The Molen FemtoJava Engine

Julio C. B. Mattos
Embedded Systems Lab
UFRGS
Porto Alegre, Brazil
julius@inf.ufrgs.br

Stephan Wong
Computer Engineering Lab
TUDelft
Delft, The Netherlands
J.S.S.M.Wong@ewi.tudelft.nl

Luigi Carro
Embedded Systems Lab
UFRGS
Porto Alegre, Brazil
carro@inf.ufrgs.br

Abstract

This paper presents the Molen FemtoJava engine that is extended with concepts taken from the Molen polymorphic processor. This allows for the existing FemtoJava to be augmented with reconfigurable hardware with only a single extension of the bytecodes and thereby but still allowing the implementation of arbitrary hardware implementations. Therefore, computationally intensive functions can be moved to the reconfigurable hardware to improve their performance. Our experimental results on MP3 decoding, which is a common embedded application, can be improved by at least of 27% reduction of execution cycles with minimal additional hardware area (about 7%). Finally, our synthesis results also show that the Molen extension of the FemtoJava engine only required an additional 10% of area (in terms of FPGA slices).

1. Introduction

In reconfigurable computing, reconfigurable hardware structures augment general-purpose processors by offloading computationally intensive operations from software to these structures. This enables quick prototyping or even product roll-out due to two reasons. First, the application can be specified in any high-level programming language (e.g., Java) and thereby exploiting state-of-the-art compiler technologies to quickly generate efficient machine code (bytecode). Second, computationally intensive operations can be modeled in a high-level hardware description language (e.g., VHDL) and thereby utilizing automatic synthesis tools that can quickly generate hardware implementations. Consequently, the ability to do this is becoming increasingly more important in the design of embedded systems. Moreover, from a technological point of view, the increasing densities of field-programmable gate arrays (FPGAs) coupled with falling prices of FPGAs (due to increasing production volumes) allows for integration of several embedded systems into a single FPGA device.

Moreover, software development costs must not be overlooked as they are quickly becoming a major cost component in the design of embedded systems. Therefore, developers have embraced Java over the past few years, because it can provide high portability and code reuse for their applications [12, 15]. In addition, Java has features such as small code sizes and small memory footprints that stand out against other programming languages. These features make Java an attractive choice as the specification and synthesis language of embedded systems. Additionally, the efficient execution of Java programs, especially in embedded systems, can be achieved by direct execution of Java bytecodes in hardware, like PicoJava [13] and the Delft-Java engine [7]. In order to solve the software problem allied to an efficient implementation in terms of area, speed and power, we have developed a Java processor called FemtoJava and its supporting environment [9].

In this paper, we propose to apply Molen concepts [16] to the described FemtoJava engine to extend its capabilities support reconfigurable hardware (able to perform a wide variety of implementations) with only a one-time architectural extension of the Java bytecode. In addition, the extension includes an interface to 'exchange' parameters and results between the software code and the hardware implementation. We implemented a well-known application (MP3 decoding) found in many embedded systems to show the benefits of the Molen FemtoJava engine. Our results show that the existing FemtoJava engine could be easily extended to become the Molen FemtoJava engine with minimal hardware overhead (about 10% in terms of FPGA slices). Additionally, our results show that the total number of execution cycles (when compared to the fastest software alone solution) can be reduced by about 27.5% and only paying a small amount of the FPGA area overhead (about 7%).

This paper is organized as follows. Section 2 discusses the related work. Section 3 presents an overview of the proposed approach and section 4 introduces the results obtained from the experiments. Finally, Section 5 draws conclusions and discusses future work.

2. Related Work

Reconfigurable systems are efficient in implementing computationally intensive parts of software in reconfigurable hardware. There are many approaches in the research community that present a general-purpose processor coexisting with a reconfigurable hardware. A good overview on reconfigurable computing and software is presented in [3].

There are several approaches that combine reconfigurable computing and the Java language. Most of current approaches use the reconfigurable hardware as a coprocessor and the host general-purpose processor runs a Java Virtual Machine [6, 10, 4, 11]. Fleischmann [6] presents a run-time manager schedule method for execution either as software on the JVM of the host processor or as hardware on the reconfigurable hardware. The scheduling depends on the dynamic behavior of the application and on the current partitioning table chosen by the designer. A set of tools that support the run-time execution of applications that mix software running on networks of workstations and reconfigurable hardware has been also proposed [10].

