Certificate

This is to certify that the work contained in the B.Tech. Project report titled "**Issues with designing a dual core processor with a shared L2 cache on a Xilinx FPGA board**". submitted by V Bhanu Chandra (Y3383) and Varun Sharma (Y3393) has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Mainak Chaudhuri 03/05/2007

Dr. Mainak Chaudhuri
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
May, 2007

Issues with designing a dual core processor with a shared L2 cache on a Xilinx FPGA board

V Bhanu Chandra (Y3383), Varun Sharma (Y3393)
{vbhanu,varunsh}@cse.iitk.ac.in
Project Supervisors: Dr. Mainak Chaudhuri, Dr. Rajat Moona
{mainakc,moona}@cse.iitk.ac.in
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Abstract

Dual core processors provide better performance than single core processors because of the support of the thread level parallelism. We discuss the issues with designing a dual core processor with a private L1 cache and shared L2 cache on an FPGA board.

1. Problem Statement

We discuss the issues involved with designing a MicroBlaze based dual core processor on Xilinx FPGA board. The first level cache is implemented and is private to each core. So, we assess the problems involved with designing the second level cache which would be shared by the two processors.

2. Introduction

As observed by the Moore's law the transistor density is very rapidly increasing. The performance that can be extracted from a single core has neared it's threshold. So, in this situation people are looking at multi-core processor designs to improve the overall performance of the system. A multi-core microprocessor is one which combines two or more independent processors into a single package, often a single integrated circuit (IC). A dual-core device contains only two independent microprocessors. Very often the applications exhibit thread level parallelism (TLP) and multi-core microprocessors can enhance the overall performance of the system by exploiting the TLP without including multiple microprocessors in separate physical packages. This form of TLP is often known as chip-level multiprocessing, or CMP.

International Business Machines(IBM)'s POWER4 was the first dual core module processor that came out in 2000. The most recent advancement in this field is Intel's Centrino Duo Mobile Technology [2] which is being used in Centrino notebook.

3. Simulation Environment Tools

3.1. Xilinx Platform Studio(XPS) and Embedded Development Kit(EDK)

EDK [1] is an integrated software solution for designing embedded processing systems and implementing on a Xilinx FPGA device. The components of the Xilinx EDK are:

- Hardware (Intellectual Property) for the Xilinx Embedded Processors and their peripherals.

- Drivers, Libraries and a Micro Kernel for Embedded Software Development.
- Software Development Kit(SDK), Eclipse based IDE.
- GNU compiler and debugger for C development for MicroBlaze and PowerPC.

XPS provides an integrated environment for creating software and hardware specification flows for embedded processor systems based on MicroBlaze and PowerPC processors. It provides a graphical system editor for connection of processors, peripherals and buses. The salient features of XPS are:

- Ability to add cores, edit core parameters, and make bus and signal connections to generate an MHS(Microprocessor Hardware Specification) file.
- Ability to generate and modify the MSS(Microprocessor Software Specification) file.
- Support for the following tools:
  - Xilinx SDK
  - Base System Builder(BSB) Wizard
  - The create and import IP wizard
  - Platform generator
  - Library Generator
  - Xilinx Microprocessor Debugger(XMD)
  - Platform Specification Utility
- Ability to generate and view system design report.
- Process and tool flow dependency management.

3.2. Virtex 4 FPGA board

We used the Virtex 4 FPGA board and the ML403 Evaluation Platform[4] for implementing our designs. Some of the salient features of the board are:

- 64 MB DDR SDRAM, 32 bit interface running up to 266-MHz data rate.
- General purpose LEDs and push buttons.
- RS-232 serial port.
- One 4Kb IIC EEPROM.
- PS/2 mouse and keyboard connectors.
- JTAG configuration port for use with Parallel Cable III and Parallel Cable IV cable.
- System ACE and Compact Flash Connector.

