Coupling of a Reconfigurable Architecture and a Multithreaded Processor Core with Integrated Real-Time Scheduling

Sascha Uhrig[1], Stefan Maier[1],
Georgi Kuzmanov[2], Theo Ungerer[1]

[1]University of Augsburg
Institute of Computer Science
86159 Augsburg, Germany

{uhrig, ungerer}@informatik.uni-augsburg.de

[2]Delft University of Technology
Computer Engineering Laboratory
Electrical Engineering, Mathematics,
and Computer Science Department
2628 CD Delft, The Netherlands
G.Kuzmanov@ewi.tudelft.nl

Abstract

This paper defines a real-time capable interface between the simultaneous multithreaded CarCore processor and a MOLEN-based reconfigurable unit. CarCore is an IP core that enables simultaneous execution of one hard-real-time thread and further non-real-time threads. The coupling described in this paper extends CarCore by a reconfigurable hardware such that both can execute different threads simultaneously, while the real-time behavior of the hard-real-time thread is not harmed. The challenge is the design of a common memory interface for both, the CarCore and the reconfigurable hardware, such that memory operations fulfil hard-real-time constraints. Experimental results with an MJPEG benchmark show an overall application speedup of 2.75 which approaches the theoretically attainable maximum speedup of 2.78.

Keywords: multithreading, real-time scheduling, reconfigurable architecture

Acknowledgments: This work has been supported by *The European Network of Excellence on High-Performance Embedded Architecture and Compilation* (HiPEAC).

1 Introduction

Reconfigurable hardware is generally used to speed up data-intensive operations. Depending on the target application, reconfigurable hardware units could be located directly within the pipeline as functional unit, besides the pipeline as coprocessor, or outside the memory interface and the caches [1]. In this paper we focus on reconfigurable hardware which works as coprocessor in parallel to the processor. In most cases the reconfigurable unit takes over control of the memory system to reach the highest possible data throughput. Hence, the general purpose processor is stalled or slowed down.

Several research projects like *GARP* [3] and *MOLEN* [9] introduce instruction set extensions to configure and run a reconfigurable architecture which is not located directly within the pipeline. The *XTENSA* chip from Tensilica, the CHIMAERA [2] project, and the *PRISC* processor [7] use reconfigurable functional units within the processor pipeline. The *PiCoGA* architecture is a VLIW processor pipeline [5] with an additional pipelined reconfigurable architecture which works as a functional unit.

In contrast to the known projects, which use single threaded processors, our objective are (1) to combine a multithreaded processor core with a reconfigurable extension as coprocessor such that other threads can execute in parallel to the thread that uses the reconfigurable hardware and (2) to provide full real-time capability to one thread.

The main problem arises from simultaneous memory accesses of the reconfigurable hardware and of other threads running within the CarCore. Our solution is to extend the real-time scheduling inside the CarCore to handle concurrent memory accesses from the reconfigurable unit and the processor core.

The next two sections shortly introduce into the CarCore and the MOLEN reconfigurable unit. In section 4 we describe our interface between both which we evaluate in section 5. Section 6 concludes the paper.

2 The CarCore

The CarCore [8] is a superscalar, five stage pipelined RISC processor core, which features simultaneous multithreading with two specialized pipelines. The processor is based on the TriCore 1 instruction set [4]. Figure 1 shows the block diagram of the CarCore. Each pipeline contains one register set per thread and its own ALU and branch unit. The fetch unit maintains four program counters and four instruction windows decouple the fetch stage from the rest of the pipeline. The scheduler keeps track of the thread priorities and issues instructions to the data and the address pipeline. Some complex instructions of the TriCore instruction set, i.e. CALL and RET, are realized as microcodes which are also interruptible by instructions of other threads.

Scheduling: Every clock cycle the scheduler chooses instructions from the instruction windows and tries to issue them to both pipelines. Thereby it follows the priorities of the threads which are given by the priority unit. Up to now, two priority schemes are supported. The fixed priority scheme (FPP) associates fixed priorities to the hardware threads, i.e. thread 0 has always highest, thread 3 lowest priority. This scheme prefers one thread which can be considered as a real-time thread. The round-robin (RR) scheme circularly changes

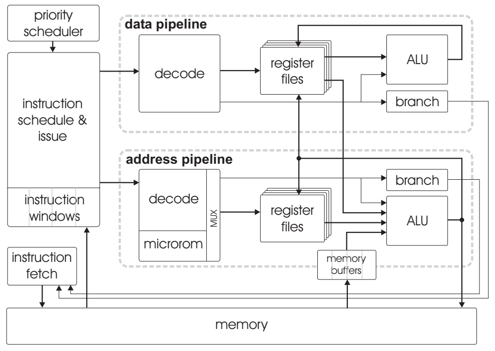

Figure 1. Block Diagram of CarCore.

the thread priorities and therefore equally disposes processing time.

