University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

The Impact of the Memory Subsystem on the Performance of MicroBlaze Processors

Harris Corporation
25 Dyas Road
Toronto, Ontario, Canada

Prepared by
Paul William Roukema
ID XXXXXXXX
userid pwroukem
2A Computer Engineering
13 October 2012

26 Gretna Green
London, ON, Canada
N6G 1X1

13 October 2012

Manoj Sachdev, chair
Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Sir:

This report, entitled "The Impact of the Memory Subsystem on the Performance of MicroBlaze Processors", was prepared as my 2A Work Report for the University of Waterloo. This report is in fulfillment of the course WKRPT 200. The purpose of this report is to analyze the impact of various memory subsystem configurations and optimizations on overall system performance in embedded systems utilizing Xilinx MicroBlaze soft-core processors, with the overall goal of finding an appropriate configuration for use in a situation encountered during my work term.

Harris Corporation's Broadcast Communications Division is a leading developer and manufacturer of television and radio broadcast systems and services, including transmission, infrastructure and networking, software solutions and digital signage.

The Video Processing R&D group, in which I was employed, is managed by Bogdan Szmajda. The group develops a range of video processing products including up/down converters and synchronizers in both standalone and modular configurations. This report was written for Clarence Ip and Radu Iliescu.

I would like to thank Radu Iliescu for helping me understand the memory configuration of the system being analysed. I would also like to thank Mr. Douglas Wilhelm Harder for creating the Microsft Word template and guide this report is based on. I would like to thank my classmate Jonathan Jekir for proofreading this report. I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Paul William Roukema
ID XXXXXXXX

Contributions

The team I worked with was relatively small. It consisted of 15 people, although only between 7 and 9 of them were working on the same project as I was at any point in time. This team constituted a small fraction of the employees at the office where I was located. In addition to the primary team I worked with I had some interactions with a second team working on related projects at another office.

The team's main goal was the development of a new high density frame synchronizer and up/down/cross converter to serve as the flagship item in a new product line. As the first major module development project occurring for this platform, this project presented a number of interesting problems and decisions. I found the process used and decisions made in the course of the project to be quite enlightening.

My tasks were the development of the supporting software for the audio I/O expansion cards for the product, the development of control software for an add-on audio processing module and various board support items. This included developing the software for the audio expansion boards almost completely from scratch, starting from a basic test shell application. Code that had already been developed had to be ported to the platform with the minimum possible set of changes. In addition I developed a communications protocol for controlling the expansion cards over a high speed serial link from the primary card.

My work regarding the audio processing module included the development of resource allocation algorithms and strategies in order to ensure stable packing of configurations into the available resources in an array of 6 DSP chips connected across 3 different processing boards. In addition I developed the user interface definitions for the processing functionality as well as implementing much of the back end functionality for transforming the using interface parameters into usable configurations.

My board support work included the integration of several standard Linux packages into the build system being used for the project, as well as a number of more technical tasks. This included low level debugging and bootloader modifications in order to enable higher

serial link speed. The final major element of my board support work was to significantly increase the performance of our kernel mode bridge driver which communicates between our primary application and our custom hardware.

The relationship between this report and my job is that, during the development of the software for the audio expansion cards mentioned above, I encountered a number of situations where performance limitations presented themselves. These limitations were largely related to the memory subsystem of the processor involved. I have often read about the effects of memory bandwidth and latency of computer performance; however I had not previously dealt with systems where the parameters of the memory subsystem were under my control. In determining the best approach for solving these performance issues I compiled the data used for the report.

In the broader scheme of things, the software I developed touches on several key components of the product being developed. In particular I was the primary developer for much of the advanced audio functionality of the project, working with a number of tools and technologies that were relatively new to the company. This audio functionality is one of several areas where the new product will be on the cutting edge of the industry, helping to build prestige and drive sales for the company.

Summary

The main purpose of the report is to determine the appropriate memory configuration for the Xilinx MicroBlaze processors being used with external memory in new product modules. The selection of an appropriate configuration is essential in order to meet performance goals and provide a responsive user experience.

The major points covered in this report are an overview of the existing system, an exploration of the options for improving performance, evaluation of these options and selection based on the evaluation. Since the design of the hardware is considered fixed only options involving pure programmable changes are considered for use.

The major conclusions in this report are that the use of an instruction cache is critical to providing reasonable performance, resulting in more than an order of magnitude performance increase during evaluation. In addition, for some workloads a data cache also results in significant performance increase, although its use in not absolutely required.

The major recommendations in this report are that MicroBlaze based systems using external memory utilize one of two memory configurations. The first and preferred configuration is to store code, data and program stack in the external memory and to utilize both an instruction and data cache. The second, slightly less desirable configuration would be to store code and data in the external memory while keeping the program stack in internal memory. In this configuration only an instruction cache is required. Either configuration offers reasonable performance for the applications being examined, however the second configuration is somewhat slower.

Table of Contents

List of Figures

List of Tables

1 Introduction

Harris Corporation's Broadcast Communications division is a worldwide supplier of broadcast equipment and software. It serves customers in over 150 countries, providing highly integrated solutions to meet the needs of broadcasters worldwide [**1**]. Harris has a history of innovation, providing over 70 technical firsts in the broadcast industry. The purpose of this report is to examine the impact of various elements for the memory hierarchy on the performance of a processor being used in a new product design. In this section, background on the product and problem being worked on is presented, in addition to information on the scope and outline of the report.

