications must process streaming data, those commonly associated with High Performance Computing, they found that using a cache can actually hurt performance. While the cache results do not directly related to this thesis work, it is an important result to remember if designers consider implementing a caching mechanism in the reconfigurable logic and are targeting HPC applications with high levels of streaming data.

This thesis is studying communications similar to those studied by this paper, the difference is the focus on communications from reconfigurable logic to off-chip memory rather than the embedded processor. While a single core may be designed to effectively communication with off-chip memory, HPC requires multiple cores have access to off-chip memory.

Lee and Bergmann [21] also investigate the interface between IP cores and buses in order to improve performance and automate the connection between a variety of interface architectures. Adding a bus interface, for example Xilinx's IP Interface, adds additional overhead at the sake of portability. However, there is still a limitation on the core's connect-ability with other buses. Creating additional interfaces to meet each bus may result in performance loss. While this research does not directly relate to this thesis, it is of interest as future design's portability will be of great concern.

Anderson et al. [2, 3], in their Hyridthreads Computational Model, use on-

chip dual ported memory (BRAM) to create global distributed local memory within each Hthread hardware thread. One port on the BRAM gives a thread direct access to its local memory without contention for the bus. The second port allows other cores access to the same memory through the standard bus protocols. The result is “cache-like” memory access performance; however, this memory is not cache as commonly found in commodity processors. The requirement for populating the local memory falls on the Hthread core rather than additional logic. It is an interesting approach in that it gives hardware threads access to all of the other thread’s local memory. Hardware threads needing to access memory or provide another thread memory can do so without the penalty of accessing off-chip memory.

In contrast to the above approaches, other vendors such as Lattice Semiconductor, have embedded specialized hardware in low-cost FPGAs to manage the memory interface [28]. This approach eliminates design concerns about the need to map, place, route and meet timing requirements on a critical interface. However, it requires physical space on the FPGA as it is a hardcore rather than a more traditional softcore memory controller. Most applications find a need for off-chip memory which makes the decision sensible; however, it limits future designs which may have greater demands for the FPGA fabric and less of a need for off-chip memory access. In this case having an embedded memory controller would be a waste of resources.

Chapter 4

Design & Architecture

The Reconfigurable Computing Cluster is designed with modularity in mind to allow for many different applications to be run and tested. However, the prototype uses a commercial development board (Xilinx ML310 board) to constrain costs. The physical pin connection between off-chip memory and the FPGA are limited on the board's layout. As a result the requirements of the cluster will require different on-chip interconnects within the FPGA to connect to the off-chip memory pins. In order to test these different on-chip organizations to determine their suitability for the cluster this thesis will consider the following:

- different access patterns from a core to off-chip memory
- varying on-chip interconnection organizations
- varying number of cores within the system access to off-chip memory

The experiments were tested on one node of the Reconfigurable Computing Cluster under construction at the University of North Carolina, Charlotte. The prototype node is a Xilinx ML-310 board with a Virtex-II Pro (XC2VP30) and Wintec 184-pin DDR400 PC3200 SDRAM DIMM.

The on-chip memory structures were developed using Xilinx Embedded Development Kit (EDK) 8.2sp2 and Integrated Development Environment (ISE) 8.2sp3. EDK contains a subset of the IBM CoreConnect IP catalog which includes a Processor Local Bus (PLB), an On-Chip Peripheral Bus (OPB), as well as bridges. The off-chip DDR memory controllers used in these experiments were *plb_dds 2.00.a* and *opb_dds 2.00.c* along with bus bridges *plb2opb 1.01.a* and *opb2plb 1.00.c*. Several systems were synthesized using different bus structures and special-purpose cores designed to generate common HPC memory access patterns.

Using Xilinx’s EDK a base system was designed for each of the primary configurations. For a more complete description refer to Subsection 2.4.4. The base system includes the PowerPC processor, PLB, OPB, and DDR memory controller. These off-the-shelf components were selected based off of their current availability and price.

4.1 Memory Access Patterns

The following are different access patterns commonly associated with external memory and High Performance Computing [20, 31]. From within the Test Core each test uses the same mechanism to retrieve an address to read from external memory. During the initialization of the test while software is supplying the Test Core’s BRAMs with addresses, each access pattern’s address range is filled in a specific sequence. Below is a quick description of the access patterns used within these experiments: sequential non-burst, sequential burst, random and strided access.

4.1.1 Sequential Non-Burst Access Pattern

Sequential access consists of both burst and non-burst transactions to move adjacent words to or from memory. During non-burst transactions a single request is issued per word retrieved. The test core accesses a single address and waits until the data returns. This is the simplest access pattern to perform. Upon the return of the data, the next request issued is the next adjacent address from off-chip memory. An entire bus transaction occurs for each single request.

4.1.2 Sequential Burst Access Pattern

In burst transactions more than one datum is retrieved per single request. Although similar in principle to a single sequential access, the difference is that unlike a single access the burst access specifies a number of sequential addresses (transfer length) to receive data from. Up to 128 bytes of sequential data are retrieved starting at the requested address. While the first word's access is equivalent to non-burst transactions, subsequent words arrive one clock cycle apart through the completion of the transaction. As a result, the average time per access with burst transactions is drastically reduced.

4.1.3 Random Access Pattern

Random access is similar to single access in that each new request is issued to, and responded by, the memory controller prior to the next request. The addresses are randomly generated during the test's initialization and supplied to the test core prior to execution. It is not possible to perform burst transactions during random access due to the burst mode's sequential access restrictions. Random access potentially offers the worst performance, as SDRAM needs to close the

current row and open a new row (a time consuming process which adds to the latency).

4.1.4 Strided Access Pattern

Strided access retrieve data a uniform distance apart in memory. While the stride length may vary depending on the application, the concept can be seen in a loop iteration or a jump (goto) statement which may cause the program to access memory a uniform distance from the previous access. When the stride length is small it is possible all data may be contained within the same row of SDRAM, potentially reducing latency to subsequent requests. As the stride length increases the performance will more likely match that of random access.

4.2 Single Core Design

Under a two-bus architecture, a number of different bus structures are possible. The memory controller, an IP core that used to access off-chip DDR SDRAMs, translates bus requests into SDRAM memory transactions. Two DDR memory controllers exist: one for the peripheral (OPB) bus and one for the system (PLB) bus. Likewise, the HPC core can be placed on either bus which leads to the four combinations illustrated in Figure 4.1.

The goal of the different single core organizations is to test the performance of a single HPC core in isolation. Single core tests are performed without any additional contention for the bus or off-chip memory; this provides accurate effective bandwidth results for each configuration. These results will be used as a baseline to then understand the impact of adding multiple cores into the system. Both OPB and PLB tests are included for completeness. The testing system consists

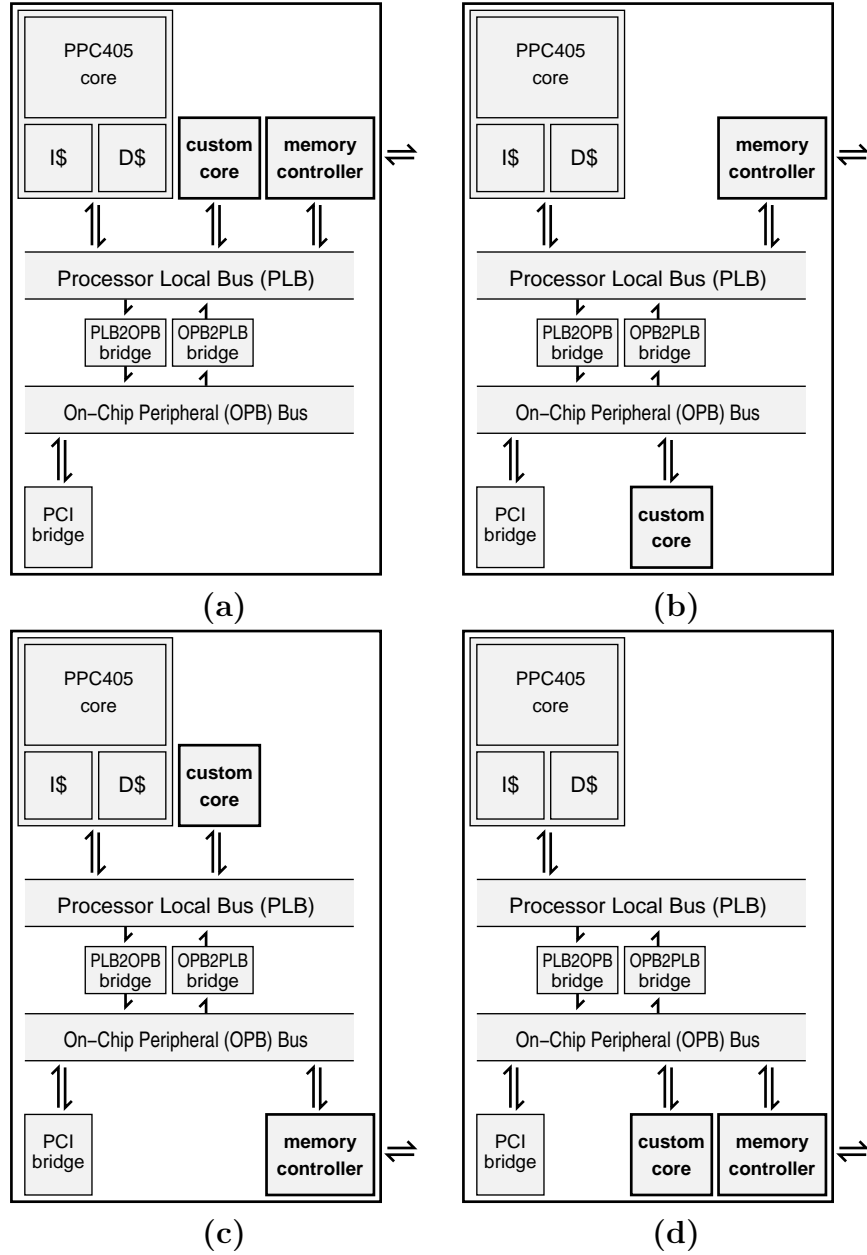


Figure 4.1. On-Chip memory organization and computational core options: (a) custom core and memory controller on PLB; (b) custom core on OPB, memory controller on PLB; (c) custom core on PLB, memory controller on OPB; and (d) custom core and memory controller on OPB

of three parts:

