

# Supporting the Linux Operating System on the MOLEN Processor Prototype

Filipa Duarte, Bas Breijer and Stephan Wong

Computer Engineering  
Delft University of Technology

F.Duarte@ce.et.tudelft.nl, Bas@zeelandnet.nl, J.S.S.M.Wong@ewi.tudelft.nl

*Abstract*— In order to support an operating system on the MOLEN processor prototype, there is a need to increase the size of the memory the prototype can support. In this paper, we present an OCM-based shared DDR memory controller that can achieve an improvement from 6% (best-case) to 13% (worst-case) over a PLB based design. Subsequently, we explain the steps needed to execute Linux on the previously designed system. We present the (thus far undocumented) limitations such a system can impose on the Linux operating system. Due to these limitations, we also present the new design of the MOLEN prototype that can support Linux operating system.

## I. INTRODUCTION

The MOLEN polymorphic processor [13] is a custom computing machine based on the co-processor architectural paradigm. The two main components in the MOLEN machine organization (depicted in Figure 1) are the core processor, which is a general-purpose processor (GPP), and the reconfigurable processor (RP). The ARBITER performs a partial decoding on the instructions in order to determine where they should be issued. Instructions implemented in fixed hardware are issued to the GPP and instructions for custom execution are redirected to the RP. Data transfers from/to the main memory are handled by the “Data Load/Store”-unit. The “Data Memory MUX/DEMUX”-unit is responsible for distributing data between either the reconfigurable or the core processor. Pieces of application code can be implemented on the RP in order to speed up the overall execution of the application. A clear distinction exists between code that is executed on the RP and code that is executed on the GPP. Data must be transferred across the boundaries in order for the overall application code to be meaningful. Such data includes predefined parameters (or pointers to such parameters) or results (or pointers to such results). The parameter and result passing is performed utilizing the exchange registers (XREGs) depicted in Figure 1.

The prototype design of the MOLEN polymorphic processor is implemented utilizing a Virtex-II Pro

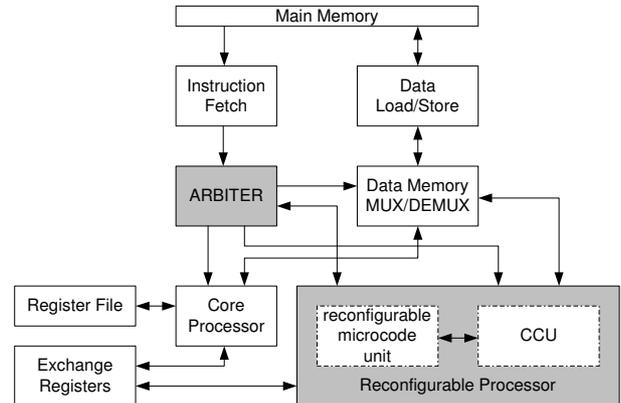


Fig. 1. The MOLEN machine organization

FPGA of Xilinx [8]. This FPGA has an embedded PowerPC (PPC) processor and the current organization of the MOLEN prototype utilizes the PPC as the core processor and extends it with a RP that occupies the remainder of the FPGA. The FPGA utilizes two main buses to connect peripherals and memory to the core processor: the On-Chip Memory (OCM) and the Processor Local Bus (PLB). Both busses are split into two separate parts, the instruction- and data-side buses. The MOLEN prototype utilizes the Instruction-Side OCM (ISOCM) bus to connect the instruction memory to the PPC and the Data PLB (DPLB) to connect to the data memory. The instruction memory is implemented using 64kB of Block RAMs (BRAMs) connected to the ISOCM bus. The data memory is implemented using 64 kB of BRAMs connected to the DPLB bus. The MOLEN prototype only utilizes the on-chip BRAMs for data and instruction memory. The RP can be configured with a custom hardware design, which performs a partial functionality of the program. To transfer the control from the core processor to the RP, the program source code is annotated. Using the MOLEN compiler [12], the annotated source code is compiled to a sequence of instructions that loads the operands of the function configured in the RP into the XREG. Figure 2

depicts a schematic overview of the organization of the MOLEN prototype.