Memory Accesses: Though memory accesses may last several cycles the address pipeline must not be blocked by one thread and the runtime behaviour of the other threads must not be disturbed. To address this issue the CarCore executes load accesses as split-phase loads. The scheduler separates address generation and data write-back in two operations. Valid data from memory is latched in a buffer until the scheduler explicitly inserts the data into the register file. In between the scheduler assigns instructions of other threads to the address pipeline. Thus it is possible that multiple threads initiate load/store accesses in an overlapped fashion without disturbing each other.

3 The Reconfigurable Architecture

The MOLEN Polymorphic Processor [9] combines a single threaded general purpose processor with a reconfigurable custom computing unit. Figure 2 shows the Molen architecture consisting of a general purpose processor (GPP) and a reconfigurable processor (RP). The Arbiter predecodes and issues instructions to either of the processors. The exchange registers (XREGs) allow data exchange between the GPP and RP. The RP itself mainly contains the reconfigurable microcode unit (rm-code unit) and a custom configurable unit (CCU). The basic operations of the RP are *set*

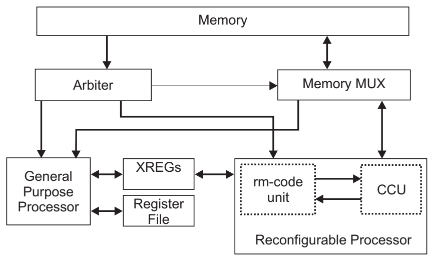

Figure 2. The Molen organization

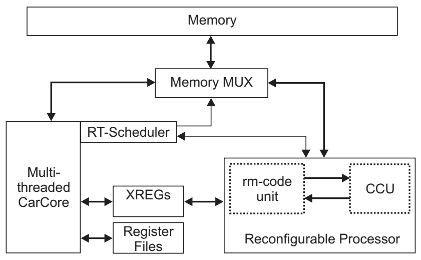

Figure 3. The CarCore reconfigurable unit

and *execute*. The set instruction configures the CCU, the execute instruction performs a computation on the CCU.

Operation: Before the RP is ready for operation it has to be configured with the configuration data by the help of the *set* instruction. After configuration, computation data like pointers has to be loaded into the XREGs. The last task is the *execute* instruction with a pointer to a so-called *microprogram* which deals as control sequence for the CCU. The GPP is stalled during the whole computation of the RP. During this period the RP takes over the control of the whole system, i.e. it has exclusive access to the memory system and the XREGs.

4 Integration

We integrated the CarCore and the RP of Molen into the so called *Reconfigurable CarCore* (RCC) [6]. Figure 3 shows its overall architecture. Because of the aimed real-time capability of all hardware thread slots, we have to prohibit the RP to take over control of the whole system. Otherwise the runtime behavior of the still running threads would be harmed. We made the constraint that only one thread slot can use the RP at a time. Additionally, the thread using the RP has to be suspended during the RP execution. Hence we obtain a direct relationship between the task of the RP and a single thread slot.

The RP was modified in a way that its memory interface works as slave rather than as master. If the RP wants to access the main memory, it first checks the availability of the memory bus. If no other bus request is pending it initiates the memory access immediately. Otherwise it makes a request to the thread scheduler of the CarCore. The scheduler checks the priority of the suspended corresponding thread against the priorities of all other running threads. Depending on this priority check, the scheduler grants or delays the memory request of the RP.

5 Performance Evaluation

To evaluate the impact of the reconfigurable unit a single-threaded workload was chosen. An implementation of the MJPEG algorithm serves as benchmark with a computational intensive DCT kernel. Figure 4 shows the runtime of MJPEG in processor cycles using pure software execution compared to the execution using the reconfigurable unit. This is depicted with and without compiler optimization flags.

The comparison of the not optimized binaries shows a speedup of the pure DCT function call of about 150 times using the reconfigurable unit. The DCT function is mostly executed by the RU and needs more than half of the processor cycles in software. The speedup of the whole application is 2.75. The theoretical maximum is 2.78 (assumed a runtime of zero cycles for the DCT). The performance gain of the optimized binaries is 34.44 for the kernel and 1.97 for the application (theoretical maximum of 2.03).