1. initialization of the core (in software)
2. execution of the test by the core (in hardware)
3. retrieval of the results from the core (in software)

4.2.1 Software Initialization

Before the core is able to issue read requests to memory it must first be initialized for the particular test. During the initialization process, code running on the PowerPC provides all of the necessary data the core needs. The initialization consists of providing:

1. addresses the core will read from off-chip memory
2. core specific addresses to be used during read requests
3. transaction information (burst, non-burst and transfer length)

As stated earlier there are three access patterns: sequential, strided, and random. To simplify the tests and provide greater flexibility in testing the addresses are not generated by the core. Instead the addresses are supplied to the core during the aforementioned initialization sequence. During the initialization the core is not performing any requests and the timer is not counting; therefore, the setup time does not factor into the performance and more importantly no additional testing overhead is introduced.

The different addresses for each corresponding access pattern are generated through C code executed on the PowerPC. For single sequential addresses the contiguous addresses are supplied to the core. Strided access patterns require a

specified stride length. Multiple tests are required for this access pattern due to the multitude of different stride lengths. As addressed in Chapter 2, SDRAM theoretically provides the lowest latency to addresses within the same row. Therefore smaller strides within the same row should provide approximately the same latency as sequential access. For random access addresses are randomly generated in software and supplied to the core. The advantage of such an approach is less time spent designing a core which can generate its own random addresses. Additionally, if there is a need to test alternative access patterns the hardware core will not require modification. Instead the software can generate the addresses and supply the hardware core with the new addresses.

4.2.2 Single core control and data flow

After software provides the addresses (and access patterns) to the core additional signals are issued to specify the core's master request signals. The core consists of two finite state machines, shown in Figure 4.3; the slave state machine responds to requests made by the bus (in these tests it responds to the PowerPC and memory controller's requests) and the master issues the requests to the bus (issues read requests to memory controller). Figure 4.2 provides a simple illustration of the hardware core and its master and slave components.

The test core consists of both *User Logic* and *IPIF* components. The user logic includes the slave and master finite state machines which perform all of the user defined functionality. The slave and master communicate with the bus through a Xilinx specific Intellectual Property Interface (IPIF). The benefit of using the IPIF is to first reduce the number of signals the user logic is responsible for and second allow the user logic to connect to either the PLB or OPB by using the the

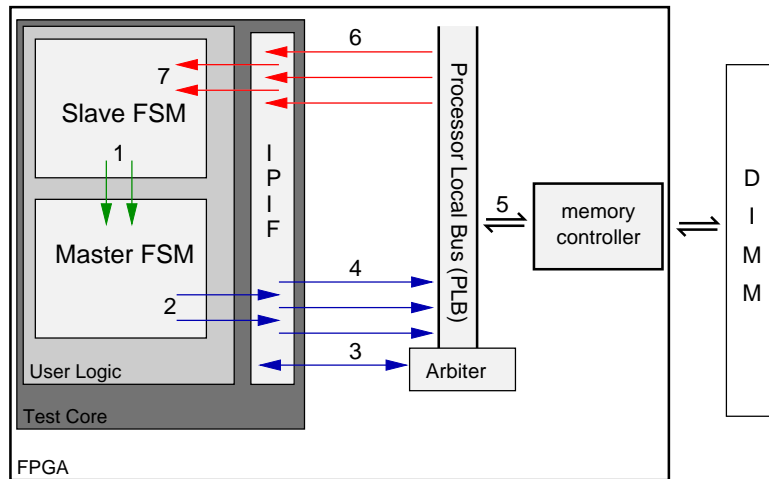


Figure 4.2. Hardware core signaling and bus connection

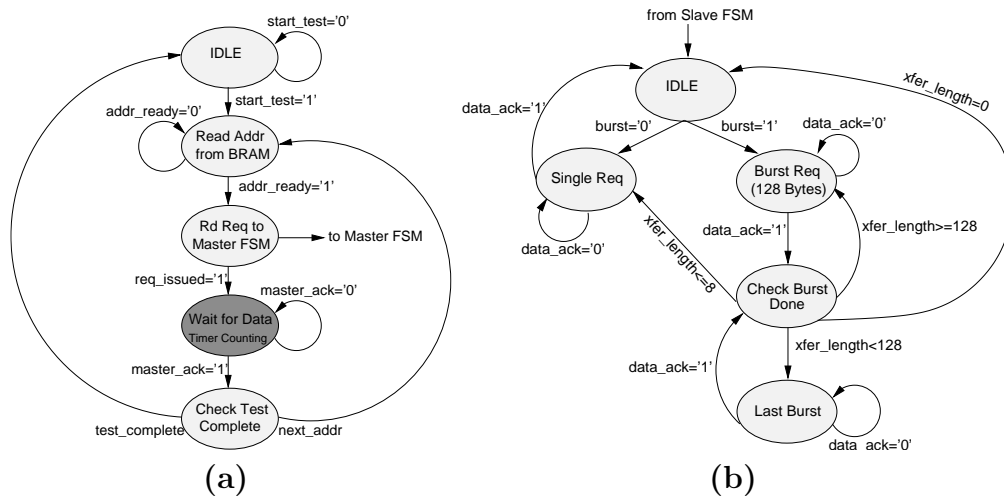


Figure 4.3. (a) slave and (b) master finite state machines

appropriate interface since two different interfaces have been developed by Xilinx, one for the PLB and a second for the OPB. A request is instantiated by the slave finite state machine and terminated once the slave finite state machine receives the requested response. Figure 4.2 shows the general flow of control signals and data throughout a request. Each step is enumerated which corresponds with:

1. slave signals master to perform read across the bus

2. master issues request through the IP Interconnect
3. the IPIF propagates master's request to the bus beginning with the requesting bus access through the arbiter
4. when given access to the bus, the read request is issued across to the PLB along with control signals to complete the transaction
5. the PLB issues a read request to the on-chip memory controller, which in turn issues the read request to off-chip memory
6. the memory controller returns the requested data to the bus to be returned to the requesting hardware core's IPIF
7. the IPIF propagates the data through the IP Interconnect to the slave which acknowledges the request

4.2.3 Slave finite state machine

At this point the data has arrived at the slave and the data is stored into the data BRAMs for later analysis which is discussed below. Figure 4.3(a) depicts the finite state machine for the slave component of the hardware core. The *idle* state is the default state where the slave can either be issued its initialization data or a start signal to begin the test. After receiving a start signal the slave reads a single address from the pre-loaded address BRAM. The address along with master specific control signals are issued to the master finite state machine. The slave then waits until it receives a write request from the bus indicating the data is available and ready to be captured by the slave. The slave stores the data into a separate BRAM to be used at the completion of the test for analysis of

the functionality of the test. The slave then determines whether or not it has performed the last request in the test. In the event more requests are needed to be made, the slave increments internal status registers and counters and reads a new address from BRAM. The entire sequence is performed again. When the last request is issued and the corresponding data stored, the slave returns to the *idle* state signifying the test's completion at which point the slave remains until the tests results are retrieved by the PowerPC and the external analysis is performed.

4.2.4 Master finite state machine

The slave issues a request to the master finite state machine which is shown in Figure 4.3(b). The master waits in an *idle* state until it receives a start signal from slave. Additionally, the slave provides the master with the request type (read or write), transaction type (burst or non-burst), transfer length (for burst mode only), and address signals. Two addresses are given to the master, the address to read from off-chip memory and the address of the slave's register to write the data to. The slave waits for this register to be written to which is describe above.

In the event of a single transaction only one address is read from off-chip memory and the master waits until the bus issues the data to the slave before returning to the *idle* state to wait to preform another request. During a burst request it is possible to specify the transfer length up to sixteen 64-bit words to be read for the PLB. These tests focus on the maximum number of transfers for both the PLB and OPB; however, the master keeps track of transfer length as the request is made to determine if the burst is complete or if additional requests are necessary. Again, just as with the single request, when the request completes and all the data is written to the slave the master returns to the *idle* state.