There are approaches that use Java as an input to high-level reconfigurable compilation, such as described in [2]. This work presents a compiler that starts from a representation of a high-level algorithmic abstraction in a Java virtual machine (JVM) instructions and it targets reconfigurable computing architectures based on a commercial FPGA. In [1], a dynamic translation of Java bytecodes into combinational logic is presented. This approach combines a reconfigurable array with a binary translation mechanism in a Java machine. A compiler and language that uses the high level syntax and semantics of Java with additions to support reconfigurable hardware description has been also proposed in [8].

Our approach uses a Java processor as a host processor making the Java execution more efficient in terms of performance and power, instead of the current approaches that use a JVM. In addition, the software tools automatically generate an efficient Java processor adapted to each application by removing unused hardware. Finally, we extended our Java processor with a reconfigurable hardware structure using the Molen approach that allows for the implementation of arbitrary function with a single instruction set extension.

3. Molen FemtoJava organization

The proposed approach combines the Molen [16] and Sashimi/FemtoJava [9] concepts. On the one hand, the Molen concepts allow the designer to modify and extend the processor functionality using reconfigurable hardware. On the other hand, the Sashimi approach uses concepts such as system specification using a subset of Java, CPU customizing and high-level design space exploration providing a general methodology to support the development of embedded applications, based on a single-language and single-chip approach to reduce costs (our main goal is to reduce design time) allowing the designer to rapidly develop, simulate and validate embedded applications. The algorithm is programmed in Java language and the resulting Java bytecodes will be executed by a customized processor, FemtoJava [9].

This paper presents an approach that combines concepts taken from the Molen Polymorphic processor and the Sashimi/FemtoJava platform. Fig. 1 illustrates the main components of Molen-FemtoJava organization. The main components are de core processor (FemtoJava) and the reconfigurable processor (RP). The reconfigurable processor is divided into the $\rho\mu$-code unit and the custom configured unit (CCU). The CCU consists of reconfigurable hardware. All code runs on the FemtoJava except pieces of code implemented on the CCU in order to speed up program execution. Exchange of data between the GPP and the RP is performed via the exchange registers (XREGs). The XREGs in our approach are implemented inside the FemtoJava processor and this file register is not a conventional register file, but it is mapped in the processor memory space. In this way, when the processor wants to communicate with the RP, it writes or reads in its own memory space.

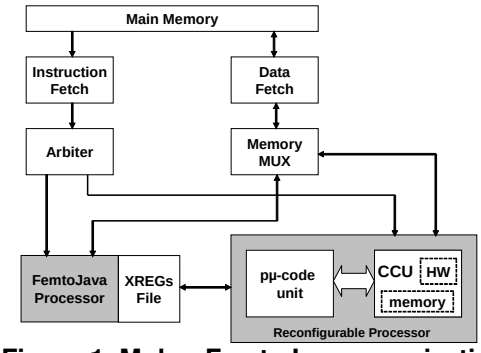

Figure 1. Molen-FemtoJava organization

We implemented the minimal instruction set (πISA) of the $\rho\mu$-code unit. Four instructions are implemented using unused bytecodes: set, execute, movtx, movfx. The set and execute instructions are new instructions, however, movtx and movfx are mapped to existing FemtoJava instructions (store_idx and load_idx). In fact, the set and execute instructions are issued to the reconfigurable processor while movtx and movfx are issued to the processor.

All the modifications of the original code (translated to a code that makes calls to reconfigurable hardware) are performed by the Sashimi Tool. The Sashimi Tool extracts the code that will be implemented in hardware and inserts the right instructions to do the communication to the RP.

The main steps of the process to design an application using the Molen and Sashimi/FemtoJava concepts are: the

designer codes the application using Java and uses the Java compiler to generate the Java bytecodes. Subsequently, the designer has to select what routines that will run in the reconfigurable hardware (there are profiling tools to help the designer). And finally, the designer utilizes the Sashimi tool.