3.3. Integrated Software Environment(ISE)

ISE is a design software suite. ISE can be used to design in different source types such as HDL, Schematic Design files, State Machines and IP cores. HDL sources may be synthesized using Xilinx Synthesis Technology(XST). The Project Navigator interface provides a visual design manager for the entire process.

3.4. Hardware Used

We are using a Virtex 4 ML403 Evaluation Platform. The other hardware components include:

- RS232 cable for the I/O. This is connected to the RS232 port on the board on one end and serial port on the other end.
- JTAG adapter for hardware debugging and also downloading the bitstream on to the board.

4. Design

4.1. Design of the processor

The processor that we have is a highly configurable 32-bit MicroBlaze[3] softcore. MicroBlaze processor uses Big endian bit-reversed format to represent the data. The L1 cache is implemented, it is private to the processor. If L1 cache is needed then it can simply be enabled. The size of the L1 cache is configurable, while the minimum size of the cache is 2kB.

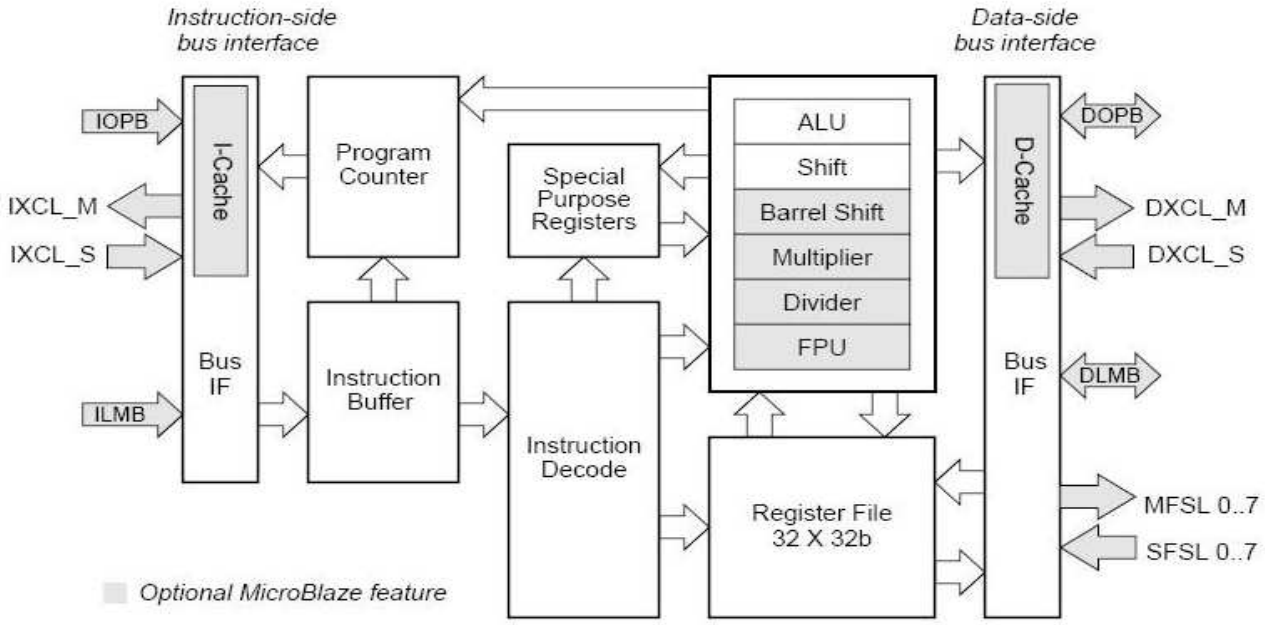


Figure 1 MicroBlaze Architecture

The processor is a master on an OPB [5] (on-chip peripheral bus) and also on the LMB (local memory bus) each of which addresses an address range. When the OPB bus is added in the system the OPB arbiter is generated by the XPS. When the processor does not find certain data in the first level cache (L1 cache is private to the processor), a complete request for that data is sent on to the OPB bus which has access to that address range. The local memory bus is connected to the local memory which is a small memory made of BRAM blocks. It is a high speed bus and BRAM memory is also very efficient. Hence, this memory can be used to host small programs which would improve the performance of the program.