Internally the slave controls the master by telling the master what requests to make and then waiting for the request to complete (the data to arrive). The master must be supplied: request type, read address, write address, burst or non-burst transaction type, and transfer length signals. These signals are set before the test is run through software, which again provides the core added flexibility and does not require the entire system to be re-synthesized, a very time consuming process, between tests.

4.2.5 Executing the Test

Now that the core has been initialized and all of the required signals are set, the core is issued a start signal. All of the addresses the core will read from SDRAM are stored within the core in local BRAMs. The core reads the first element in the BRAM which corresponds with the first address the core will read from SDRAM. During non-burst transactions the slave waits for the requested data to arrive at a pre-defined address which is written by the off-chip memory controller. When the data arrives it is stored in a second BRAM to be accessed after the completion of the test. After the arrival of each address the next request is issued with the subsequent address.

Burst transactions issue a request and wait for not only the first word to return, but for the completion of the burst transfer. To specify burst transaction, part of the software initialization must give a transfer length. Currently the CoreConnect documentation specifies the burst transfer length to be 128 bytes on the PLB and 64 bytes on the PLB.

The metric considered during these tests is the number of clock cycles it takes a core to issue a request until the slave receives the data. The timing is strictly

controlled by an additional process which increments a counter, based on the 100 MHz clock of the bus, whenever the core is waiting for data. In Figure 4.3(a) the darkened core labeled *wait for data - timer counting* is the state in which the timer is incremented. By only counting during this state the most accurate timing results possible are reported. Moreover, any time spent by the test to retrieve addresses or store data is not included in the timing mechanism as it is considered timing overhead.

When the core performs multiple requests the timer must start and stop to reduce any additional overhead associated with the testing mechanism and provide accurate timing results. During burst transactions a single request is treated the same as non-burst transactions; however, the timer does not stop when the first word is received. Instead the timer is stopped at the completion of the burst transaction.

4.2.6 Retrieval of Results

When the core has performed all requests specified during the initialization stage the core turns to an idle state and the results are retrieved by the PowerPC. These results include the timer counter, transaction counter, error counter, and data read from off-chip memory. The data is used to verify the core correctly retrieved data from off-chip memory.

4.3 Multiple Cores

Although there may be situations which require only a single core accessing SDRAM, more commonly in High Performance Computing multiple cores will need access to off-chip memory. Based on the results from the single core test (see

Chapter 5) only one access pattern — sequential (both non-burst and burst) — was deemed necessary for testing. Moreover, resource limitations dictated that a simple design be used to increase the number of parallel cores in the tests.

The new test core maintains the same functionality as the original core; however, it only is capable of performing sequential non-burst and burst transactions due to the removal of the internal BRAMs previously used to store random and strided access patterns. The new slave state machine is a direct result of the simplification of performing only sequential read requests. The tests are still measured in the same manor and for completeness will be compared with the original core’s sequential results. As a result, eight cores are now able to be connected to either the OPB and PLB as shown in Figure 4.4. The following description will cover how the tests were built and implemented.

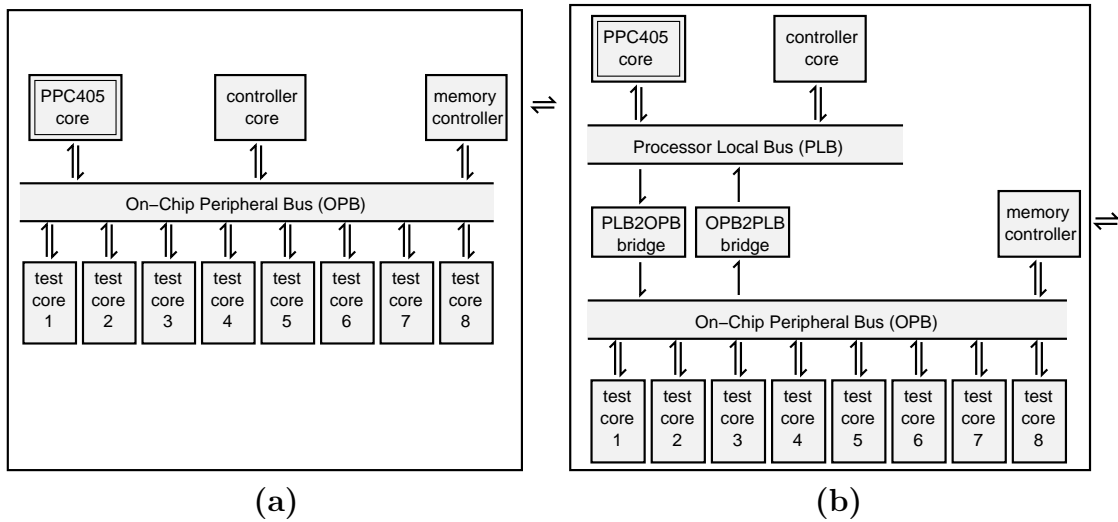


Figure 4.4. Four multiple core organizations: (a) custom cores on PLB and memory controller on PLB; and (b) custom cores on OPB and memory controller on OPB

4.3.1 Software Initialization

Before the core is able to issue read requests to memory it must first be initialized for the particular test. During the initialization process code running on the PowerPC provides all of the necessary data the core needs.

4.3.2 Executing the Test

The test is designed to allow multiple cores to connect to either the PLB or the OPB and perform read requests. With multiple cores a new mechanism is needed to signal each of the cores, telling them to start their respective test. This *Controller Core* is connected to the PLB and waits for the PowerPC to tell it which cores should be included in any particular test (as part of the entire test's initialization process). The controller core outputs a start signal directly, without requiring any bus transactions, which results in no additional testing overhead.

To test the functionality of the test controller a single core is first added to the base system and started via the test controller. As expected, the results matched those of the single test core previously described. A second core is added to the base system. The test controller is now tested by starting only the first core and then both cores. As expected, the test controller starting only the first core resulted in the same results as before. This procedure is repeated until eight cores, the maximum number of cores we were able to fit on the FPGA due to the FPGA's resource limitations, are added to the base system.

A single core is first connected to the bus along with the memory controller. The single core performs reads requests to off-chip memory in: 1, 4, 16, 64, 256, 1024, 4096, 16384, and 65536 sequential addresses. As multiple cores are added this large number of reads will provide for a more accurate understand of how

cores interfere with one and other as they vie for the bus.

Each Test Core waits in an idle state until it receives the start signal from the Controller Core. Since each core in the test is setup to listen to its own unique signal on those cores activated by the Controller Core will participate in the test. A read request is performed identically to the original test core. After each read the next contiguous address from SDRAM is read.

Logically a second core is then instantiated on the bus. again, the start signal is issued from the Controller core and both core 1 and core 2 begin to request access to the bus and perform their individual read request.

Once core should receive its requested data from SDRAM before the second core receives its data. The second core will be required to wait, potentially for the entire time it takes for the first core to receive its data, before it can issue its read request and receive its first datum. The first core will issue its next read request immediately after receiving its first datum. This process will continue through the number of read requests for the specific test.

The tests continue by adding additional cores to the bus until the resources (slices) are depleted. Using an ML310 with a VirtexII-Pro the maximum number of cores added is 8. This is not a limitation on either the PLB or OPB rather it is a limitation on the size of the Test Cores and amount of available resources on the FPGA. In future work it with newer FPGAs it is conceivable to perform the same tests with even more cores.

4.3.3 Retrieval of Results

After the test completes each core's finite state machine will return to its idle state and the results are read via the PowerPC. At this point the test has

finished and no interference is added to the rest of the test. These results primarily include the timer counter, read counter, and error counter. The final datum is read back from each core and compared to the data in off-chip memory to verify the core performed the correct number of reads and received the correct data from memory.

Chapter 5

Results and Analysis

Armed with the IP cores and experimental infrastructure described in the previous chapter, we are now ready to investigate the key question of this thesis. First, we will look at the theoretical bandwidth and the results of using simulation to provide a basis for the measured results. (This is especially useful since these numbers are considered representative of off-chip memory performance.) Following the theoretical and simulations, we measure and discuss the results from single core performance tests and then multiple core performance tests.

5.1 Theoretical Bandwidths

First, theoretical DDR SDRAM bandwidth suggests the greatest bandwidth achievable to directly access off-chip memory. Commodity memory offers theoretical peak bandwidth in the range of 1600 MB/s to 3200 MB/s and is based on direct access to the DRAM itself without any buses, memory controllers or component to component communications. With the addition of dual-channels [8] the bandwidth could be doubled; however, this is the peak *theoretical* bandwidth

current DDR SDRAM is capable of achieving.

Next, when connecting DDR SDRAM to a bus for global access, the theoretical bus bandwidth is commonly a more realistic measure of memory bandwidth. In calculating the theoretical bandwidth of the PLB and OPB (800 MB/s and 400 MB/s respectively [6, 7]) the off-chip bandwidth limitation becomes apparent. Clearly when connecting multiple cores, which all need to access memory, a bus interconnect is also necessary to provide a convenient interface (Xilinx's IPIF). The result is the addition of bus arbitration necessary to determine which requesting component will be granted access to the shared resource. This increases the amount of time to access memory. For these tests which use a bus architecture the bandwidth is limited from 3200 MB/s to 800 MB/s (400 MB/s for OPB).