For the evaluation of a multithreaded workload all four threads executed the same MJPEG benchmark and independently processed the same picture. Figure 5 shows the runtimes of the optimized binaries using the fixed and the round robin

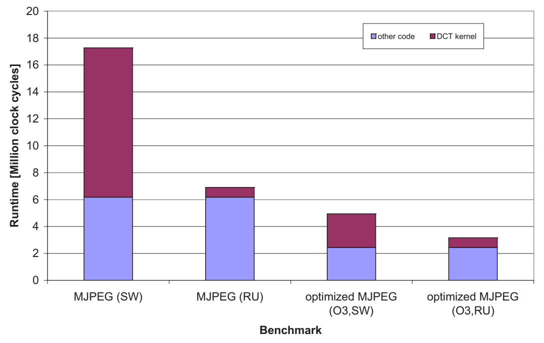

Figure 4. Runtimes of MJPEG benchmark in one single thread.

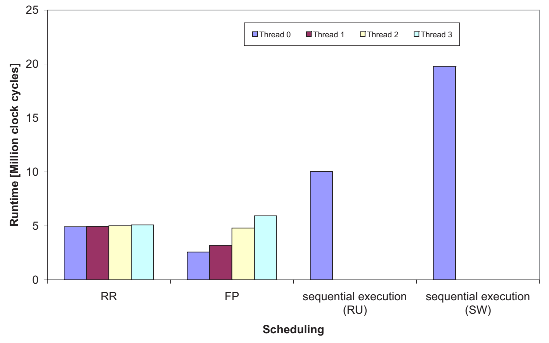

Figure 5. Runtimes of the multithreaded execution of the MJPEG benchmark.

priority scheme. They are compared to the runtime of the sequential execution of the benchmark with and without using the reconfigurable unit. The speedup of the parallel multithreaded workload compared to the sequential execution totals to 1.97 (RR) and 1.69 (FPP). Compared to the sequential execution purely in software the multithreaded speedup nearly doubles to 3.89 (RR) and 3.34 (FPP) respectively.

6 Conclusions and Future Work

We presented a real-time capable combination of a reconfigurable architecture and a multithreaded processor core with integrated real-time scheduling. The so far implemented FPP scheduling scheme allows a significant WCET analysis of the thread with the highest priority and it could guarantee that the highest priority thread keeps its deadlines despite of the usage of the reconfigurable architecture. It is unimportant which thread uses the reconfigurable architecture. In the future we will enhance the scheduling by the integration of the *Guaranteed Percentage* scheduling which guarantees predefined percentages of the performance to the threads. It will be possible to predict the run-time behavior of all threads, so all threads are capable for hard-real-time requirements. Additionally, we will integrate a *share* technique that enables all threads to use the reconfigurable architecture in an area shared manner.

References

[1] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software. *ACM Computing Surveys*, 34(2):171–210, June 2002.

[2] S. Hauck, T. Fry, M. Hosler, and J. Kao. The Chimaera Reconfigurable Functional Unit. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 1997)*, pages 87–96, 1997.

[3] J. Hauser and J. Wawrzynek. Garp: a MIPS processor with a reconfigurable coprocessor. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 1997)*, pages 12–, 1997.

[4] Infineon Technologies AG, München. *TriCore 1 Architecture Manual*, Sept. 2002. Version 1.3.3.

[5] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri. A VLIW Processor With Reconfigurable Instruction Set for Embedded Applications. *IEEE Journal of Solid-State Circuits*, 38(11), Nov. 2003.

[6] S. Maier. Reconfigurable Extension for Car-Core. Technical Report 2005-17, Institute of Computer Science, University of Augsburg, 2005.

[7] R. Razdan and M. D. Smith. A High-Performance Microarchitecture with Hardware-Programmable Functional Units. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 172–80, 1994.

[8] S. Uhrig, S. Maier, and T. Ungerer. Toward a processor core for real-time capable autonomic systems. In *IEEE International Symposium on Signal Processing and Information Technology, ISSPIT 2005, Athen, Greece*, 2005.

[9] S. Vassiliadis, S. Wong, and S. D. Cotofana. The MOLEN $\rho\mu$-coded Processor. In *in 11th International Conference on Field-Programmable Logic and Applications (FPL), Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147*, pages 275–285, August 2001.