4.2. System Design

The fact that OPB is a multi master bus allows one to connect both the cores to the bus and then also have the cache controller on the same bus. Optionally one can enable the L1 cache in the processor, this would be private to the processor. If the cache is enabled in the processor then the L2 cache mechanism would be the second level cache else it would act as the first level

cache. The L2 cache could be implemented in the BRAM blocks and the controller is also a peripheral which is connected to the OPB bus.

The processors would generate request for certain action at a certain address location by setting the signals on the OPB. The respective device on the OPB bus would identify the request being generated and process it as needed. When the request is for certain data, the L1 cache controller, private to each processor, checks whether the data is in the L1 cache else a complete request is sent on to the OPB bus. This could then be observed by the L2 cache controller and then if that data is present in the L2 cache then that data is returned else the data should be fetched from the DDR SDRAM and the data be sent on the OPB bus. This data is then sent to the processor, caching it in L1 on the way. The same data should be cached in the L2 cache by the cache controller.

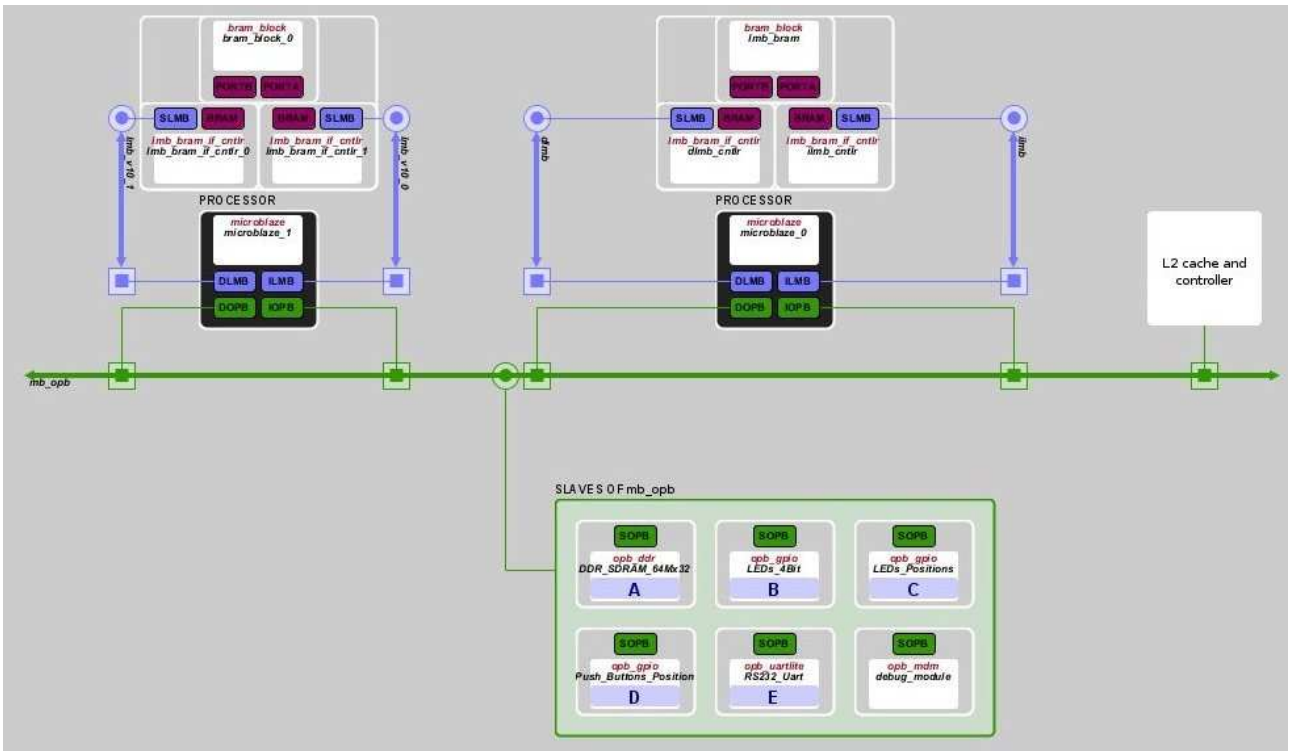


Figure 2 Final Design

The processor puts the address from which the data has to be fetched on the Bus2IP_Addr port and at every clock cycle this is examined and if the appropriate Read signals are set and the data is available in the cache then it is put on the bus. The port of the controller which has the data is IP2Bus_Data. If the data is not available in the cache then the data is fetched from the DDR SDRAM by setting the DDRRAM_IP2Bus_Addr to the address from which the data is to be fetched. The data can be obtained by reading the port DDRRAM_Bus2IP_Data.

5. Implementation

We have created a design which has two processors, each with it's own local memory and two separate programs each of which prints a separate string. Each program is loaded into the local memory corresponding to one of the cores and is executed on the corresponding core. The expected and the actual output is a jumbled string.

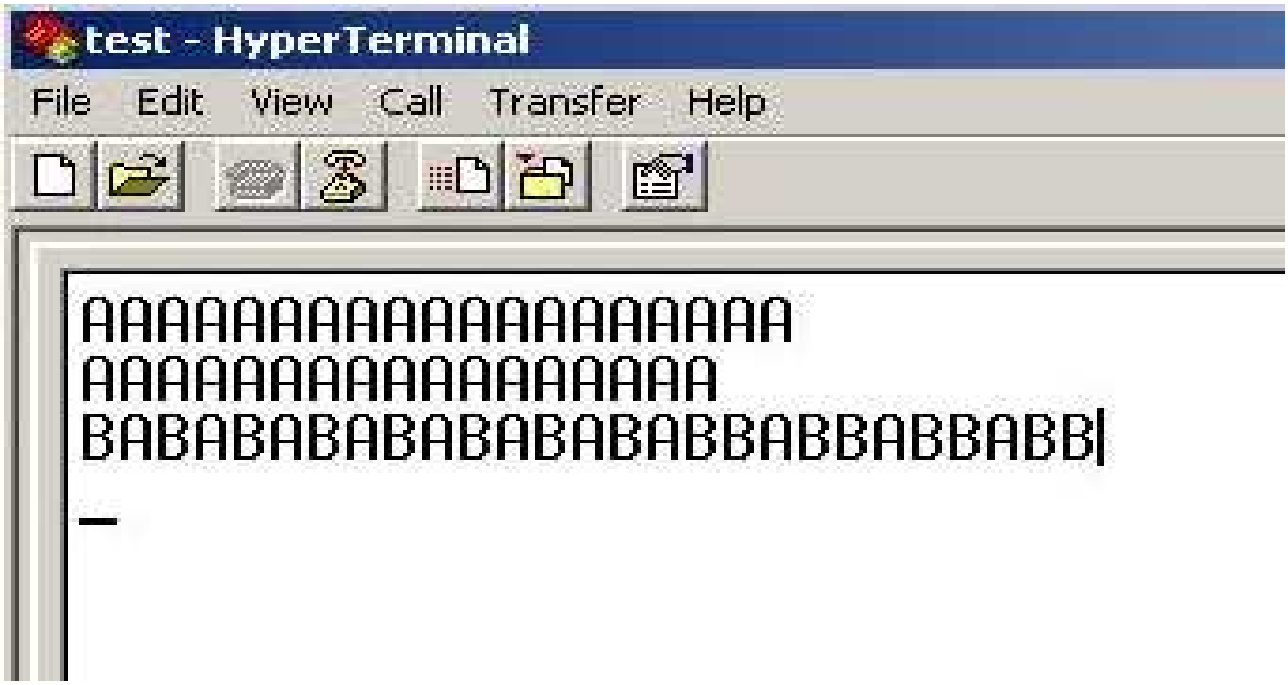


Figure 1 MicroBlaze Architecture

Due to a bug in the 8.2i version of the software it was not possible to create a peripheral which could be coded in verilog. We have found a workaroud for this, the workaround requires one to change the MPD and PAO files generated by the software.