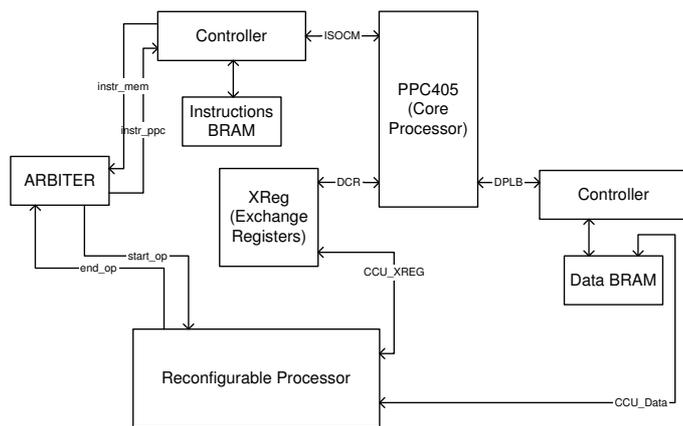


Fig. 2. Schematic overview of the organization of the MOLEN prototype

Due to the limited amount of on-chip memory, the MOLEN prototype cannot support an operating system (OS). In order to increase the memory size of the MOLEN prototype to support an OS, there is a need to connect it to an off-chip DDR memory. As such, we present in this paper, first the details of an off-chip DDR memory controller (using the OCM buses) that allows to share one physical memory interface to store/load both data and instructions. Subsequently, we elaborate on the (thus far undocumented) limitations of using the OCM buses for supporting Linux. Finally, we discuss the impact of these limitations on the organization of the MOLEN prototype when Linux support is required.

This paper is organized as follows. In Section II, we discuss related work and in Section III, we explain the design of the OCM-based shared DDR memory controller. In Section IV, we present details of the implementation of the previously presented controller and in Section V, we address the (thus far undocumented) limitations of executing an OS on an OCM-based design. In Section VI, we present the impact to the MOLEN prototype of such limitations and propose a new MOLEN prototype design able to support Linux. Finally, in Section VII we conclude our work and present some future directions.

## II. RELATED WORK

In this section, we first present related work that implement and connect a DDR memory on the OCM buses and, subsequently, we address arbitration related work. Finally, we present related work on the

execution of the Linux operating system on embedded system.

The use of the DSOCM bus to connect DDR memory is considered in [7], where the performance gain by changing the external memory from the PLB to the DSOCM bus is presented. Our design, besides using the DSOCM, additionally introduces a mechanism to connect memories with a latency larger than one clock cycle to the ISOCM bus. Moreover, we introduce a controller that combines both the ISOCM bus and the DSOCM bus to connect to the same physical memory. Our controller allows the memory to be addressed as if it is part of a Von-Neumann architecture. Furthermore, our design still allows techniques like scheduling, pre-fetching, pipelining, and caching to be applied to further improve performance.

The shared character of our design implies that the presented controller must arbitrate requests for instructions and requests for data in a fair manner. Memory arbitration is an important issue in multiprocessor systems. In [14], a multiprocessor system using arbitration is implemented. In [11], three important metrics for memory arbitration in general are introduced. Memory arbitration must be fair, a memory arbiter must have a low overhead and a memory arbiter must be easy to insert.

Besides the hardware design of the OCM-based shared DDR memory controller, we also run Linux [3] on our design. Linux is an open-source OS, therefore it is well-suited to be ported to embedded devices and development boards. There are several projects presenting guidelines on how to port and execute the Linux on an embedded development board similar to the ML403 and XUP boards which were mentioned previously. In [1], a guide is presented on how to port the Linux 2.4.26 kernel to Xilinx FPGA development boards. In [6], the MontaVista [2] Linux distribution is ported to the XUP board.

## III. DESIGN OF THE OCM-BASED SHARED DDR MEMORY CONTROLLER

In [10], a theoretical comparison is made between the PLB and the OCM buses using simulation results. The comparison focuses on the utilization of BRAM blocks connected to the OCM buses and to the PLB. As the amount and type of memory connected to both the PLB and OCM buses is the same, the comparison only reflects the buses' performance. The author shows that especially large applications benefit from using the OCM by offloading traffic from the PLB.

As the benefits of the OCM buses are clear from

[10], we designed an OCM-based shared DDR memory controller that utilizes both OCM buses to connect an external DDR memory on a Virtex-4 FPGA. This option allows for faster and deterministic access to the DDR memory due to the deterministic nature of the OCM buses. The controller introduces a shared address space for instructions and data. This controller consist of four basic blocks: the DSOCM controller, the ISOCM controller, the Arbiter, and the DDR controller. Figure 3 depicts the internal controller organization.