1.1 Background

In the broadcast industry, many products are built in a modular manner, using a "frame" or "mainframe" architecture, in which a customer can purchase a single rack mountable enclosure and then based on their specific needs, purchase a number of modules or cards, which are hosted by the enclosure and provide the actual processing functionality desired by the customer. Under this architecture, the frame has traditionally provided power for each module, in addition to any required cooling. Early iterations of these designs provided no means for management and control of the modules within the frame. Module controls, if any, were typically limited to switches or potentiometers mounted on the cards, preferably, although not always, along the front edge.

Towards the late 1990s product complexity grew and modules began featuring microcontrollers for the management of internal functionality. Once functionality became managed by software, controlling the parameters of the modules remotely became practical, first using asynchronous serial links and later over IP networks. At this point in time essentially all new products featured some type of microprocessor for management purposes.

A parallel development was the introduction of FPGAs to product designs. These programmable devices largely replaced custom ASICs, ASSPs and discrete logic in the

processing path, replacing them with custom tailored, field updateable logic. Despite the silicon area, speed and power disadvantages of FPGAs relative to hard silicon devices, the flexibility, rapid turnaround and economies of scale of FPGAs have won out, at least in the broadcast industry.

Traditionally, the microprocessors used in products have been standalone microcontrollers, including various 8051, Atmel AT91 ARM7 or Freescale MPC8xx series PowerPC devices. Since the microcontrollers are simply digital integrated circuits, there is no fundamental reason that these dedicated chips cannot be subsumed into one of the FPGAs in a product. When a microprocessor is implemented within an FPGA it is commonly termed to be a soft-core processor, indicating that the CPU core is itself almost a piece of software. In contrast to traditional "hard" designs soft-core microprocessors allow the designer to tailor many elements of the processor design to their specific application, including ISA features, interrupt handling and memory hierarchy. This gives great freedom to the designer but also requires that they pay attention a number of system design elements that are usually under the purview of the microcontroller designer.

1.2 Scope

In the case of the particular project being examined in this report there are two similar modules being developed. Both modules utilize a single FPGA with no external processor, instead relying on a soft-core processor within the FPGA. This configuration had not previously been used in any of the group's prior projects. In addition the product family these modules were being developed for was a new design and not all of the product family specific communication and control protocols had been completed during the module's design phase.

The hardware on the modules being developed was already essentially fixed at the time of this research. Each module featured a Xilinx Spartan 6 FPGA, a Numonyx NOR flash device and an ISSI asynchronous SRAM. Xilinx offers two internally developed soft-core processors, the 8-bit PicoBlaze and the 32-bit MicroBlaze. For this application the PicoBlaze is inappropriate due to code size limitations and the lack of a C compiler [**2**].

The MicroBlaze processor is a typical 32-bit RISC design, although some changes have been made to better utilize the flexibility afforded by FPGA based design. It is supported by a port of the GNU Compiler Collection and a set of libraries provided by Xilinx. The MicroBlaze core was selected for this project. Other synthesizable cores, such as the ARM Cortex-M1 or Altium TSK3000 were not considered for cost and toolchain integration reasons. For the purposes of this report, all non-programmable hardware is considered fixed.

Analysis of memory hierarchy impacts will be limited to the placement of code and data across internal and external memory, both with and without caches enabled. Memory timings will be limited to the optimal values for the available memories. Different memory types outside of those natively available on the board will not be considered.

2 Description of System

The embedded system being examined is implemented in two real world systems. Both implementations use the same external components and the same embedded system configuration. The only relevant distinction for the purposes of this report is the precise FPGA used. Both systems use Xilinx Spartan 6 LXT devices in the FGG484 BGA package, but the first implementation utilizes a XC6SLX45T device while the second implementation uses a larger XC6SLX150T device. The difference is only relevant in relation to potential resource constraints; however these were largely dictated by the surrounding non-processor logic, rather than just the device size.

The embedded system consists of a processor core and a set of peripheral devices. In this case the processor core is a Xilinx MicroBlaze 7.20d core [**3**]. Optional processing features of the MicroBlaze processor that were enabled for these tests are the MUL32 hardware multiplier, which provides fast division support, the PCMP instructions, which provide pattern compare abilities for the processor, and the MSR bit set and clear instructions, which provide atomic set/clear capability for the MSR register. The overall implementation selected was the area optimized version of the processor, which uses a 3 stage pipeline instead of the 5 stage pipeline of the normal version. This trades maximum

clock frequency and performance for reduced implementation resources. In the implementation described here the processor clock is 50 MHz and the same clock is used for all attached peripherals as well.

The MicroBlaze processor utilizes a Harvard memory architecture, with separated instruction and data paths. While this configuration is enforced at a low level, typical implementations result in the code and data occupying a shared address space. As a result of the split paths, the MicroBlaze processor supports up to 8 separate bus masters. There are a total of 4 supported bus protocols with each implementing a separate master for instructions and data. The supported buses are the Xilinx specific LMB, which is used to connect to internal BRAMs with single cycle access times, providing maximum performance operation, and the Xilinx specific XCL, which is used to connect instruction and data caches direct to memory controller. In addition to these internal, high-speed, busses, the processor also supports two busses from IBM's CoreConnect portfolio. These are the PLB protocol, which is used for most peripherals and the OPB protocol, which is not used in this implementation.

