This is the accepted version of a paper presented at 2nd Swedish Workshop on Multi-core Computing.

Citation for the original published paper:

Bao, Y., Brorsson, M. (2009)
An Implementation of Cache-Coherence for the Nios II™ Soft-core processor.
In:

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-109653
An Implementation of Cache-Coherence for the Nios II™ Soft-core Processor

Yan Bao and Mats Brorsson
KTH Royal Institute of Technology
School of Information and Communication Technology, and
Swedish Institute of Computer Science
E-mail: bao@kth.se, matsbror@kth.se

Abstract
Soft-core programmable processors mapped onto field-programmable gate arrays (FPGA) can be considered as equivalents to a microcontroller. They combine central processing units (CPUs), caches, memories, and peripherals on a single chip. Soft-core processors represent an increasingly common embedded software implementation option. Modern FPGA soft-cores are parameterized to support application-specific customization. However, these softcore processors are designed to be used in uniprocessor system, not for multiprocessor system. This project describes an implementation to solve the cache coherence problem in an ALTERA Nios II soft-core multiprocessor system.

1. Introduction
Nios II is a configurable and high-performance soft-core processor intended for FPGA implementation on Altera devices. It is easy to define an MPSoC using the Altera tool chain, but the Nios II core has no support for hardware cache coherence and therefore it is difficult to use modern parallel programming models such as OpenMP [2] and Cilk++ [1, 4]. In this paper we describe an experimental implementation of a cache coherence implementation for a multicore processor using the Nios II soft-core in an FPGA configuration. We have used the Altera DE2 development and education board, using the Nios II soft-core processor for which we have obtained a clear text licens. The DE2 board has a Cyclone II EP3C35 device with 33 216 logic elements. A soft-core design with Altera devices uses a tool called SoPC (System-on-Programmable Chip) builder, where cores are connected with memories and I/O devices. The SoPC builder generates HDL-files (VHDL or Verilog) for the design and these files are used in projects. We have worked with the VHDL generated by the Altera configuration tool and added the necessary modifications for making a dual-core system cache coherent and therefore being able to support modern parallel programming models with high performance.

The FPGA technology is flexible for many applications in all industries. With its rapid developing, current FPGA devices can be programmed with high-performance 32-bit soft-core processors. Comparing with ASIC and digital signal processing (DSP), FPGA has many advantages:

Efficiency: An FPGA may have a higher processing speed than a DSP since it can run operations defined in hardware in parallel, process more jobs per clock cycle and without execution of software programs as DSP-based solutions.

Short design-cycle: Flexibility is the most important feature of FPGA:s. The Latest FPGA:s provide platform solutions and large vendors, like XILINX and ALTERA, are also providing development tools which are very easy for engineers to use for customization. The use of abstract hardware description languages (HDL), such as VHDL and Verilog, makes FPGA:s easy to upgrade.

Low costs: The non-recurring engineering (NRE) costs are quite high for ASIC-based solutions, because the production is usually small comparing to the engineering effort. If the production of an ASIC design is not widely used like in the case of a mobile phone, then each unit will share a large fraction of the design cost. FPGA designs do not have these shortcomings.

Currently, the system with FPGA-based solutions combines the advantages of both ASICs and DSPs. With the hardware control platform of FPGAs, designers could easily develop the hardware to fulfill the requirements from customers and industries at relative low cost and high efficiency. The integration of soft-core processors makes an FPGA employ typical processor capabilities. The easy upgrading feature makes FPGA solutions advantageously used in the systems that already exist.
According to MOORE’s law, the performance of a chip and the number of transistors integrated on a single chip will grow dramatically over time. A Multicore system indicates that there are more than two CPUs integrated in a single chip system. With parallel execution no the processors the performance of the whole system on the chip increases at a relative low cost level since it is usually much cheaper to integrate multiple processors than to design a single processor system with high performance. This, however, requires that software can be developed that scales up in performance when processors are added to the system.

However, the design of a multiprocessor system is more complex than for a single-core system. In the past, system designs were based on single-processor or coprocessor architectures, today new system designs are mainly based on heterogeneous MPSoC architectures. New platform brings new design flows and methodologies. The management of processors and memory in a multiprocessor system is equally important with the management of memory in a single-processor system.