The main steps of the process to insert the Molen instructions are as follows: the Sashimi tool searches in the Java bytecodes for the functions that are implemented in reconfigurable hardware and extract them from the bytecode. When there is a call for these functions, the tool inserts the appropriate Molen instructions (set, movtx, movfx, execute) in the bytecode.

During the execution in the Molen-FemtoJava architecture, the arbiter selects the instructions to be issued to FemtoJava processor or to the reconfigurable processor (RP). When the arbiter issues instructions to the RP, at the same time it issues to FemtoJava processor nop (no operation) instructions. Furthermore, when the RP is processing the function, the FJ processor halted (just waiting until the reconfigurable processor finishes its task).

In order to continue the execution of regular bytecodes when executing on the RP, an additional instruction must be added to handle synchronization issues (see [16]). In the current Molen FemtoJava engine this is left out for simplicity reasons.

4. Experimental Results

This section presents the results of our approach using Molen and Sashimi/FemtoJava concepts. The framework, the case study application and results are presented in the following.

4.1. Application

MPEG-Audio is an international standard for digital high quality sound compression. Generally speaking, the standard takes a digital audio file and reduces its size, while maintaining the quality of the recording. Our application code is based on a description freely available on the Internet [14]. All the MP3 code was written in Java but obeying certain constraints of the current Sashimi and FemtoJava architecture. An example constraint is the use of integer arrays instead of floating-point arrays because there is no such unit available in the processor.

4.2. Results

In order to identify the functions that are more suitable for hardware implementation, we performed the measurements using the CACO-PS [5] cycle-accurate simulator and the FemtoJava pipeline version to determine the amount of time spend in each function. Table 1 presents the profiling results of the MP3 application decoding a sample song. The table shows the methods that spend more time during the decoding process. The calculatePCMSamples consumes more than 35% of the application time, thus it is the best candidate for hardware implementation. The IMDCT method requires about 10% of application time, making this method another good candidate for hardware implementation.

Table 1. MP3 Software Profiling Results.

Method Name	Executed Time (%)
calculatePCMSamples	37.26
IMDCT	11.97
dequantizeSamples	9.23
stereo	7.74

The FemtoJava processor was described in VHDL and the additional Molen hardware components have has been described in VHDL. These componets are the arbiter, XREGs and $\rho\mu$unit. Our targeted hardware platform was an available XUP Virtex-II Pro Development System which uses a Xilinx Virtex-2 Pro XC2VP30 FPGA [17]. Table 2 shows the synthesis results for the pipelined FemtoJava processor and the FemtoJava processor plus the additional Molen components. The synthesis results were obtained using Xilinx ISE version 7.1i. The first column presents the FPGA resources considered. Column two shows only the results for the FemtoJava processor. The third column reports the synthesis results considering combined the FemtoJava and Molen organization. These results show that for this target device, the FemtoJava and Molen use a few amount of FPGA resources.

Table 2. Molen and FemtoJava Results.

FPGA resources	FJ	FJ + Molen
# Slices	1506	1661
# Flip Flops	808	955
# 4 inp LUTs	2757	3078
Fmax (MHz)	81.15	79.50

Using the profiling results, we decided to implements two methods in hardware; namely IMDCT and calculatePCMSamples. Both methods were described in VHDL. Table 3 presents the FPGA resources necessary to implement each function in hardware. The second column shows the results of IMDCT and the third column reports the results of the calculatePCMsamples implementation. One can observe that both hardware implementations use a few amount of FPGA area.

The table 4 shows the number of cycles of each CCU implementation necessary to perform its task. For comparison, the table also presents the number of cycles of each

software routine. For example, using the IMDCT functions implemented in SW costs 76,652 cycles for each call and using the in reconfigurable hardware costs just 5,346 cycles. This cost includes the cycles spent on the hardware execution plus the cycles sent with transferring parameters transfer.

Table 3. Synthesis Results of the CCUs.

FPGA resources	IMDCT	CalcPCMsamples
# Slices	24	96
# Flip Flops	12	49
# 4 inp LUTs	46	181
Fmax (MHz)	220.30	204.80

Table 4. Performance Results of the CCUs.