Finally, the simulated effective bandwidth is commonly used to determine a hopefully more realistic memory bandwidth. With the inclusion of the bus it is no longer reasonable to expect peak (off-chip memory) bandwidth. When designing systems that consume data with a high duty cycle, time spent waiting for data instead of processing the data, the expected performance is coupled with the off-chip memory bandwidth. Synthesizing a design can be a complex and time consuming process; moreover, simulating individual components abstracts away some of the complexities. The important takeaway from this simulation exploration is bandwidth estimates are based on general (broad) assumptions that potentially leave the designer with a false positive for the expected performance output.

The results discussed in this chapter will show that such expected simulation performance is not a reality. High performance computing is producing lofty goals and expecting significant performance gains through the use of reconfigurable

computing (FPGAs). This thesis investigates the use of commodity off-the-shelf IP cores for memory interconnects to determine the feasibility of using such cores in a large scale High Performance Computing cluster.

5.2 Simulation Results

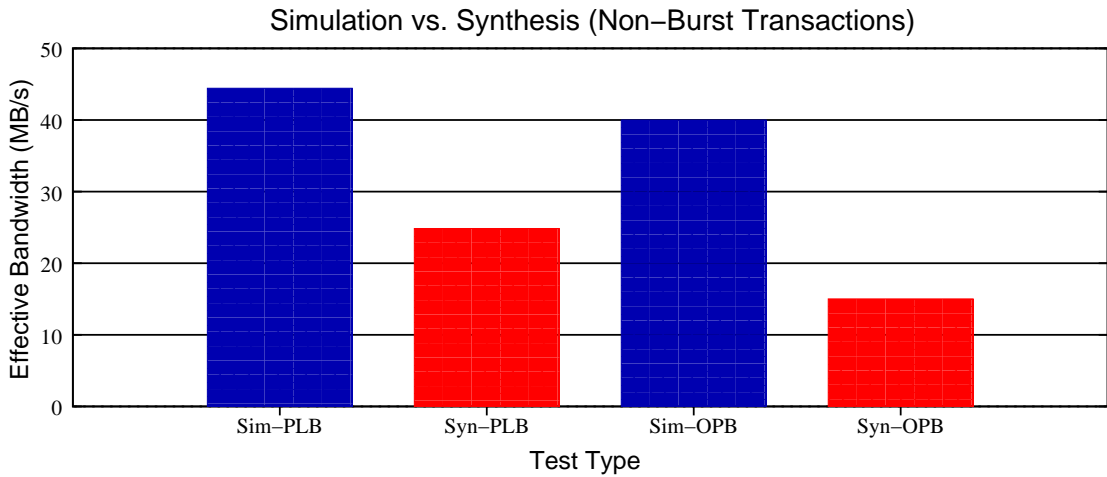
The tests were first simulated so that the synthesized results could be compared with the simulated results, a metric that we noted as distinguishing possible system performance based on simulation instead of synthesized results.

From Figure 5.1(a) we see that the simulated results of non-burst transactions are 44.44 MB/s or 5.56% of the PLB's theoretical bandwidth of 800 MB/s. On the OPB the bandwidth drops to 40 MB/s as a result of the OPB's bus width of 32-bits. Since the theoretical bandwidth of the OPB is only 400 MB/s, compared to 800 MB/s of the PLB, the simulated 40 MB/s is 10% of the OPB's theoretical bandwidth.

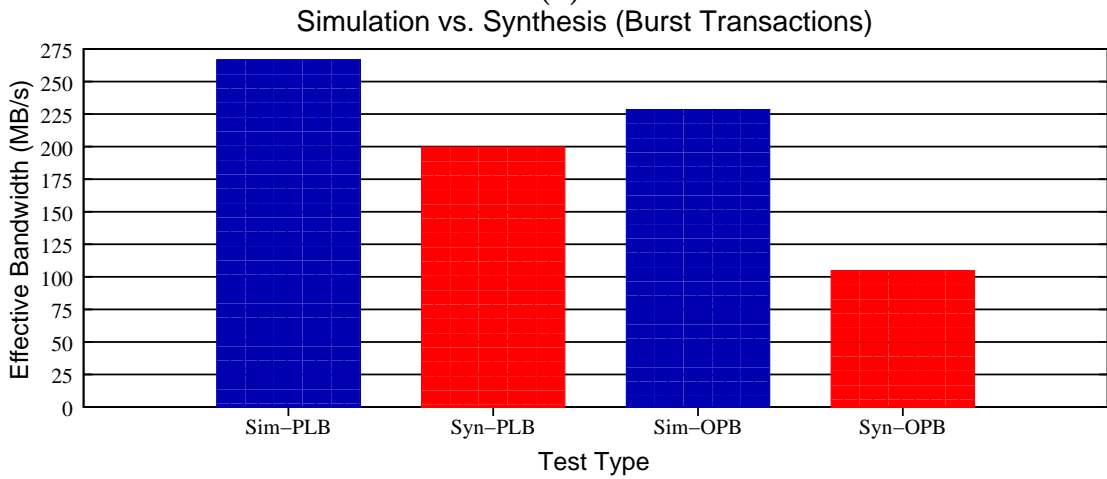
When considering sequential burst transactions the bandwidth is 266.67 MB/s or 33.3% on the PLB and 228.57 MB/s or 57.14% on the OPB shown in Figure 5.1(b). Under burst transactions the PLB has an advantage of transferring 128 bytes of data per request, whereas the OPB can only sustain 64 bytes. This is a current limitation of the PLB and OPB designed by IBM as discussed in Chapter 2.

While these simulated bandwidth results are low, a designer may consider them acceptable based on: the estimated bandwidth needs of the system, the ability to utilize burst transactions more often than non-burst, or the time and money required to design and implement an alternative memory interface. As a result the designer may determine the simulated bandwidth estimates sufficient and use

these bandwidth numbers (or worse, the theoretical SDRAM or bus bandwidth numbers) in the remainder of their system’s performance calculations.



(a)



(b)

Figure 5.1. Simulated off-chip effective bandwidth compared to synthesized effective bandwidth for (a) non-burst and (b) burst transactions

5.3 Single Core Synthesize Results

When synthesizing the design and running the same tests in hardware the results for both non-burst and burst transactions differ from 25% to a more drastic

54% and 62%. While the simulated results may be still less than what most system designers would like (more is always better), the difference between simulation and synthesis clearly shows some organizations will result in a decrease in system performance by up to 60% for applications with vast amounts of off-chip memory accesses.

The synthesized results shown in Figure 5.2 summarize the effective bandwidth under each of the different access patterns for a single core. Perhaps the most surprising result of the single core tests is the order in which memory is accessed is at best a secondary factor of the overall performance. That is, the bandwidth limitations of using a bus and its protocol are so substantial that an intelligent HPC core can only make modest improvements in RAM bandwidth by scheduling its memory accesses if a typical bus structure is employed. Indeed, the effective bandwidth of a core performing non-burst, strided, or random access patterns is at best a mere 24.85 MB/s or 3.11% of the theoretical bandwidth of the bus. When considering the theoretical bandwidth of a single DDR SDRAM (nominally between 1600 MB/s and 3200 MB/s according to the data sheet), a single core's effective bandwidth is between 0.78% and 1.55%!

5.3.1 Non-burst transaction results

When considering single data access, strided data access, and random access it is predicted single data access will outperform the rest. The reasoning for the prediction is due to the design of the DDR SDRAM. As discussed in the Chapter 2, DDR SDRAM accesses data through Column Address Select (CAS) and Row Address Select (RAS). Performing multiple requests within a close proximity in memory seemingly should result in a significantly higher bandwidth than random

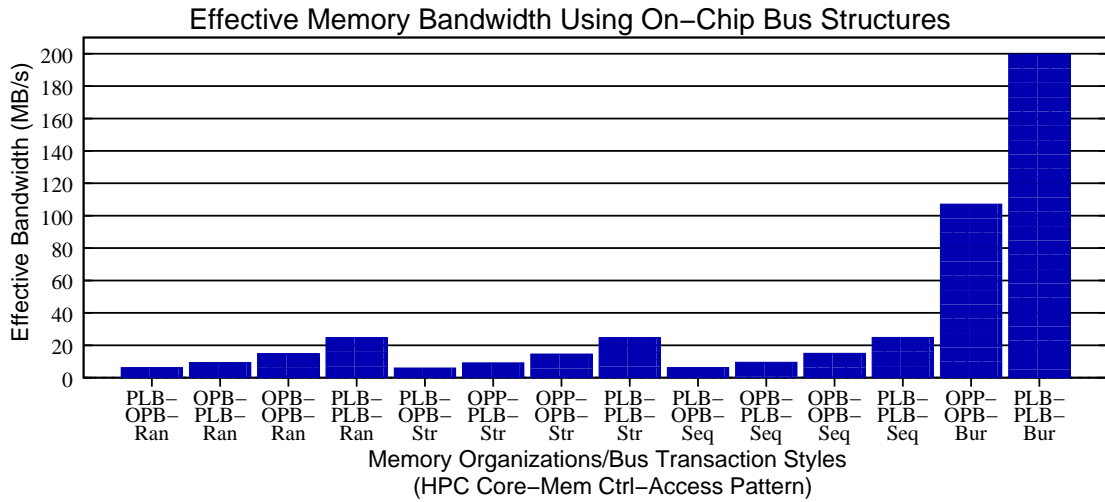


Figure 5.2. Single core effective memory bandwidth results

accesses since the SDRAM does not have to suffer the penalty of looking up data in a different row.