5.1. Cache Implementation

Focus of our work so far has been to implement a second level cache for a single processor. The idea is to implement the cache using BRAM blocks and the controller written in verilog. The IPIC (intellectual property interconnect) makes it simpler for one to address the signals that are needed to access data from BRAM blocks and also the information on OPB is more easily addressable. The idea is to have separate BRAM blocks for cache and for the tags.

The address sent onto the OPB by the processor can be received by the cache controller and then sent to the BRAM to see if the corresponding data is available. If it is then the data is sent to the processor else the data is obtained from the DDR memory and written to the cache before it is sent to the processor. The data read from DDR SDRAM is read in units of eight words and the cache is write through.

The cache that we have discussed and implemented has the following structure:

1. Six tag bits are associated with each cache line.
2. Ten bits have been allocated for the index.
3. Five bits have been taken for the block offset.

First 11 bits are 0	6 tag bits	10 index bits	5 block offset bits

Note that with the above structure we would need 32KB of space for cache and 2KB for the tag, half of which is for the state bits. The tags are stored in a BRAM block and the cache data is stored in 16 BRAM blocks. Hence we take 17 BRAM blocks. In the tag BRAM block the first line is assigned to the tag and the second one is given to the corresponding state bits. This continues, so half of the BRAM block is taken by the tags while the other half is assigned to the state bits.

6. Issues with the design

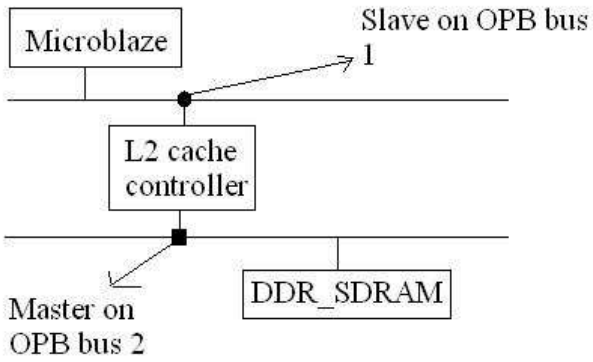
As soon as our cache controller was compiled, we were stuck with a problem in our initial design. The address generated by the processor for getting the data belonged to the DDRRAM. Since the cache controller was also a peripheral attached to the same bus, it had a different address range. Hence it cannot capture the signals of the DDRRAM. Also we cannot have an overlap of addresses for the cache controller and the DDRRAM. So there was no way, we could run the system with a fully functioning L2 cache.

We had overlooked this problem in it's initial stages hoping that we could have different disjoint address spaces for the cache controller and the DDRRAM. The idea was to configure the processor to generate addresses in the range of the controller and then translate the address that comes to the controller before sending it to the DDRRAM. Later we discovered that it is not possible to change the whole set of addresses generated by the processor since the processor would always have instructions which would require it to address the absolute address and not just the offset.

7. Alternate Designs

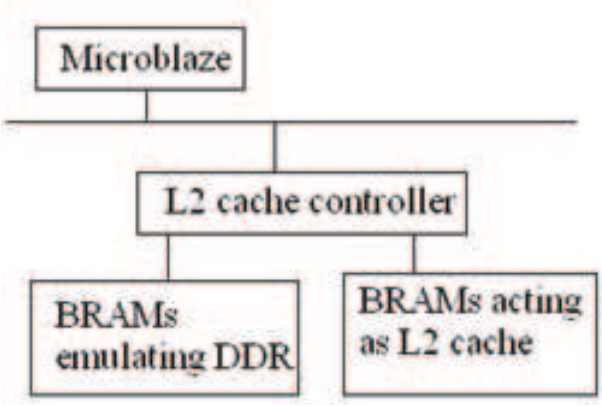
We discussed the following designs after we realised that the earlier design does not work:

1. This design had 2 OPB buses. The Microblaze processor and other peripherals(except DDRRAM) were attached on one of the bus and the DDRRAM was attached on the second OPB bus. The L2 cache controller was slave on the first bus and master on the second. The idea was that all the signals that are directed towards the DDRRAM will have to pass through the cache controller and thus here the cache can come into the picture if it has the requested data.