FPGA resources	IMDCT	CalcPCMsamples
# cycles (SW)	76,652	12,911
# cycles (CCU)	5,346	7,232

Table 5 shows the results of the improvement performance (total cycles reduction) and the area overhead (added area over the FemtoJava-Molen) using the proposed approach. This table shows the performance improvements and area overhead in percentage using only the IMDCT method implemented in hardware (the first row), using the only calculatePCMSamples in hardware (row two), and using both.

Table 5. MP3 performance and area overhead.

CCU	Total Cycles Reduc.(%)	Added Area (%)		
		Slices	Flip Flops	4 input LUTs
calculatePCM	16.39	5.77	5.13	5.88
IMDCT	11.11	1.44	1.25	1.49
Both	25.50	7.21	6.38	7.37

5. Conclusions

This paper presented an approach that combines the Molen and Sashimi/FemtoJava concepts. Therefore, in this design flow the designer can modify and extend the processor functionalities and in the other hand, the designer can still use a widely used language. Experimental results have confirmed the hypothesis that extending FemtoJava with Molen concepts introduce a minimal effort and area overhead. Moreover, adding this specialized hardware reduces 27.50% of execution cycles in MP3 decoding with only 10% area increase (in terms of FPGA slices) for the Molen components and 7% area increase for the special hardware implementations

References

[1] A. C. S. Beck and L. Carro. Dynamic reconfiguration with binary translation: breaking the ILP barrier with software compatibility. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 732–737, New York, NY, USA, 2005. ACM Press.

[2] J. M. P. Cardoso and H. C. Neto. Compilation for FPGA-Based Reconfigurable Hardware. *IEEE Des. Test*, 20(2):65–75, 2003.

[3] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[4] P. Faes, M. Christiaens, and D. Stroobandt. Transparent Communication between Java and Reconfigurable Hardware. In *Proceedings of the 16th IASTED International Conference Parallel and Distributed Computing and Systems*, pages 380–385, 11 2004.

[5] A. Filho, J. C. B. Mattos, F.R.Wagner, and L.Carro. CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator. In *Proc. 16th Symposium on Integrated Circuits and Systems Design*, pages 349–359, Los Alamitos, 2003. IEEE Computer Society Press.

[6] J. Fleischmann, K. Buchenrieder, and R. Kress. Codesign of embedded systems based on Java and reconfigurable hardware components. In *DATE '99: Proceedings of the conference on Design, automation and test in Europe*, page 26, New York, NY, USA, 1999. ACM Press.

[7] C. J. Glossner and S. Vassiliadis. The Delft-Java engine. *Java microarchitectures*, pages 105–123, 2002.

[8] J. Hopf. Comparing the Bitstreams of Applications Specified in Hardware Join Java and HandelC. In D. S. Komatsu, editor, *2003 IEEE International Conference on Field-Programmable Technology (FPT)*, pages 399–402, Tokyo, 2003. IEEE.

[9] S. A. Ito, L. Carro, and R. P. Jacobi. Making Java Work for Microcontroller Applications. *IEEE Des. Test*, 18(5):100–110, 2001.

[10] L. S. King, H. Quinn, M. Leeser, D. Galatopoullos, and E. Manolakos. Run-Time Execution of Reconfigurable Hardware in a Java Environment. *iccd*, 00:0380, 2001.

[11] E. Lattanzi, A. Gayasen, M. Kandemir, V. Narayanana, L. Benini, and A. Bogliolo. Improving Java Performance by Dynamic Method Migration on FPGAs. In *Proceedings of RAW 2004*, 2004.

[12] G. Lawton. Moving Java into Mobile Phones. *Computer*, 35(6):17–20, 2002.

[13] H. McGhan and M. O'Connor. PicoJava: A Direct Execution Engine For Java Bytecode. *Computer*, 31(10):22–30, 1998.

[14] MP3. http://www.mp3-tech.org/. 2006.

[15] D. Mulchandani. Java for Embedded Systems. *IEEE Internet Computing*, 2(3):30–39, 1998.

[16] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte. The MOLEN Polymorphic Processor. *IEEE Trans. Computer*, 53(11):1363–1375, 2004.

[17] Xilinx. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet. 2005.