The original purpose of this project has been to gain in-depth knowledge of the implementation aspects of multiprocessor system. A challenge has been to come up with a design which is both independent of broadcast communication mediums and in addition scalable to more than two processor nodes. The design has been discussed with Altera Nios II chief designers. The architecture is designed to scale beyond dual-core designs but this will be evaluated in subsequent projects. A secondary objective, which seems more and more feasible, is that this could form the basis in an FPGA computer architecture research platform when we can use larger devices and we will strive for a design where we can extend the cache coherence functionality beyond the chip level to create larger systems.

2. Design

To implement a high performance cache coherency multi-processor system with Nios II core, the issues when searching proper solutions must be considered. The Nios II multi-core system is not designed for cache coherency, which is the largest challenge. Especially, the Avalon Switch Fabric in the system is a point-to-point connection, which makes the traditional snooping mechanisms that rely on the broadcast communication of a bus impossible to use. Altera suggests a software controlled cache coherence model where cores can invalidate specific cache lines through special instructions. However, this requires that cache contents is written back to the memory greedily (and maybe unnecessarily) since there is no way to communicate modified cache contents to other cores since the Nios II core uses write-back caches.

A previously proposed solution for cache coherence for the Nios II core utilized the write-through caches of Nios I to keep track of modified cache contents and then an interrupt-based mechanism to invalidate other processors’ caches [3]. This is not acceptable in high-performance solutions and in addition, the Nios II core uses write-back caches so modifications are not visible outside the core. Therefore, to reconfigure the Nios II core internally is the only viable way to avoid interrupting the processor. A special license was offered by Altera to modify the source code, since Nios II core is not an open source product.

The goal of the design is to implement the cache coherency protocol with minimum increments to the original Nios II system at hardware level. To achieve this, careful examination of the VHDL code of the Nios II core, Avalon Switch Fabric, and the memory controllers were required. We searched for all blocks which could affect cache coherency and which can be utilized for cache coherency. The most difficult part was to understand the design of Nios II core since we had no documentation or access to support under our license agreement.

A multiple soft-core processor system can easily be implemented in a single programmable logic device. These modern devices have adequate resources to implement complex systems with cache and memory controllers. The SoPC Builder software provided by Altera allows users to easily implement the systems they need. It also generates VHDL code for most of the modules, which make it possible that users could modify these modules as they want. Since the device on our development board is quite small, our design only contains two cores, but the cache coherency interface is designed to be scalable to more cores.

A simplified architecture of the original Nios II multi-processor system is shown in Figure 1. This is a symmetric multiprocessor system, which means the cores are all the same, including separate instruction and data caches and master ports connecting to peripheral devices. Master ports and slave ports are connected through the Avalon Switch Fabric, the switch fabric could be seen as a pipeline starting at a master port and ending at the requested slave port. The SDRAM cannot be directly connected to the slave port, therefore an SDRAM controller is thus used to drive the SDRAM through memory-mapped ports. All parts except the SDRAM devices in Figure 1 are realized inside the FPGA device.

In a standard dual-core design, each core is connected to the SDRAM controller through a data master port (M) and the SDRAM controller has a corresponding buffered data slave port (S). The slave port includes an arbiter to control which of two simultaneous core accesses gets the priority.

The architecture of the multicore system with cache coherency interface is shown in Figure 2. All shaded mod-

\footnote{Nios I is a predecessor to the Nios II core.}
ules are new components added for the cache coherency interface. The system consists of enhanced data cache controllers in the cores and a directory controlling the cache coherence protocol inside the SDRAM controller. The directory has a limited number of entries and when an entry has to be removed because of an address mapping conflict or limited capacity, the corresponding cache lines are invalidated from the cores first.

On the processor side, the Data Cache Controller (DCC) is the main part. It connects to the data cache and monitors the requests to the data cache. According to different requests to the data cache, extra actions should be taken depending on the state a particular cache line is in. The DCC sends action requests to the SDRAM controller through its own master port (DCC_M), and receives information from the directory controller and other processors by its own slaver port (DCC_S). On the SDRAM controller side, all accesses must pass through the directory controller before arriving to the SDRAM. The directory controller is the main part of the cache coherency protocol in the system. It coordinates requests from different processors and ensures that the system should write atomically and avoid dead lock. The directory controller uses individual master and slave ports (Dr_M and Dr_S) to send and receive data from the DCC:s of processors. When the directory controller receives a request, corresponding operations will be executed after checking the data in directory. The byte data field of directory is shown in Table 1.
The data cache controller and the directory controller are the central parts on their respective side of the cache coherency interface. They operate with their own protocols. Actually, in data cache controller, there are two protocols: One protocol is for the occasion that the cache controller monitors a request to the data cache, the other protocol is for the occasion that requests are received from the DCC_S. The protocols are shown in Figure 3, Figure 4 and Figure 5, respectively. Because of the limitation device on the DE2 board, the directory does not have large enough space to contain the information of all SDRAM lines; instead it is implemented to store information of all cache lines of CPUs. That means all valid lines in data caches of all processors must have the same tag. The state machines of directory controller are shown in Figure 6.