In the system being examined there are 3 levels of memory available by speed before considering caching. The memory and bus configuration used is seen in Figure 2-1. The fastest memory is the 64kB of internal BRAM connected to the LMB. This memory supports single cycle access, providing equivalent performance to a cache that always hits. This memory is also implemented in a dual ported manner, taking full advantage of the split instruction and data buses to allow simultaneous instruction and data fetch. This memory is pre-populated by the FPGA when it loads a bitstream, allowing this memory to be used to store initial boot instructions.

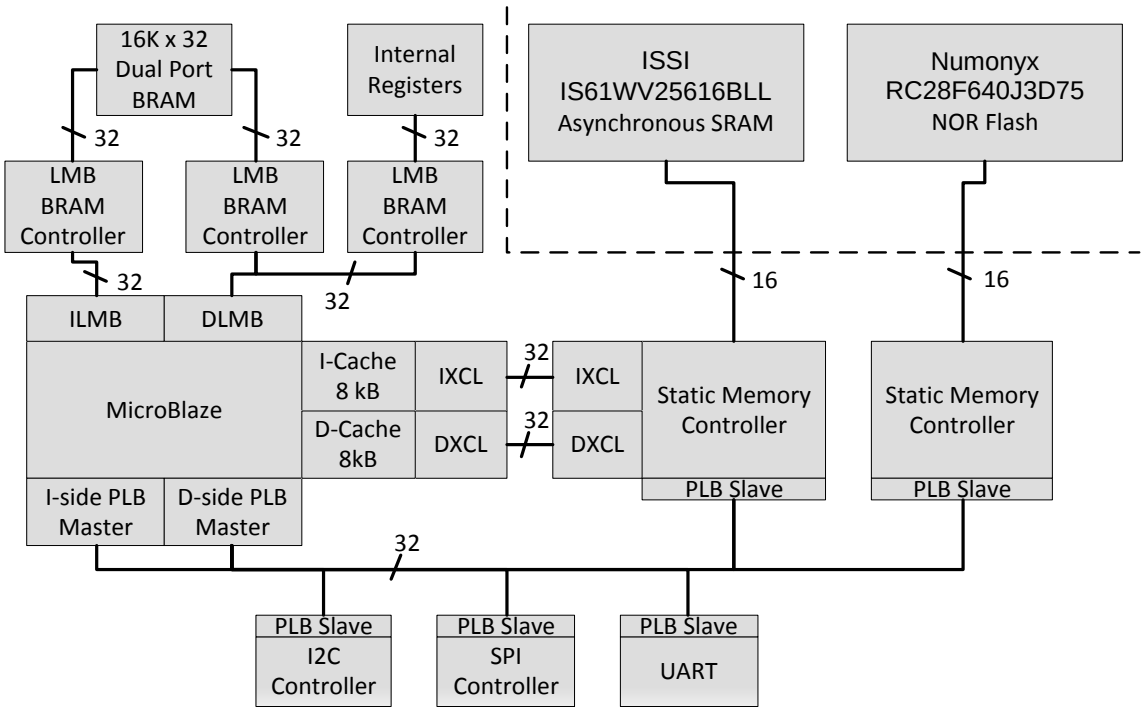

Figure 2-1: System bus and memory configuration

The next memory is an external 512 kB asynchronous SRAM memory connected to the system in two ways, depending on whether the caches are enabled. If caches are enabled, then all accesses flow through the two XCL interfaces. If a cache hit occurs this allows for simultaneous code and data accesses. If the cache is disabled, accesses go to one of the two PLB masters of the MicroBlaze and access the external memory controller through the single shared PLB. In addition the SRAM is organized in a 16-bit width, so all word reads require a multi-cycle burst. As this is an asynchronous device, the memory timings, which are listed in Appendix A, are specified in terms of time spans rather than a single operating frequency.

The last memory in this system is the external flash memory. This is attached to the FPGA's configuration port and is used to store both FPGA configurations and the software images to be loaded into the SRAM. Because of the access patterns used with this memory, namely single sequential read passes during system boot; performance is not a significant concern and is not considered in this report.

3 Requirements, Criteria and Metrics

One of the primary functions of the system being examined is to configure an audio processing unit. These processing units accept 4 kB configurations, which are generated on a primary control board and have to be sent to these modules. The communications protocol used between the modules is limited in packet size, requiring several hundred packets to send the data required for a single configuration. Due to the nature of the data it is possible to obtain a typical compression ratio between 25 % and 33 % using a custom RLE compression algorithm. This significantly reduces the number of packets that need to be transferred, improving performance. In order for this algorithm to be practical however, it is necessary for the execution time of the compression and decompression routines of a representative 4 kB block of configuration data to be suitably fast. In this case, based on desired system responsiveness and existing latencies, 10 ms was selected as the upper threshold for the execution time in either direction.