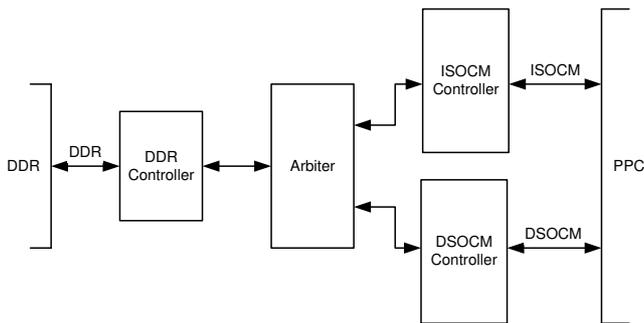


Fig. 3. Internal OCM DDR controller organization.

**ISOCM Controller:** The ISOCM bus is designed to interface with on-chip BRAM blocks that provide data one clock cycle after a request was sent. Unfortunately, DDR memory has a latency that is greater than one clock cycle. Therefore, we introduce a mechanism based on the scheme used in the MOLEN prototype [13] called the  $\mu$ -instruction arbiter [9]. The scheme used by [9] provides the PPC repeatedly with the *bl* instruction causing the PPC to store the next instruction address in the link register (LR) and branch to the current instruction. This scheme will cause the PPC to loop indefinitely. Once the PPC is allowed to continue, the *blr* is provided to the PPC to branch on the LR and continue normal execution. In our design, the PPC needs to loop until the DDR provides valid data. Since we are applying the scheme in normal execution, the contents of the LR must remain the same. As long as the DDR memory does not provide valid data, the *b* (branch) instruction is provided to the PPC, this will cause the PPC to branch to its current instruction (a branch). Summarizing, the authors of [9] allow the processor to change its internal state, while in our design the processor must be kept in the same state, therefore, we use the *b* instruction.

The normal program flow of the PPC can be interrupted by an internal or external interrupt request.

To correctly handle interrupted program execution, the requested instruction address is compared to the previous requested address. As soon as the address changes without supplying valid instructions to the PPC, the ISOCM controller detects an interrupt. We introduce a waiting mechanism to delay the interrupt request until the DDR memory finished its previous operation. Because the internal state of the processor does not change, the interrupt can be delayed.

**DSOCM Controller:** The DSOCM bus on the Virtex-4 is designed to interface with peripherals that requires variable latency. This is because the DSOCM bus introduces a *read/write complete* signal to provide handshaking between the peripheral and PPC. Therefore no mechanism has to be introduced to handle the variable latency of the DDR memory.

**Arbiter:** In the OCM-based shared DDR memory controller, we use the Arbiter to allow either the ISOCM or DSOCM controllers to gain access to the DDR controller. The Arbiter is designed in such a way that no request will be lost nor information regarding a request will be lost. The Arbiter monitors each connected controller to wait for a request using a bus monitor. Once a request is detected, the signals are buffered to be reused once the controller is granted access to the DDR memory. The Arbiter uses the read/write request and acknowledge signals to determine the start and end of a memory read or write.

**DDR Controller:** The DDR controller is provided by Xilinx as part of the PLB DDR controller included in the Xilinx Embedded Development Kit (EDK) 8.1 [15]. This controller handles all DDR specific requirements like, initialization, pre-charging and the translation of linear addresses to a combination of bank and row address.

#### IV. IMPLEMENTATION AND RESULTS OF THE OCM-BASED SHARED DDR MEMORY CONTROLLER

We implemented and tested the OCM-based shared DDR memory controller on the Virtex-4FX12 FPGA on the Xilinx ML403 [16] development board. All components of the memory controller are developed using the hardware description language VHDL and are synthesized using the XST synthesis compiler which is distributed with the Xilinx EDK 8.1. The design is implemented as a custom module for EDK. The PPC is configured to run at 300MHz (maximum available), the PLB and OCM buses run at 100MHz.

Table I presents an overview of the post-synthesis

TABLE I  
RESOURCE UTILIZATION

	Total	OCM DDR	PLB DDR
Slices	5,472	622(11%)	597(10%)
Flip Flops	10,944	778(7%)	781(7%)
LUTs	10,944	952(8%)	746(6%)

resource utilization. Post-synthesis results of the OCM-based shared DDR memory controller indicate that a comparable PLB based controller utilizes slightly less resources. This difference is mainly due to the result of the hardware needed for arbitration included in our design, while the PLB arbitration is not taken into account in the PLB design. Post-synthesis timing estimates a theoretical value for the maximal operating frequency of the OCM-based shared DDR memory controller of 123MHz and 157MHz for the PLB design, however the real operating frequency of both designs is 100MHz. The difference is due to different VHDL coding techniques. We utilized high-level behavioral VHDL versus device-specific structural VHDL for the PLB design.