The entire protocol of the cache coherency protocol is a directory based, invalidation-update hybrid, MSI protocol. As mentioned before, the data cache in the Nios II soft-core is a write-back cache, it has three states: Modified, Shared and Invalid. In the directory, there are only two states in each line: exclusive and shared. There is no need for a special uncached state in the directory since if all presence bits are false or there is no hit in tag field, this indicates that the requesting tag in the line is at state Uncached. The directory stores all cached SDRAM lines’ information. Each request to SDRAM should check the information of corresponding line in directory and decide whether extra operations need to be executed before it accesses SDRAM.

To maintain writing atomicity, the protocol allows only one processor to access SDRAM at the same time. If one request is granted by directory controller, other requests will be refused until the request responding is completed. Since there are two slave ports in SDRAM controller, if one slave port grants a request, it should hold the other one until it has finished responding.

To avoid dead-lock, since one request from a CPU could generate extra actions to other CPUs, only the granted processor can hold its slave ports and refuse all requests from SDRAM and other processors. This means that only one processor could hold its slave port at the same time in the system. This could avoid two processors requesting to each other and waiting forever.

To reduce latency, the protocol is designed with both invalidation and update actions. When a processor executes a store operation to an invalid cache line, and corresponding state in directory is exclusive, update action has less latency. To invalidate, the data owner processor should change its state to invalid and writeback the whole cache line data to SDRAM, then SDRAM send the data to requesting processor, after receives the whole line data, requesting CPU will be allowed to store new data. If a cache has 8 offsets (one offset has 32-bits data), the whole transaction needs at least 16 clock cycles to finish. To update, the requesting proces-
Figure 5. Protocol of the directory controller.
Figure 6. State machines of the directory controller.
Figure 4. Protocol of the data cache controller request receiver.

<table>
<thead>
<tr>
<th></th>
<th>Original design</th>
<th>New design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total logic elements</td>
<td>5 642</td>
<td>7 282</td>
</tr>
<tr>
<td>Memory bits used</td>
<td>230 656</td>
<td>243 072</td>
</tr>
<tr>
<td>Max clock frequency</td>
<td>187.34 MHz</td>
<td>72.35 MHz</td>
</tr>
</tbody>
</table>

Table 2. Key number differences between the original design and our cache coherent design.

The design has been implemented, simulated and tested, in VHDL in an decrypted version of the Nios II processor graciously provided by Altera. We are currently targeting the Cyclone II EP2C35 device for the DE2 development kit board. It is difficult to compare differences in logic element and memory usage for FPGA devices as the compiler can optimize differently depending on the design. Table 2 shows some key numbers of a dual-core system with 8 kByte data caches and a 1024 entry directory. The results nevertheless indicate that the memory overhead is negligible in this device. The maximum frequency has decreased much comparing with the original design and the reasons for this needs further investigation.

3. Implementation aspects

The design has been implemented, simulated and tested, in VHDL in an decrypted version of the Nios II processor graciously provided by Altera. We are currently targeting the Cyclone II EP2C35 device for the DE2 development kit board. It is difficult to compare differences in logic element and memory usage for FPGA devices as the compiler can optimize differently depending on the design. Table 2 shows some key numbers of a dual-core system with 8 kByte data caches and a 1024 entry directory. The results nevertheless indicate that the memory overhead is negligible in this device. The maximum frequency has decreased much comparing with the original design and the reasons for this needs further investigation.

4. Conclusions

We have designed and implemented a cache coherence functionality for the Nios II soft-core processor from Altera. As far as the authors and key processor designers of Altera knows, this is the first attempt to provide this functionality for the Nios II processor. Compared to a dual-core design without cache coherence, our design has negligible overhead in memory bits and logic elements. And the design is extendable to the system with more than two cores. Our future work will be to implement it on a large FPGA board and add more functional performance.

Acknowledgments

This work would not have been possible without the generous support from Altera in form of the clear-text Nios II license, licenses for the Quartus tool chain, the Nios II IDE environment and development boards.
References