Although the execution time of this algorithm was one of the primary drivers for the examination of this system's performance, the general performance characteristics of this system are also of interest. To this end, the Xilinx provided port of the industry standard Dhrystone benchmark was also selected as a performance analysis tool [**4**]. This benchmark is commonly used when examining the integer performance of computer systems, especially in the embedded domain. As a synthetic benchmark based on a range of use cases originating from 1984 it is not truly representative of the actual application code being run in this system, but because of its wide industry use it provides an interesting basis for comparison.

In addition to the two application level metrics analyzed, low level memory bandwidth data was also collected. This utilized two different sizes of memory block in order to examine both fully cacheable and non-cacheable scenarios. The first block size was 4 kB, which fit comfortably inside the cache used for testing. The other block size was 32 kB, which is small enough to allow testing using the internal BRAM, but exceeds the size of the internal data cache.

The primary criterion used for evaluation will be the ability of a configuration to execute the compression code at sufficient speed, which forms a first gate for any memory configuration. A secondary consideration is the resource utilization of the configuration. The ability to add or expand product capabilities later is life is important and keeping resource utilization down is a key enabler.

4 Possible Configurations

Given the available external and internal memories, as well as the inherent flexibility of an FPGA based system, a number of possible configurations arise. These configurations revolve around the placement of code, data and program stack space across cached and uncached memories both inside and outside the FPGA. By exploring this problem space the optimum solution can be identified.

Using a combination of standard linker scripts and compiler function and variable attributes the positioning of code and data can be varied between internal BRAM and external SRAM. Using this approach in combination with enabling and disabling the instruction and data caches yields the configurations in Table 4-1. Some of the configurations outlined in this table are not implementable for the final system, but serve as baselines for the ideal case. In particular these are configurations 1, 2, and 5. All of these cases assume the availability of large amounts of BRAM, which is not the case in the system being examined. In particular, placing all code or data in the internal BRAM is not practical due to code and data size constraints. Placing partial code elements into BRAM is possible, and will produce similar results to the benchmark as long as all benchmarked code is placed in BRAM.

Table 4-1: Memory configurations

Configuration	Location			Cache	
	Code	Data	Stack	Instruction	Data
1	BRAM	BRAM	BRAM	Disabled	Disabled
2	BRAM	SRAM	BRAM	Disabled	Disabled
3	SRAM	SRAM	BRAM	Disabled	Disabled
4	SRAM	SRAM	SRAM	Disabled	Disabled
5	BRAM	SRAM	BRAM	Disabled	Enabled
6	SRAM	SRAM	BRAM	Enabled	Enabled
7	SRAM	SRAM	BRAM	Enabled	Disabled
8	SRAM	SRAM	SRAM	Enabled	Disabled
9	SRAM	SRAM	SRAM	Enabled	Enabled

One of the primary methods used in modern processors to improve memory performance is caching. Caches provide low latency access to code and data nominally placed in main memory, hiding the latency and bandwidth costs of actual memory accesses. In addition, caches are constructed using memory groupings called cache lines, which are typically a power-of-two multiple of the machine word size in length. By accessing these larger linear blocks of memory, caches encourage access patterns conducive to burst reads and writes, allowing increased performance when accesses to main memory are required.

On the MicroBlaze processor, as with many modern processors, the first level (in the case of MicroBlaze, only level) cache is separated into an instruction cache and a data cache. The automatic instruction fetch of the processor, as driven by the program counter is vectored though the instruction cache while all other access are routed to the data cache. This also allows the instruction cache to be simpler since it has no need to handle write cycles, only invalidation. For the purposes of this testing both caches were configured with a size of 8 kB using a cache line size of 4 32 bit words. Both caches are direct mapped, which simplifies design at the cost of reduced operational efficiency under some use patterns. Both caches are configured to use the entire 512 kB size of the external SRAM. The combination of these options results in a 6-bit cache tag size. The data cache is configured into write-back mode since no other peripherals access the values written by the processor into the SRAM; as a result it is unnecessary to maintain any type of synchronization between cache and SRAM except when evicting a cache line.

5 Evaluation Methodology

Five distinct test types were utilized uniformly across all of the configurations outlined in Table 4-1. Of these, three tests (those involving memory bandwidth) were run in 2 different variations in order to gauge the effects of data sets which would not fit into the data cache. All other tests had no parameters or were run in such a manner as to simulate real word operation. Timing information for all tests was derived from a programmable timer driven from the CPU clock. Some timing data was collected in milliseconds while others used raw clock cycles depending on the characteristics of the tests.

The primary test set used in evaluation was derived from a custom RLE compression algorithm. In this test, starting from a cold cache, 4 kB of representative data was run through the compression algorithm, producing 1051 B of output. The compressed data was then decompressed into a separate buffer. Timing data was collected as the number of clock cycles required for each phase of the operation.

The second test code being used was a set of memory bandwidth tests. These included a resister to register transfer test which essentially serves as a baseline for performance, since none of the instructions in the loop body require data accesses to memory. The tests involved either reading or writing a value from a register into the data memory under evaluation. The program sequentially accessed each address in the target memory block until all values had been read or written. In order to maximise performance all transfers were 32-bits wide. In addition, to reduce loop overhead, the inner portion of the loop was unrolled 64 times and hand coded in assembly. For ease of reading and display execution time values were converted in to transfer rates in MB/s. The block sizes tested for this test set were 4 kB and 32 kB.