For performance analysis, we utilized a synthetic benchmark based on two nested for-loops that write and read to/from the memory address space. Both instructions and data reside in the DDR memory. However, since part of the DDR memory contains instructions and local variables, the benchmark can not utilize the entire 16MB addressable space of the DDR memory. We tested the memory connected to both the PLB and the OCM buses with peripherals connected to the PLB. If the peripherals connected to the PLB are not used, the OCM-based design introduces an average performance improvement is 6.34% compared with the PLB based design. If the peripherals connected to the PLB request the bus every cycle it is available (worst-case), the improvement of the OCM design is 13.34% on average.

## V. LINUX SUPPORT ON THE OCM-BASED DESIGN

MontaVista provides a Linux distribution [2] based on the Linux 2.4 kernel. This distribution includes the board-dependent files needed to run Linux on the Xilinx ML403 development board. In order to use of this version of Linux on the ML403, a patch is applied to the Linux source tree. MontaVista Linux can be used with the ML403 board by following the steps listed in [6].

Using MontaVista Linux on an OCM-based system is, in theory, straight forward. However, the OCM bus provides non-cachable access to both instruction-side and data-side memory [5]. As such, all cache related behavior in the Linux kernel should be taken out. The cache-related instructions are well specified in [4] and are only used in the parts of the Linux kernel source code which are written in assembly. Therefore, these instructions are only used in the architecture dependent part of the kernel source tree.

Cache instructions can be separated into two groups, instructions that are related to the data-cache and to the instruction-cache. The use of one of the cache storage instructions on a OCM-based memory leads to a instruction- or data-storage interrupt, which cannot be handled by the OCM-based system due to its non-cacheable nature. This implies that the cache related instructions have to be stripped from the Linux kernel. However, the boot process of the kernel still halted as soon as the `start_kernel()` function was called. The jump to the `start_kernel()` function is the first use of the virtual address range assigned to the Linux kernel. The kernel halted by raising a machine check interrupt, which is caused by an invalid address translation.

Since the kernel halted at the first use of the virtual address space, we decided to run some simple tests involving the use of virtual-to-physical address translation. Table II lists the tested physical and virtual address combinations and the observed PPC behavior. The results of these tests lead us to the conclusion that there are limitations on the use of virtual-to-physical address translation in combination with the OCM buses. This limitation implies that when using the OCM buses to connect memory, we can only use virtual memory as long as the virtual address is the same as the physical address, which is conceptually the same as using a system without a memory management system. Therefore, porting MontaVista Linux with memory management unit to a system based on an OCM memory system is not possible.

## VI. IMPACT OF OCM LIMITATIONS ON THE MOLEN PROTOTYPE

In Section III, we demonstrated that the use of an OCM-based shared DDR memory controller to support an OS on the MOLEN prototype would be the best option in terms of performance. However, in Section V we reached the conclusion that the proposed memory system is not suitable to execute an OS due to the lack of virtual-to-physical translation. In this

TABLE II  
ADDRESS TRANSLATION BEHAVIOR

Physical address	Virtual address	Behavior
0x00000000 - 0x00FFFFFF	0x00000000 - 0x00FFFFFF	Normal
0x00000000 - 0x00FFFFFF	0xC0000000 - 0xC0FFFFFF	Machine check exception
0xC0000000 - 0xC0FFFFFF	0x00000000 - 0x00FFFFFF	Machine check exception
0xC0000000 - 0xC0FFFFFF	0xC0000000 - 0xC0FFFFFF	Normal

section, we propose an alternative memory system design based on the PLB buses. This option introduces a minimum deviation from the contemporary way of connecting external modules to the PPC and, therefore, enables the execution of the Linux kernel on the MOLEN prototype.

In this scenario, the memory size is expanded by using the PLB DDR controller provided by Xilinx. In the original MOLEN prototype, the arbiter is connected to the ISOCM bus (which could only access on-chip BRAM, that have a limited size). The option presented in this section relies on moving the ARBITER from the ISOCM bus to the instruction interface of the PLB, IPLB. In addition, the BRAMs used to store the data and instructions required by the RP are omitted. Instead of using BRAMs, we introduce a PLB to RP controller, able to retrieve data from every peripheral addressable through the PLB. The use of such a controller implies the addition of a handshaking protocol in the RP to the data memory interface.

To provide a common interface to the PPC for all RP related connections, we propose to integrate the connection to the XREG in the PLB to RP controller. In the current MOLEN prototype, the XREGs are addressed using the DCR bus, which does not use the common read/write instructions, but uses specific DCR instructions that are only available in the private mode of the PPC [4]. Therefore, transferring data to the XREGs outside of the Linux kernel space (which is the common case), has to be handled by a specific kernel device driver. If the connection to the XREGs is included in the PLB to RP controller, the XREGs can be assigned their own address space.