From the results we see the three different access patterns provide nearly identical results. An important question to then ask is – Why do these three access patterns produce such similar results? In Figure 4.2 we can see the connection between the core making the requests and the off-chip memory. This simple diagram shows that in order for a HPC core to access off-chip memory the request must first be issued from the master side of the core, through which the request is then issued through Xilinx’s IPIC and then through Xilinx’s IPIF before finally being considered by the bus’s arbiter. In the test the core is the only component connected to the bus that is making requests; therefore, we can eliminate bus contention from this discussion. The request is issued to the memory controller which issues the request to off-chip memory. This process is similarly repeated in reverse when the data is located in off-chip memory and made available to the memory controller to be sent back to the HPC core.

5.3.2 Burst transaction results

Burst transactions make a substantial improvement, providing up to 199.71 MB/s or 24.96% of the bus's theoretical bandwidth. This result is as expected since a single burst request can deliver up to 128 bytes of data and each word retrieved is on average twice as fast as a non-burst transaction. However, burst mode transactions only work for sequential access and are limited to lengths of 128 bytes (16 64-bit words). In addition, burst mode transactions increase HPC core design complexity which burdens hardware designers by requiring the core to receive the entire burst of data rather than a single word. Also, it is up to the programmer to implement a burst transaction rather than rely upon the hardware to know what type of transaction to implement because the hardware would not be able to determine if future requests will read from the next sequential location. If the application does not have a great deal of sequential accesses, then burst transactions will probably not benefit the application.

A burst transaction works by asserting a `IP2Bus_MstBurst` signal and specifying the transfer length (up to 16 64-bit words). In addition to the request, the core must be designed such that it can store each datum as it arrives from DDR across the bus. This adds to the complexity mentioned above where the hardware would need additional logic to handle either single transactions or burst transactions. The remainder of the burst transaction (signaling) is identical to the single read transaction.

Also, during a burst transaction the bus is dedicated to the core requesting the data until the transaction completes. When transferring up to 16 64-bit words this may improve a single core's bandwidth and performance as it can access memory on average faster than individually accessing memory; however,

with the addition of multiple cores, the remaining cores must wait for the burst transaction to complete before they can gain access to the bus. This increases bus contention and decreases overall system performance. Therefore it is important for the designer to consider the system as a whole instead of any single core's access patterns.

The OPB and PLB are limited in the number of cores that may master the bus. To support a larger number of cores, typical designs use a two-bus approach with bridges to connect the two. In the case of the off-the-shelf embedded systems cores, the bridge limits the bandwidth to 9.52 MB/s, illustrating the impact of the core's location on bandwidth. Also, the bridge itself is a component on both the OPB and PLB so there is an additional penalty of both buses granting the bridge access. When the off-chip memory is located on the opposite bus of the cores, each core must contend for the: bus, bridge, opposite bus, and off-chip memory.

High Performance Computing demands low latency access to memory. From the time the request is made until the data arrives, the time spent waiting for data is expensive, namely it is time the core or core's process is potentially idle waiting for the data. While a core could be designed to process other data while it is waiting many, general cases will result in the processor (core) idling or stalling.

5.4 Multiple Core Results

While the single core experiments provide a performance baseline, it is more realistic to assume a HPC application will have several parallel cores that need to share access to the off-chip memory. A problem arises of having multiple sinks (cores) and a single source (off-chip memory). In addition each core must vie for the bus, which connects the cores to the off-chip memory adding arbitration and

blocking times into the overall access time for each core. In all of the multiple core tests sequential access is the only access pattern considered due to the results of the single core's multiple access pattern.

The PLB is a more sophisticated bus than the OPB and requires more FPGA resources; the benefit becomes evident when more than one core is aggressively using the bus. Address pipelining is designed to improve bus bandwidth by allowing a new request to be overlapped with an ongoing data transfer. Along with separate address, read data, and write data buses, the PLB has been designed with high performance in mind [7]. Hence, two cores can overlap protocol and data communication resulting in more combined bandwidth than either could achieve as a single core.

The results illustrated in Figure 5.3(a) shows consistent performance for one and two cores under non-burst accesses; however, the bandwidth drops to 12.27 MB/s — a loss of 49% of the effective bandwidth of a single core — with four cores. The problem persists as eight cores are added and the bandwidth drops to 6.89 MB/s. Figure 5.4(a) As with the single core tests, burst transactions provide the best results. The non-burst trends carry over to the burst trends in that four cores average bandwidth drops by nearly 50% to 101.59 MB/s and eight cores drop to 69.81 MB/s.

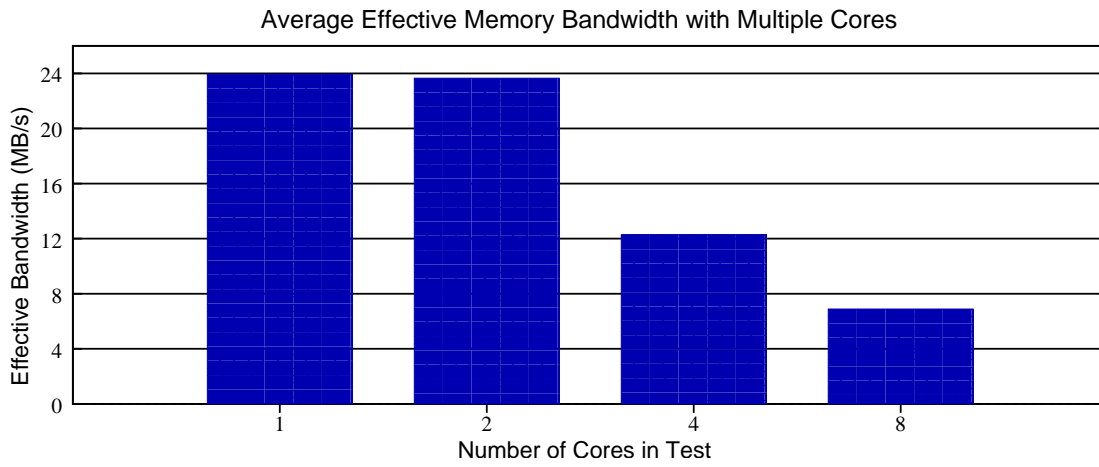
The bandwidth realized by the OPB even in a single core is nearly half of a single core on the PLB. Surprisingly, as multiple cores are added the average bandwidth achieved per core more closely matches that of the PLB. Where with a single core on the OPB the difference was 1.62 times the bandwidth achieved by the PLB, Figure 5.3(b) shows that four and eight cores provide only a 1.49 and 1.21 times increase respectively. In burst transactions seen in Figure 5.4(b), the

OPB bandwidth drops by 40% with two cores and nearly 70% with eight cores.

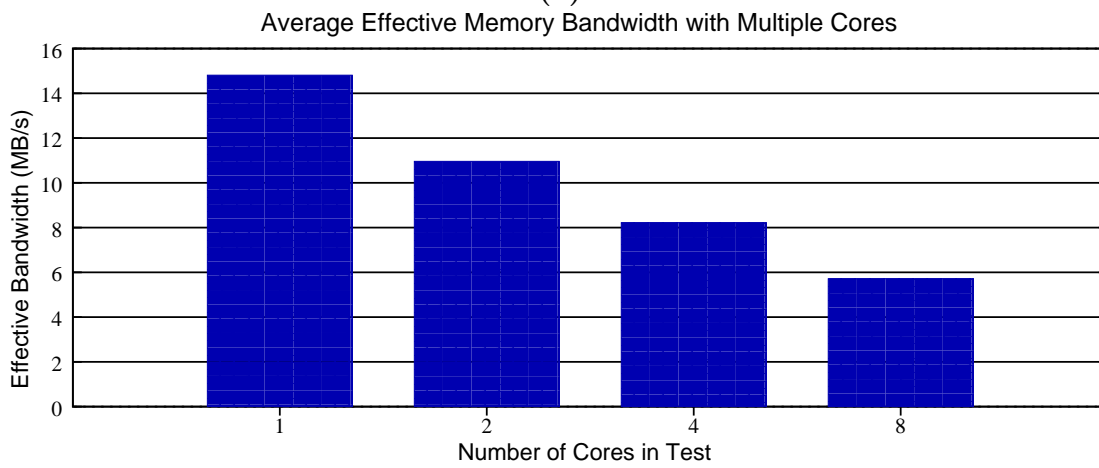
The OPB is a simpler design [6] and provides a common interface without requiring the investment in resources required to implement the PLB. The 32-bit data bus automatically reduces the effective bandwidth in half compared to the 64-bit bus of the PLB. The OPB also only provides primary and secondary arbitration and address parking instead of providing separate buses and address pipelining, as the PLB does.