The final test set used was the standard Dhrystone test. This test produces a value in DMIPS relative to a nominal 1 MIPS machine, a VAX 11/780. The results from this test suite are commonly quoted by embedded processor manufacturers when quantifying the performance of their designs. Results have been converted to DMIPS/MHz in order to

quantify computational efficiency and produce results in-line with the numbers used in industry.

6 Test Results

In order to save space and simplify analysis the results of each set of tests are analyzed here in graphical form. If numeric results are desired they can be found in Appendix B.

6.1 Compression Tests

The first sets of test results to be examined are the primary metric, namely the compression tests. In these tests it is necessary for both execution times to be below 10 ms in order to provide satisfactory performance. The results can be seen Figure 6-1.

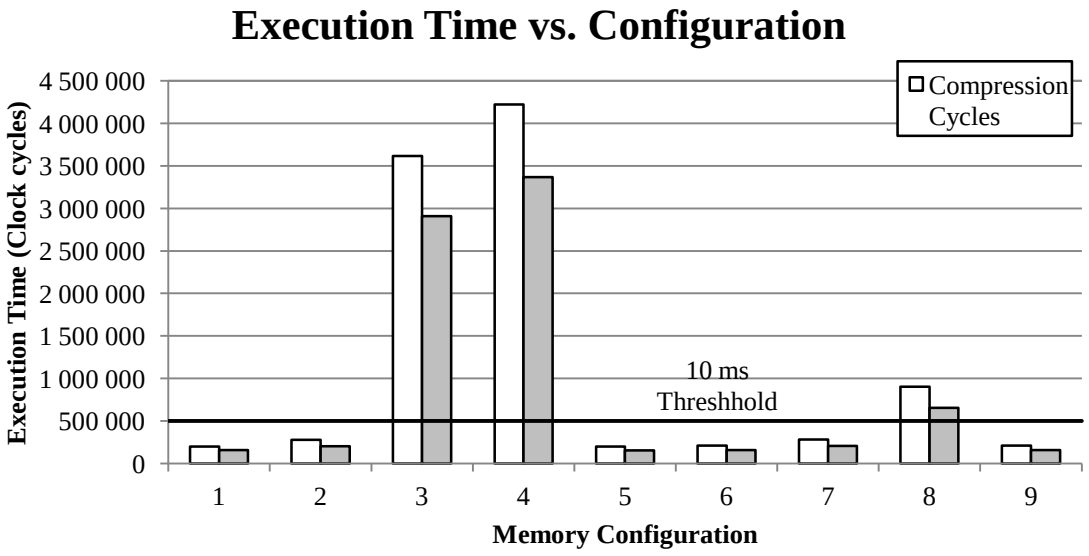

Figure 6-1: Compression test execution times

In this test, most of the configurations result in passable runtimes. The only exceptions are configurations 3, 4 and 8. Configurations 3 and 4 are the only configurations utilizing uncached SRAM as the instruction store. Configuration 8 is similar to configuration 7, with the exception of using uncached SRAM as the program stack. Aside from configurations 3 and 4, which appear to suffer from an uncached instruction store rather

than from stack related issues, this is the only configuration in which this condition exists.

6.2 Bandwidth Tests

The second set of tests were the memory bandwidth tests. There is no set performance target for these tests, but higher results will be considered better. In addition to the read and write bandwidth tests, there is also a register to register transfer test, which serves as an indication of instruction rate as well. Each memory based test was performed over a 4 kB and a 32 kB block. The results can be seen in Figure 6-2.