The current RP design does not support data memory with a variable latency to be connected, therefore, the interface between the RP and data memory should be extended with a facility to support variable latency read/write operations. To preserve as much backwards compatibility as possible, we propose to

use the current RP to BRAMs connection as basis for the PLB to RP connection and expand this connection with a *read/write complete* signal to inform the RP of the completion of a PLB operation. Summarizing, the use of a MOLEN prototype based only on the PLB implies:

- Add the Xilinx PLB DDR controller to the MOLEN prototype.
- Port the MOLEN arbiter to the Instruction PLB.
- Add handshaking on the RP to data memory interface.
- Develop a PLB to RP controller.

Figure 4 depicts the proposed organization of the MOLEN prototype.

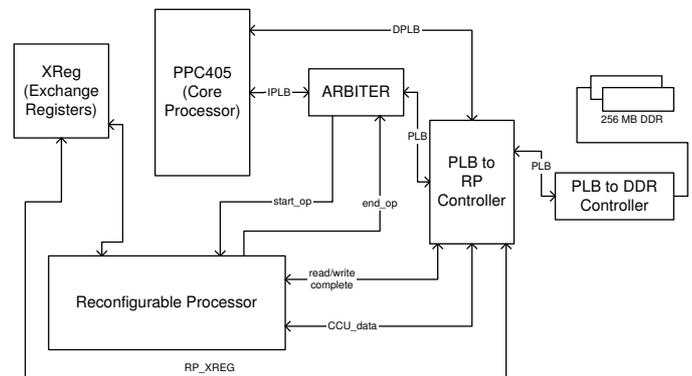


Fig. 4. MOLEN prototype based only on the PLB

## VII. CONCLUSIONS

In order to support an OS on the MOLEN processor prototype, there is a need to increase the size of the memory the MOLEN prototype can support. As such, in this paper, we first presented an OCM-based shared DDR memory controller implemented and tested on the Virtex-4 FPGA. We achieved an improvement from 6% (best-case) to 13% (worst-case) over a PLB based design. Subsequently, we explained the steps needed to execute Linux on the previously designed system. We presented the (thus far undoc-

umented) limitations such a system can impose on the Linux. Due to this limitations, we also presented the new design of the MOLEN prototype that can support Linux. Implementing the MOLEN prototype to support Linux implies implementing the proposed modules and the necessary changes to the MOLEN prototype design. This is considered future work.

#### REFERENCES

- [1] Brigham Young University Linux on FPGA Project.
- [2] Monta Vista Linux.
- [3] The Linux Operating System.
- [4] PowerPC Processor Reference Guide, September 2003.
- [5] PowerPC 405 Processor Block Reference Guide v2.1, 2005.
- [6] Jonathon W. Donaldson. Porting MontaVista Linux to the XUP Virtex-II Pro Development Board, August 2006.
- [7] B. Donchev, G.K. Kuzmanov, and G. N. Gaydadjiev. External Memory Controller for Virtex II Pro. In *Proceedings of International Symposium on System-on-Chip 2006*, pages 37–40, November 2006.
- [8] G.K. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis. The MOLEN Processor Prototype. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004)*, pages 296–299, 2004.
- [9] G.K. Kuzmanov and S. Vassiliadis. Arbitrating Instructions in an  $\rho\mu$ -coded CCM. In *Proceedings International Conference on Field Programmable Logic and Applications (FPL'03)*, pages 81–90, 2003.
- [10] Kraig Lund. *PLB vs. OCM Comparison Using the Packet Processor Software*. Xilinx Corporation. Xilinx Application Note 644.
- [11] Iyad Ouais and Ranga Vemuri. Efficient Resource Arbitration in Reconfigurable Computing Environments. In *DATE 2000: Proceedings Design, Automation and Test in Europe*, pages 560–566, 2000.
- [12] E. Moscu Panainte, K.L.M. Bertels, and S. Vassiliadis. The PowerPC Backend Molen Compiler. In *14th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 434–443, September 2004.
- [13] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, and E. Moscu Panainte. The Molen Polymorphic Processor. *IEEE Transactions on Computers*, 53:1363–1375, November 2004.
- [14] Evaggelos Vlachos. Design and Implementation of a Coherent Memory Sub-System for Shared Memory Multiprocessors. Master's thesis, Computer Science Department, University of Crete, July 2006.
- [15] Xilinx Cooperation. *Embedded Development Kit*.
- [16] Xilinx Cooperation. *Virtex-4 ML403 Embedded Platform*.