As is evident from Figure 5.6 and Figure 5.5, the PLB is capable of improving efficiency as the number of cores demanding access increases. In contrast, the OPB is only able to sustain two high-demand cores before suffering the effects of bus contention. This means when designing a system with the OPB two cores will receive nearly double the bandwidth of the next two cores. While this may seem beneficial for systems with two cores demanding bandwidth, two cores is rather limited from an HPC point of view where tens or even hundreds of cores may need access to memory.

Broadly speaking, one would expect that the effective bandwidth, per core, would decrease as more cores share the resource; however, Figure 5.7 and Figure 5.8 show the total bandwidth actually realized increases with additional cores. The PLB and OPB are designed to connect up 16 master cores which means for the PLB, which uses separate address, data, and signaling buses along with address pipelining, it would be reasonable to speculate the aggregate bandwidth would continue to increase. The OPB, on the other hand, would not increase as much as the PLB due to its individual core bandwidth's characteristics where a loss of approximately 50% occurs for every two cores.

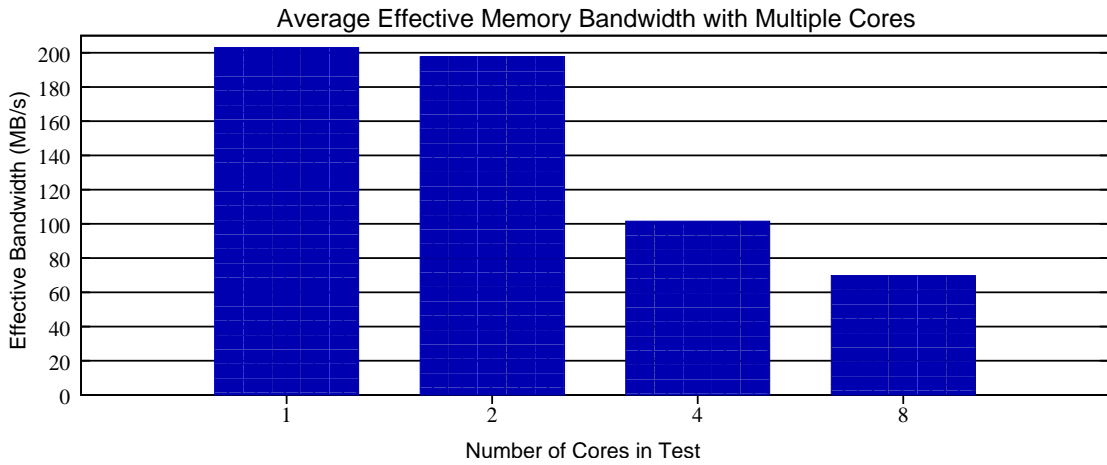


(a)

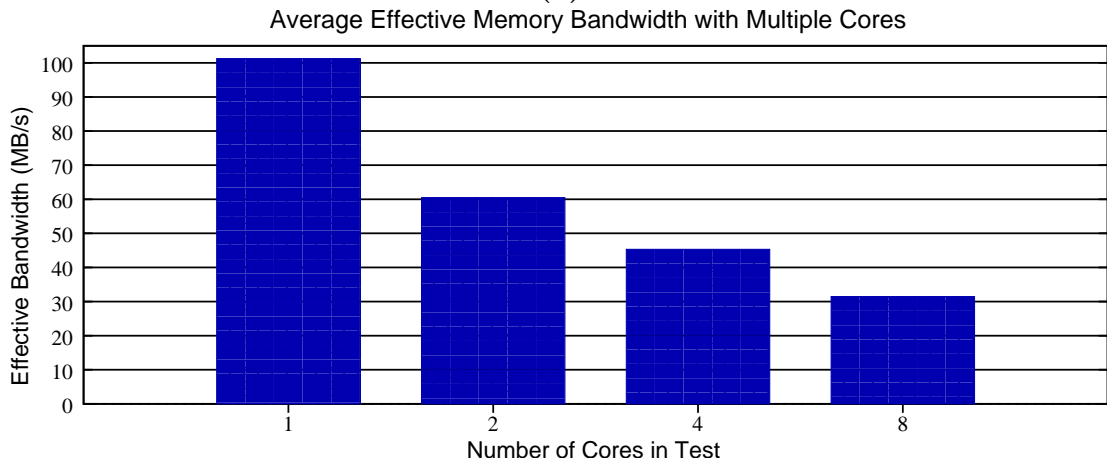


(b)

Figure 5.3. Average bandwidth of multiple cores and SDRAM connected on the same bus (a) PLB (b) OPB performing non-burst transaction

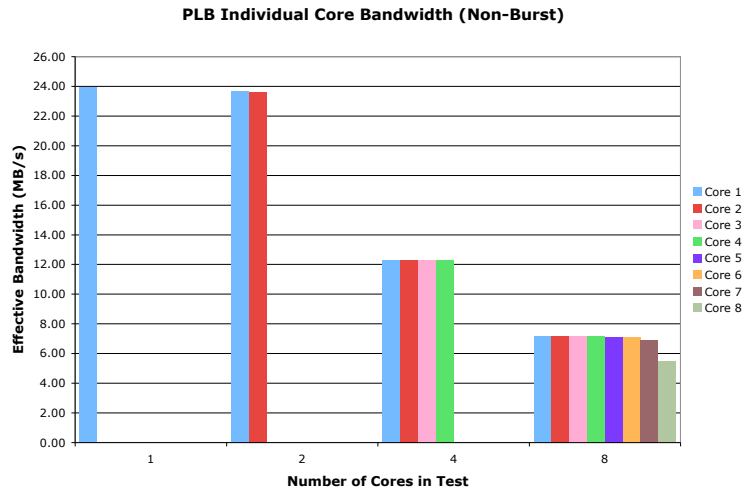


(a)

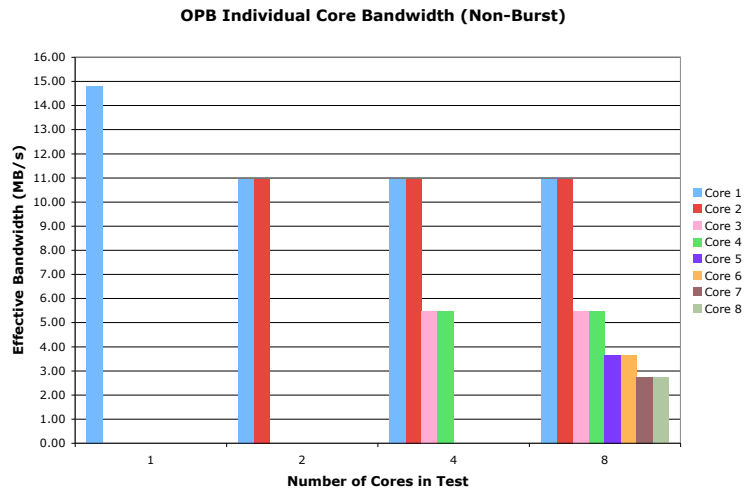


(b)

Figure 5.4. Average bandwidth of multiple cores and SDRAM connected on the same bus (a) PLB (b) OPB performing burst transaction

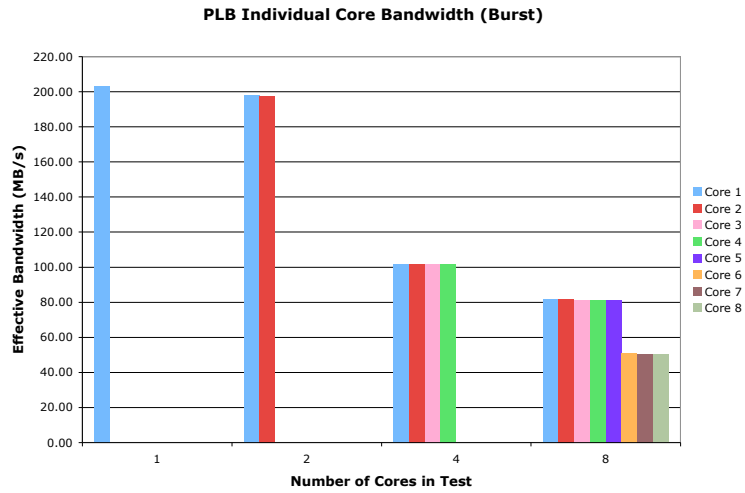


(a)