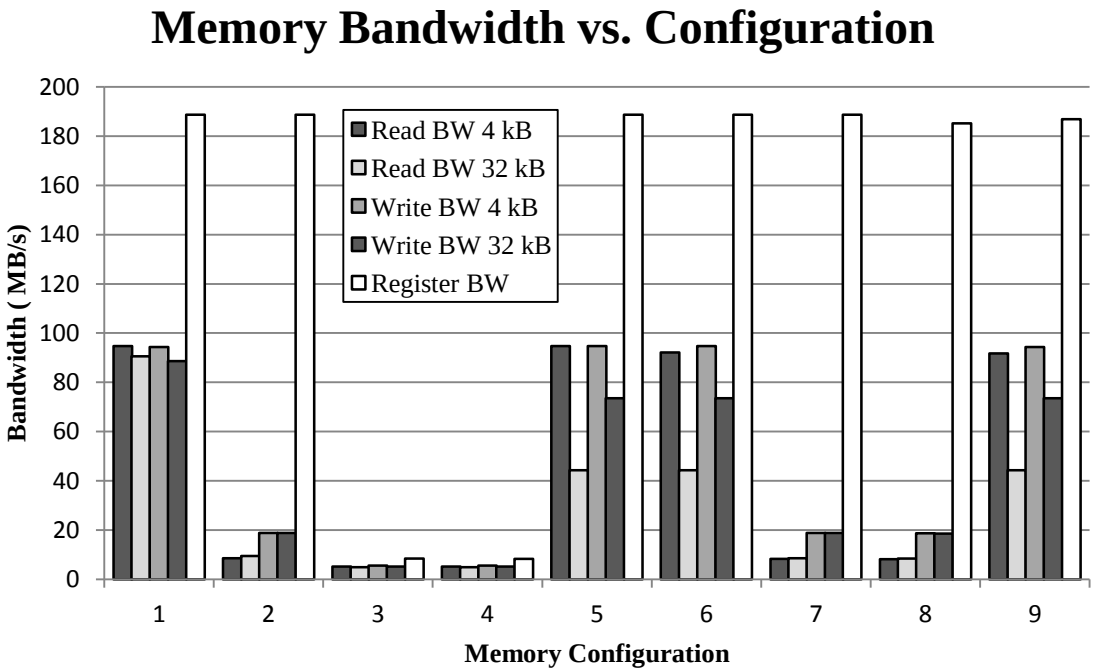

Figure 6-2: Memory bandwidths and register transfer rates

In these tests it can be seen that configurations 3 and 4 result in a reduction of register bandwidth of more than an order of magnitude. All other configurations result in similar bandwidths and instruction rates. In contrast to the previous test, configuration 8 does not result in a substantial performance decrease. This is because the test software used was contained within a single function and did not make substantial use of the program stack.

All tests which involve actual memory interaction display a maximum bandwidth half that achieved in the register tests. This is due to the use of the area-optimized version of the MicroBlaze core. One of the effects of this is to cause both load and store instructions to have a 2 cycle minimum latency rather than the 1 cycle latency possible with a using the performance oriented core. Subject to this limitation, all configurations utilizing either BRAM or cached SRAM memories display very similar results on the 4 kB tests. Since the data region for these tests fits within the data cache and the test runs are executed repeatedly in order to build up a large dataset, this is expected and indicates that the cache demonstrates that same basic latency as BRAM. When using the 32 kB block size performance drops off considerably, however it remains well above the non-cached configurations. Since the data in this case is larger than the cache, which is direct mapped, this suggests that using the cache results in significant memory bandwidth boost to the SRAM under linear access patterns. Configurations 2, 7 and 8, where instruction fetch is able to perform at full speed but data is stored in uncached SRAM demonstrate the large bandwidth differential between cached and uncached accesses.

6.3 Dhrystone Test

The Dhrystone benchmark again has no set target, but higher values are better. For reference, modern microcontroller targeted ARM cores such as the Cortex-M3 have DMIPS/MHz values on the order of 1.25 [**5**]. The results can be seen in Figure 6-3.

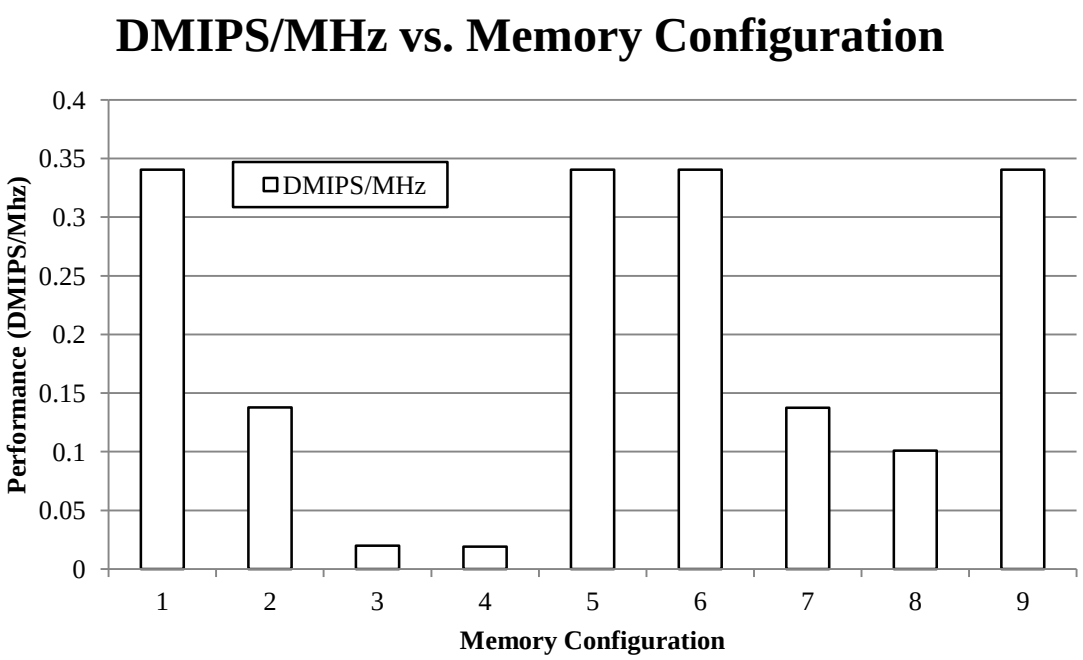

Figure 6-3: Dhrystone benchmark results

The Dhrystone results display many of the same characteristics as the previous test. In particular, the marked drop in performance displayed for cases 3 and 4 perfectly mirror the other test cases. The performance drop for configurations 7 and 8 also mirrors the bandwidth test, although the drop for case 7 is much more pronounced than for the compression test. In addition, although a performance drop does appear, the drop in configurations 7 and 8 is less severe in the Dhrystone test than in the bandwidth test. Since Dhrystone attempts to model to some degree real world code, in which not every access is a memory load or store, this is expected and suggests that most code will not see the same performance drop as demonstrated in the bandwidth test.