Design with multiple OPB buses

2. In the next design, we thought of removing the DDRRAM from the system. Instead we can use the BRAMs in the L2 cache controller to emulate the DDRRAM. The existing BRAMs attached to the L2 cache controller can be broken into two parts. Some acting as a DDRRAM and the others acting as L2 cache. The access to the first part can be delayed using delay signals while the other part can be fast as BRAMs so that it looks like a faster memory, which a cache should be.



Design with DDR emulated in BRAM

Each of the above designs met with problems because of which they cannot be implemented either. The first design again heads into the problem of two different peripherals having the same address space while the second one does not work due to the limited number of BRAM blocks on the system. The system has a total of 36 BRAM blocks, 17 of which are being used by the cache, assuming that we use a BRAM block each for first level cache for each of the processors we would be left with 17 BRAM blocks to emulate DDRRAM. Thus due to the limited number of BRAM blocks this design also has to be discarded.

8. Suggested Solutions

As we discussed these problems we came to believe that the following approaches would help solve the problems that have been mentioned above:

1. The first design change that was proposed to overcome the problems address above are to use the user defined processor. The advantage of using our own processor is that we can generate the addresses as we require. So we can add a translator in the processor and the addresses which refer to DDR memory will be appropiately mapped to the address range in L2 cache controller. Then we can even do away with 2 buses and all the IPs would be attached to the same bus.

2. The second design proposal is to add the functionality of a standard IBM bridge protocol in the L2 cache controller connected to both the buses. It will be slave on the bus that is connected the processor and master on the second bus. The bus protocol requires only an address range that has to be mapped to the second bus. It has no address range of its own. If the L2 cache controller has this protocol built in, it wont overlap with DDR. It can respond to the data request if the data is present in cache in a tranparent manner which is the functionality of the cache.

9. Conclusion

The implementation of an L2 cache is not possible in the current scenario because of software restrictions. The L2 cache and the DDR cannot have the same address range because both are IP attached on the bus and their address range cannot overlap. Therefore a need for more flexible system is required. This may be done using a user defined processor which generates addresses for L2 cache or it can be done using OPB2OPB bridge. Using a bridge will help in communicating across 2 buses. The bridge can also act like a L2 cache controller which is connected to BRAMs. It can capture the signals and reply on its own based on whether it has the data or not. And if it does not have the data, it can request the DDR to furnish the data.

For testing the correctness of the current user logic of the L2 cache controller, either of the designs needs to be implemented. Various issues have to be kept in mind while designing some of which are:

1. Failure of the software to make proper entries into the system.

2. Behavior of the processor or the bridge should be dependent on the OPB bus protocol.

3. While using the bridge, the cache controller should behave in a transparent manner.

We have listed a number of issues that we faced while designing the system and the methods to overcome them. These issues should be considered while implementing a fully functional dual core system with a shared L2 cache.

References

[1] Embedded System Tools Reference Manual
Webreference http://www.xilinx.com/ise/embedded/est_rm.pdf

[2] Intel Technology Journal, Intel Centrino Duo Mobile Technology
Volume 10, Issue 02, May 15,2006
Webreference http://www.intel.com/technology/itj/2006/volume10issue02/art01_intro_to_core_duo/p01_abstract.htm

[3] MicroBlaze Processor Reference Guide
Webreference http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf

[4] ML40x EDK Processor Reference Design
Webreference http://www.xilinx.com/bvdocs/userguides/ug082.pdf

[5] OPB IPIF Architecture
Webreference http://www.xilinx.com/ipcenter/catalog/logicore/docs/opb_ipif.pdf