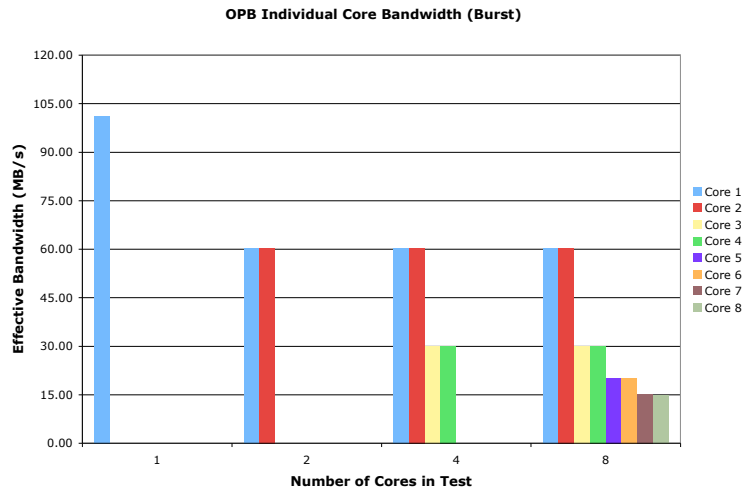


(b)

Figure 5.5. Individual bandwidth of multiple cores and SDRAM connected on the same bus (a) PLB (b) OPB performing non-burst transaction



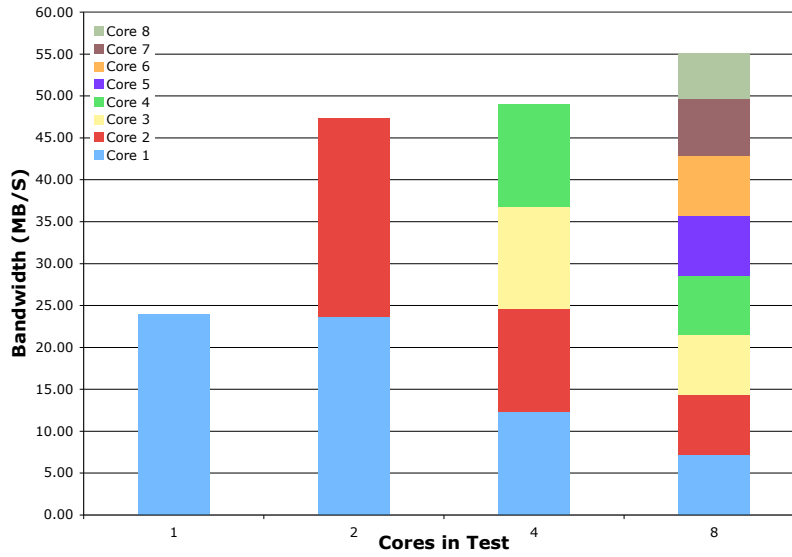
(a)



(b)

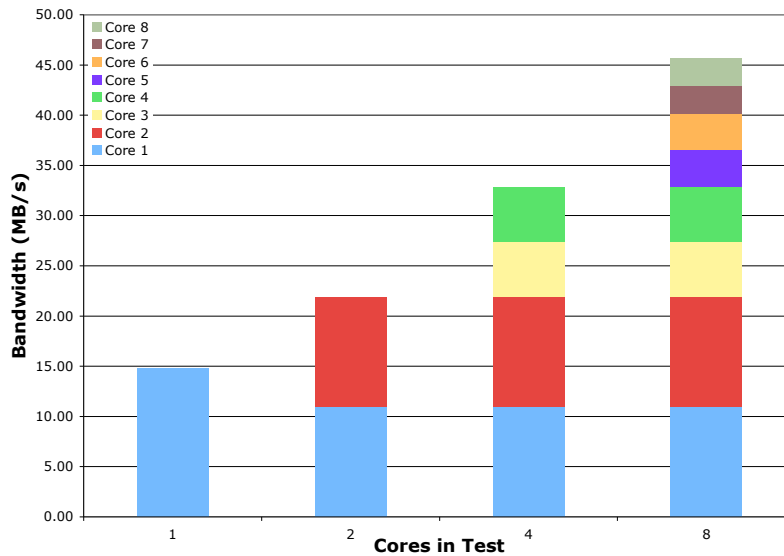
Figure 5.6. Individual bandwidth of multiple cores and SDRAM connected on the same bus (a) PLB (b) OPB performing burst transaction

PLB Aggregate Bandwidth (Non-Burst)



(a)

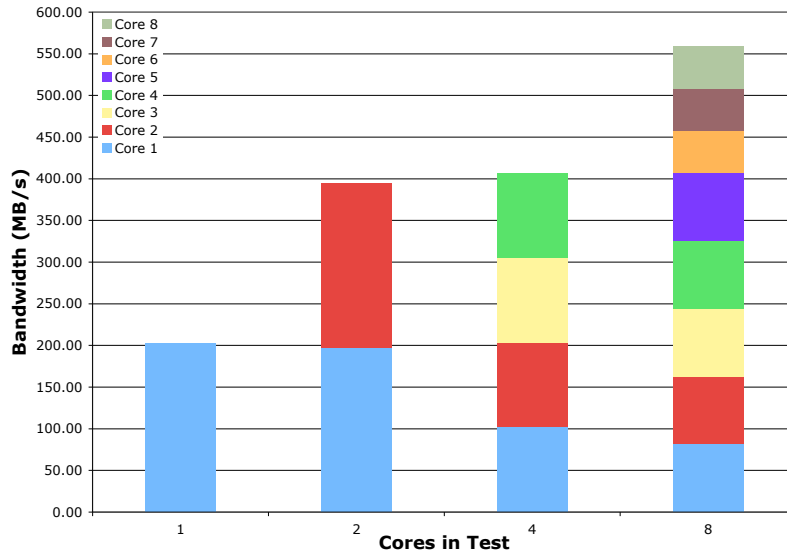
OPB Aggregate Bandwidth (Non-Burst)



(b)

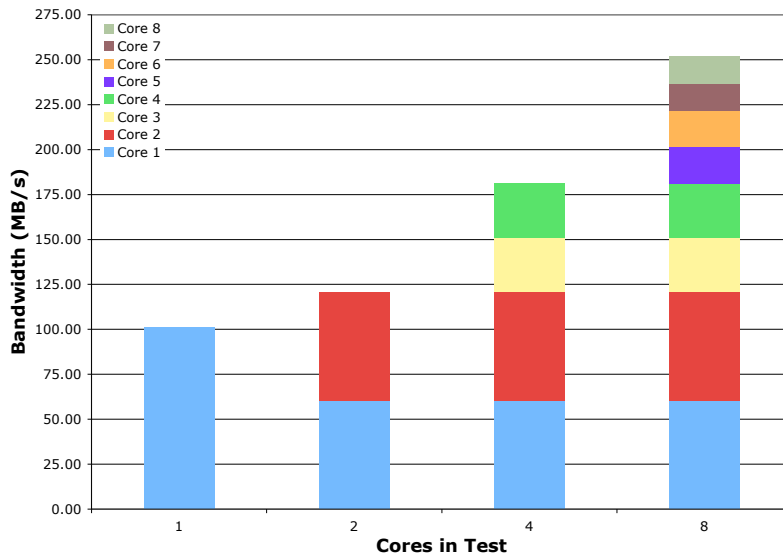
Figure 5.7. Aggregate bandwidth of multiple cores and SDRAM connected on the same bus (a) PLB (b) OPB performing non-burst transaction

PLB Aggregate Bandwidth (Burst)



(a)

OPB Aggregate Bandwidth (Burst)



(b)

Figure 5.8. Aggregate bandwidth of multiple cores and SDRAM connected on the same bus (a) PLB (b) OPB performing burst transaction

Chapter 6

Conclusion

The Reconfigurable Computing Cluster Project is investigating the use of Platform FPGAs in High Performance Computing (HPC) clusters. Specifically, the RCC Project is exploring the feasibility of building a cost-effective cluster capable of scaling to the PetaFLOPS (10^{15} floating-point operations per second) range. One part of this analysis is to understand the performance currently available off-the-shelf components could provide such a system.

This thesis focused on the effective bandwidth between hardware cores and off-chip memory to determine the feasibility for leveraging existing components and bus structures to support science applications high bandwidth requirements. While these components provide flexibility and convenience, which allows them to be used in a variety system designs, the price we pay for general-purpose is performance.

First, we empirically characterized the performance of several CoreConnect on-chip memory subsystem under three common HPC memory access patterns. Second, based on that data, synthesized a set design guidelines for getting the most performance out of the memory subsystem for High-Performance Computing. By

addressing these two goals, we can answer the question of feasibility.

Our results show quite clearly the bus is a limiting factor in off-chip memory access. So much so that the bandwidth achieved by even a single core, regardless of access pattern, is a mere 3% of the total theoretical bandwidth of the bus. The results are even worse when considering DDR SDRAM's theoretical bandwidth of up to 3200 MB/s which results in 0.78% utilization, less than a percent. An improvement in performance requires burst transactions which provides 25% of the bus's bandwidth, but again a mere 6% of DDR SDRAM's bandwidth. The location of the core within the system effects the performance as well. The PLB was designed with high performance in mind while the OPB is traditionally utilized by lower performance demanding components. Cores on the OPB obtain only 62% of the bandwidth of the PLB. Next, adding a bridge to connect two buses reduces the performance by adding additional overhead associated with traversing both buses and the bridge. Finally, adding multiple cores to the system better characterizes performance achieved by High Performance Computing. The buses are capable of connecting multiple cores and the PLB is designed to sustaining multiple requests; however, our results show a performance loss of 50% for four cores and 70% for eight cores.