7 Analysis

The performance observed in the compression tests rules out the use of configurations 3, 4 and 8 since they fail to meet the minimum performance threshold for that test. Since configurations 1, 2 and 5 are intended for reference only as per section 4, this leaves configurations 6, 7 and 9 as candidates for actual usage. As there is no significant

performance differential between configurations 6 and 9; configuration 6, which makes use of both internal and external memory as well as both instruction and data caches constitutes an inefficient use of resources.

With regard to the resource cost of implementing a cache, for a 8 kB cache as used the following numbers apply. Each cache requires 4 18 kb BRAMs to form the cache memory and an additional BRAM to act as the cache's tag RAM. If utilized as part of the BRAM attached to the local memory bus, these BRAMs would provide 10 kB of memory. As a result in order to break even on BRAM resources, implementing both caches should allow at least 20 kB of BRAM to be freed. In the case of the system examined here this criteria was met based on the ability to place performance critical code into SRAM instead of BRAM.

8 Conclusions

From the analysis in the report body, it was concluded that utilizing external SRAM for data and code storage can result in very significant performance reductions, greater than one order of magnitude. It is however possible to regain much of the performance by utilizing data and instruction caches. Based on the results observed, the use of an instruction cache is critically important as all configurations executing from uncached SRAM demonstrated very significant performance drops.

The requirement for a data cache is less clear cut. The memory bandwidth test in section 6.2 indicated very significant gains though its use, however the bandwidth test constitutes a purely synthetic benchmark that does not attempt to model real world behaviour. The observed impact of the lack of a data cache in both the Compression and Dhrystone tests was not as great as in the bandwidth test. Based on the results from configuration 8, particularly in the Compression tests in section 6.1, although general program data may be usable when placed in SRAM, the program stack is more sensitive to latency and bandwidth. As a result, when using uncached SRAM for data storage it may be desirable to leave the program stack in BRAM.

9 Recommendations

Based on the analysis and conclusions in this report, it is recommended that the modules being developed using this MicroBlaze core in particular and MicroBlaze based modules in general should utilize at least an instruction cache if executing code from SRAM. This is due to the greater that one order of magnitude performance differential this configuration offers relative to the use of uncached SRAM. The use of an instruction cache restores much of the performance lost when executing from SRAM instead of BRAM.

With regard to the use of data caches, this report makes 2 recommendations, depending on the performance requirements. If the absolute highest performance is required and it is necessary to use external SRAM, then a data cache should be used. When used in combination with an instruction cache as in configurations 6 and 9, performance is nearly equal to when executing out of BRAM. If some performance degradation can be tolerated or resources for the implementation of a data cache are limited, then reasonable performance can be obtained so long as the program stack can be located in BRAM. Given the relatively small size of a typical stack (4 kB – 8kB) and the need to reserve at least some BRAM for the boot loader, which can be safely overlapped by the stack, this should be possible in almost all MicroBlaze implementations.

The end result of these recommendations is an ordered list of memory configurations from Table 4-1 by preference. Assuming that SRAM must be used and configuration 1 is not possible, configuration 9 is recommended. This configuration allows for a minimum of performance degradation while moving as much code and data as possible to SRAM. If the use of a data cache is contraindicated then configuration 7 provides the next reasonable performance option. Either of these options will provide adequate performance in most situations, although configuration 9 is preferable.

Glossary

ASIC: Application-specific Integrated Circuit, an integrated circuit that provides specialized functionality, usually designed or contracted by a single company to fulfill their specific needs and only available for their use.

ASSP: Application-specific Standard Product, an integrated circuit that provides specialized functionality, but that is designed for use by multiple customers and is available from a vendor as a standard item.

BRAM: Block RAM, internal high speed memory in an FPGA.

CoreConnect: A family of SoC busses developed by IBM and used by multiple SoC vendors.

DMIPS: Dhrystone MIPS, a measure of system performance, the performance of a system in a specific benchmark normalized against a nominal 1 MIPS machine.

FIFO: First In First Out, a type of buffer used in digital designs.

FPGA: Field Programmable Gate Array, a reprogrammable digital semiconductor device.

IP: Internet Protocol, used to refer to computer networks built with the same technologies as the broader internet.

IP: Intellectual Property, used to refer to sections of synthesisable or hard-macro digital logic developed for a specific task. May be internally developed or licensed from a third party.

ISA: Instruction Set Architecture, the set of instructions, as well has the memory model and other programmer visible features of a microprocessor.

LMB: Local Memory Bus a Xilinx specific simple on-chip bus for accessing high speed memory.

MSR: Machine Status Register, a special purpose register containing control and status bits for the processor.

MIPS: Million Instructions per Second, a rate of instruction execution

OPB: On-chip Peripheral Bus, a peripheral bus for SoCs, part of the IBM CoreConnect family.

PLB: Processor Local Bus, a high-speed multi-master bus designed for SoCs, part of the IBM CoreConnect family.

RISC: Reduced Instruction Set Computer, an approach to Instruction Set Design that emphasises simplicity and orthogonality, relying on these aspects to scale clock speeds and performance.

SoC: System on a Chip, an integrated computer system combining may traditionally separate elements onto a single chip.

SRAM: Static Random Access Memory, semiconductor memory that does not require a refresh cycle.