A mere 1.6% - 4% of the bus's theoretical bandwidth for non-burst transactions and 25% for burst transactions highlights the vast potential to improve bandwidth. While the bus does offer many advantages, especially to embedded systems applications where memory bandwidth is not as serious of a concern, the results raise doubts for its impact on overall HPC applications performance. Fortunately, by doubling the effective bandwidth — which efforts from this thesis suggest is plausible — it is likely to double the rate-of-computation by provid-

ing each computational core with data faster, reducing the core's idle time and improving performance.

6.1 Future Work

This research has investigated the currently available off-the-shelf component's performance in order to understand what the potential for what future improvement might be. Our work has shown thus far that a bus architecture will not suffice. In addition, cores which are commonly available may provide more generic interfaces in order to provide portability instead of efficiency. Changing our focus now to how to most effectively increase performance, we are currently considering some of the following options as possible extensions to this work.

From our research we found burst transactions provided a significant increase in performance over non-burst transactions. While burst transactions are limited by sequential access, a split bus approach could be considered. Off-chip memory would connect to a small number of cores through one bus, each of which perform transactions on behalf of cores, connected on a second bus, needing to access off-chip memory. It may be possible to intelligently access off-chip memory more efficiently under this interface at the cost of FPGA resources.

In commodity processors a cache is the current solution improve latency; however, current implementations of caches for FPGAs do not provide nearly the same performance as caches designed, and hand tuned, for processors. As a result the resource utilization for a cache has left it as a less than ideal approach for FPGAs. However, it is worth investigating further as FPGAs continue to increase the amount of resources available.

Under a bus structure the amount of routing resources is minimized to connect

multiple cores together along with off-chip memory. In addition multiple cores can easily be added to the design and all have access to memory. It is worth investigating the potential increase in performance a single core would have to off-chip memory, without traversing or contending for the bus.

Alternatively focusing on a change in the actual memory used may provide the desired improvement in performance. Multiple channels of memory would effectively provide access to a wider memory increasing theoretical bandwidth by two or even four fold. Newer memory modules, such as Fully Buffered RAM, may also hold the key for performance increases. While we remain limited in memory access due to pin communication electrical characteristics, it may be possible design an alternative memory controller and off-chip memory interface to retrieve data at a higher frequency or with decreased latency.

These back of the envelope ideas are only to shed some light on the vast potential research opportunities available within this field. Clearly High Performance Computing will remain in the academic, commercial, and national interest and as such it is necessary to continue to speculate, investigate and innovate in order to push the computational envelope.

References

- [1] Double data rate (ddr) sdram specification. JEDEC Standard, JESD79E, May 2005.
- [2] E. Anderson. *Abstracting the Hardware/Software Boundary through a Standard System Support Layer and Architecture*. PhD thesis, University of Kansas, May 2007.
- [3] E. Anderson, W. Peck, J. Stevens, J. Agron, F. Baijot, S. Warn, , and D. Andrews. Memory hierarchy for mcsopc multithreaded systems wesley peck. In **to appear in** *Engineering of Reconfigurable Systems and Algorithms (ERSA'07)*, page ???, 2007.
- [4] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.
- [5] J. Cong and C. Wu. Optimal fpga mapping and retiming with efficient initial state computation. In *DAC '98: Proceedings of the 35th annual conference on Design automation*, pages 330–335, New York, NY, USA, 1998. ACM Press.
- [6] I. Corporation. 32-bit on-chip peripheral bus architecture specifications version 2.1, April 2001.
- [7] I. Corporation. 64-bit processor local bus architecture specifications version 3.5, May 2001.
- [8] I. Corporation, I. Technologies, and K. Technology. Intel dual-channel ddr memory architecture white paper, September 2003.

- [9] W. J. Dally, F. Labonte, A. Das, P. Hanrahan, J.-H. Ahn, J. Gummaraju, M. Erez, N. Jayasena, I. Buck, T. J. Knight, and U. J. Kapasi. Merrimac: Supercomputing with streams. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 35, Washington, DC, USA, 2003. IEEE Computer Society.
- [10] B. Donchev, G. Kuzmanov, and G. N. Gaydadjiev. External memory controller for virtex ii pro. In *in Proceedings of International Symposium on System-on-Chip 2006*, pages 37–40, November 2006.
- [11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.
- [12] J. Gummaraju and M. Rosenblum. Stream Programming on General-Purpose Processors. In *MICRO 38: Proceedings of the 38th annual ACM/IEEE international symposium on Microarchitecture*, Barcelona, Spain, November 2005.
- [13] J. Haas and P. Vogt. Fully-buffered dimm technology moves enterprise platforms to the next level, March 2005.
- [14] J. L. Hennessey and D. A. Patterson. *Computer Organization and Design: The Hardware/Software Interface; 2nd edition*. Morgan Kaufmann, 1994. HEN j2 98:1 P-Ex.
- [15] J. L. Hennessey and D. A. Patterson. *Computer Architecture : A Quantitative Approach; second edition*. Morgan Kaufmann, 1996. HEN j2 96:1 1.Ex.
- [16] IBM. Url: <http://www-03.ibm.com/chips/products/coreconnect/>, June 2004.
- [17] W. Industries. Wintec 32mx72 industry standard 184-pin low profile registered ddr400b-333 sdram dimm, 2000.
- [18] E. Joseph, A. Snell, and C. G. Willard. Council on competitiveness study of U.S. industrial HPC users, July 2004. Sponsored by Defense Advanced Research Projects Agency.
- [19] E. Joseph, A. Snell, C. G. Willard, S. Tichenor, D. Shaffer, and S. Conway. Council on competitiveness study of ISVs serving the high performance computing mar-

- ket: The need for better application software. http://www.compete.org/hpc/hpc_software_survey.asp, Aug. 2005. Sponsored by Defense Advanced Research Projects Agency.
- [20] H. Lange and A. Koch. Memory access schemes for configurable processors. In *FPL '00: Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, pages 615–625, London, UK, 2000. Springer-Verlag.
- [21] T.-L. Lee and N. W. Bergmann. An interface methodology for retargettable fpga peripherals. In *Engineering of Reconfigurable Systems and Algorithms*, pages 167–173, 2003.
- [22] P. Marchal. Field-programmable gate arrays. *Commun. ACM*, 42(4):57–59, 1999.
- [23] R. Miller and J. Bellan. Direct numerical simulation and subgrid analysis of a transitional droplet laden mixing layer. *Phys. Fluids*, 12(3):650–671, 2000.
- [24] J. Noseworthy and M. Leeser. Efficient use of communications between an fpgas embedded processor and its reconfigurable logic. In *ERSA*, pages 191–197, 2006.
- [25] V. Pande. Folding@home: Advances in biophysics and biomedicine from worldwide grid computing (invited keynote). In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - HiCOMB Workshop 7*, page 196.2, Washington, DC, USA, Apr. 2005. IEEE Computer Society.
- [26] J. T. R. C. O. S. T. B. S. M. E. W. R. B. Tremaine, P. A. Franaszek and P. M. Bland. Ibm memory expansion technology (mxt), January 2001.
- [27] R. Sass, W. V. Kritikos, A. G. Schmidt, S. Beeravolu, P. Beeraka, K. Datta, D. Andrews, R. S. Miller, and J. Daniel Stanzione. Reconfigurable computing cluster (rcc) project: Investigating the feasibility of fpga-based petascale computing. In **to appear in** *FCCM '07: Proceedings of the 13th Annual IEEE Symposium on*

- Field-Programmable Custom Computing Machines (FCCM'07)*, page ???, Washington, DC, USA, 2007. IEEE Computer Society.
- [28] L. Semiconductor. Solving high-speed memory interface challenges with low-cost fpgas, May 2005.
- [29] K. Underwood and K. S. Hemmert. Closing the gap: CPU and FPGA trends in sustainable floating-point blas performance. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 219–228, April 2004.
- [30] U.S. Food and Drug Administration. Parkinson’s disease new treatments slow onslaught of symptoms, June 2004.
- [31] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snaveley. Quantifying locality in the memory access patterns of hpc applications. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 50, Washington, DC, USA, 2005. IEEE Computer Society.
- [32] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, 1995.
- [33] Xilinx. Programmable logic devices. <http://www.xilinx.com/>. Last accessed April 20, 2007.
- [34] Xilinx. Opb central dma controller, December 2005.
- [35] Xilinx. Opb ipif (v3.01c), December 2005.
- [36] Xilinx. Opb to plb bridge (v1.00c), October 2005.
- [37] Xilinx. Plb double data rate (ddr) synchronous dram (sdram) controller, December 2005.
- [38] Xilinx. Plb ipif (v2.02a), April 2005.
- [39] Xilinx. Plb to opb bridge (v1.01a), July 2005.
- [40] Xilinx. Virtex-ii pro and virtex-ii prox platform fpgas: Complete data sheet, October 2005.
- [41] Xilinx. Opb double data rate (ddr) synchronous dram (sdram) controller, March 2006.