XCL: Xilinx Cache Link, a FIFO based memory interface used by Xilinx for processor caches.

References

[1] Harris Corporation. (2010, Aug.) Broadcast Overview / Backgrounder. [Online]. http://www.broadcast.harris.com/broadcastoverview/background.asp

[2] Ken Chapman. (2007, September) Picoblaze FAQ. [Online]. http://forums.xilinx.com/t5/PicoBlaze/PicoBlaze-FAQ-Is-there-a-C-Compiler/m-p/580

[3] Xilinx Inc. (2009, October) MicroBlaze Processor Reference Guide. [Online]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf

[4] Reinhold P. Weicker, "Dhrystone Benchmakr: Rationale for Version 2 and Measurement Rules," *ACM SIGPLAN Notices*, vol. 23, no. 8, pp. 49-62, August 1988.

[5] ARM Inc. (2010, September) Cortex-M3 Processor. [Online]. http://www.arm.com/products/processors/cortex-m/cortex-m3.php

[6] Integrated Silicon Solution, Inc. (2010, July) IS61V25616ALL/ALS, IS61V25616BLL/BLS, IS64V25616BLL/BLS,. [Online]. http://www.issi.com/pdf/61WV25616.pdf

[7] Numonyx B.V. (2007, December) Numonyx Embedded Flash Memory (J3 v D). [Online]. http://numonyx.com/Documents/Datasheets/316577_J3D_Monolithic_DS.pdf

Appendix A External Memory Parameters

Table A-1: External memory size and timing parameters

Parameter	IS61WV25616BLL [6]	RC28F640J3D75 [7]
Width (bits)	16	16
Depth (bits)	262 144	4 192 304
Size	512 kB	8 MB
Read cycle time	10 ns	75 ns
Write cycle time	10 ns	75 ns

Appendix B Results Tables

The numeric data collected from the tests is presented here.

Table B-2: Numeric compression test results

Configuration	Code	Data	Stack	Compression Cycles	Decompression Cycles
1	BRAM	BRAM	BRAM	199 781	156 709
2	BRAM	SRAM	BRAM	277 875	204 312
3	SRAM	SRAM	BRAM	3 617 487	2 910 292
4	SRAM	SRAM	SRAM	4 222 514	3 368 401
5	BRAM	SRAM (Cached)	BRAM	200 072	156 304
6	SRAM (Cached)	SRAM (Cached)	BRAM	212 777	157 047
7	SRAM (Cached)	SRAM	BRAM	281 084	205 622
8	SRAM (Cached)	SRAM	SRAM	902 493	654 970
9	SRAM (Cached)	SRAM (Cached)	SRAM (Cached)	212 215	157 140

Table B-3: Numeric bandwidth test data, 4kB block and Register

Configuration	Code	Data	Stack	Read BW 4 kB (MB/s)	Write BW 4 kB (MB/s)	Register BW (MB/s)
1	BRAM	BRAM	BRAM	94.7	94.3	188.6
2	BRAM	SRAM	BRAM	8.5	18.8	188.6
3	SRAM	SRAM	BRAM	5.2	5.5	8.4
4	SRAM	SRAM	SRAM	5.1	5.5	8.3
5	BRAM	SRAM (Cached)	BRAM	94.7	94.7	188.6
6	SRAM (Cached)	SRAM (Cached)	BRAM	92.1	94.7	188.6
7	SRAM (Cached)	SRAM	BRAM	8.3	18.8	188.6
8	SRAM (Cached)	SRAM	SRAM	8.2	18.6	185.1
9	SRAM (Cached)	SRAM (Cached)	SRAM (Cached)	91.7	94.3	186.9

Table B-4: Numeric bandwidth test data, 32 kB block

Configuration	Code	Data	Stack	Read BW 32 kB (MB/s)	Write BW 32 kB (MB/s)
1	BRAM	BRAM	BRAM	90.6	88.6
2	BRAM	SRAM	BRAM	9.4	18.8
3	SRAM	SRAM	BRAM	4.9	5.2
4	SRAM	SRAM	SRAM	4.9	5.2
5	BRAM	SRAM (Cached)	BRAM	44.3	73.6
6	SRAM (Cached)	SRAM (Cached)	BRAM	44.3	73.6
7	SRAM (Cached)	SRAM	BRAM	8.6	18.8
8	SRAM (Cached)	SRAM	SRAM	8.5	18.6
9	SRAM (Cached)	SRAM (Cached)	SRAM (Cached)	44.3	73.6

Table B-5: Numeric Dhrystone benchmark results.

Configuration	Code	Data	Stack	DMIPS	DMIPS/MHz
1	BRAM	BRAM	BRAM	17.0	0.34
2	BRAM	SRAM	BRAM	6.8	0.14
3	SRAM	SRAM	BRAM	1.0	0.02
4	SRAM	SRAM	SRAM	0.9	0.02
5	BRAM	SRAM (Cached)	BRAM	17.0	0.34
6	SRAM (Cached)	SRAM (Cached)	BRAM	17.0	0.34
7	SRAM (Cached)	SRAM	BRAM	6.9	0.14
8	SRAM (Cached)	SRAM	SRAM	5.0	0.10
9	SRAM (Cached)	SRAM (Cached)	SRAM (Cached)	17.0	0.34