Codesign of Embedded Systems Based on Java and Reconfigurable Hardware Components

Josef Fleischmann
Technical University of Munich
Inst. of Electronic Design Automation
D-80290 Munich, Germany
Josef.Fleischmann@ei.tum.de

Klaus Buchenrieder, Rainer Kress
Siemens AG
Corporate Technology
D-81730 Munich, Germany
{Klaus.Buchenrieder|Rainer.Kress}@mchp.siemens.de

Abstract

In the design of embedded hardware/software systems, exploration and synthesis of different design alternatives and co-verification of specific implementations are the most demanding tasks. Networked embedded systems pose a new challenge to existing design methodologies as novel requirements like adaptivity and runtime-reconfigurability arise. In this paper, we introduce a co-design environment based on the Java language which supports specification, co-synthesis and prototype execution for dynamically reconfigurable hardware/software systems.

1. Specification and Synthesis

For the design of embedded hardware/software systems with reconfiguration capabilities, we have developed a design exploration and prototyping platform. The target architecture for such systems consists of a microprocessor running a Java virtual machine, and a hardware processor consisting of one or more FPGAs. An overview of our design flow for co-synthesis is illustrated in figure 1. Starting from an initial Java specification, profiling data is gathered while executing the program with typical input data. This profiling data is then visualized to guide the designer in the partitioning process. Partitioning is done at the method level of granularity using a graphical user interface. Functions which are to be implemented in hardware are synthesized using high-level and logic synthesis tools. Previously designed hardware components are accessible through a database of parameterizable VHDL components. More information on the individual synthesis steps and the automatic generation of the hardware/software interface can be found in [1]. After co-synthesis, Java bytecode for all methods of the initial specification is stored in the pool of software methods. For all methods which are candidates for implementation in reconfigurable hardware, the FPGA configuration data as well as interface information is stored in the pool of hardware methods. The target hardware platform consists of dynamically reconfigurable FPGAs (DPGAs). These new FPGA architectures may be partially reconfigured at run-time, i.e. a portion of the chip can be reprogrammed while other sections are operating without interruption. The target software platform for system prototyping is currently a Linux PC.

2. Hardware/Software Runtime Management

For controlling the dynamic behavior of the reconfigurable system during execution, a run-time manager has been implemented. The run-time manager schedules methods for execution either as software on the Java virtual machine (JVM) of the host processor or as hardware on the reconfigurable DPGA hardware. The scheduling depends on the dynamic behavior of the application and on the current partitioning table chosen by the designer. In contrast to traditional prototyping systems, execution on this platform is a highly dynamic process. The execution flow of the hardware/software system is dominated by the software part. Software methods are executed on the JVM. Whenever the control flow reaches a hardware method, the run-

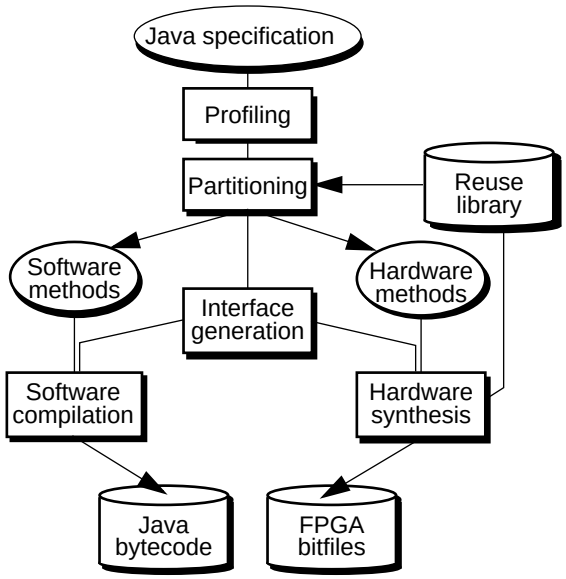

Fig. 1. Specification and co-synthesis

time system determines whether the appropriate configuration file has already been downloaded. If not, then the manager chooses a DPGA and starts configuration. If there is already a DPGA configured with the desired functionality, or if only partial reconfiguration is necessary, the address and parameters of the communication channel to the target DPGA are loaded.

Extending the Java Virtual Machine. As shown in figure 2, the core component of the run-time environment is the Java virtual machine. It basically consists of a class loader for dynamically loading Java bytecode and an execution engine for interpreting these bytecodes on the host processor. In our design framework we use the KAFFE JVM [2], as it comes with complete source code. As our execution framework integrates reconfigurable DPGA hardware, several extensions to the class loader and the interpreter became necessary. The class loader was extended for reading in the current hardware/software partitioning table and for handling hardware methods, i.e. methods which have to be executed on the DPGA board. These hardware methods and information about their corresponding interfaces which are necessary to transfer data to and from the DPGA are accessed through a database.

The execution engine needs to know whether a method is to be interpreted as bytecode or executed in hardware. Therefore, the class loader assigns a special flag to every hardware method. During execution of the application, the interpreter has to activate the hardware call module whenever flow of control reaches a hardware method. Depending on the current state of the DPGA board, several actions are triggered with every hardware call: If necessary, new configuration data is downloaded to the chip; input data is transferred to the board; the hardware design is executed and the resulting data is transferred back to the calling thread. These procedures are implemented in the hardware wrapper (figure 2). Furthermore, a strict synchronization mechanism has been implemented. Currently, only one thread at a time is allowed to access the DPGA board.

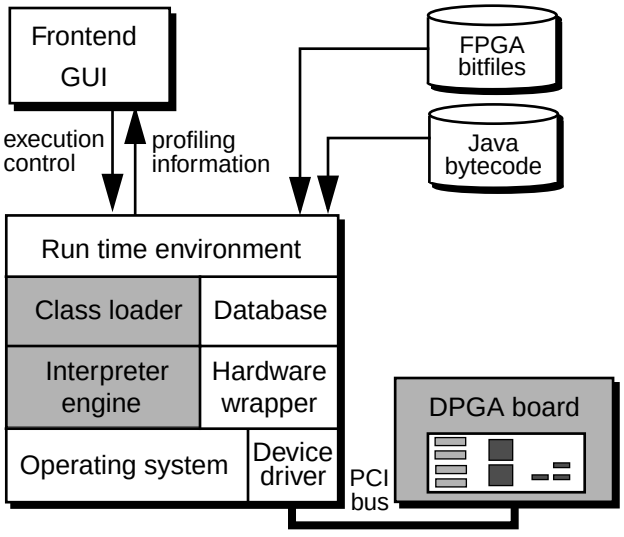

Fig. 2. Design exploration platform

Interfacing with Native Methods. By extending the Java virtual machine for interaction with reconfigurable hardware resources, we were able to implement a completely user-transparent mechanism for execution of a mixed hardware/software implementation. In the prototyping phase, the user can explore design alternatives and different hardware/software mappings via a graphical user interface. The Java code remains unchanged, as the runtime system and the extended interpreter completely manage execution of hardware and software components. However, the drawback of this approach is that only Java virtual machines can be used where the source code is available. For different VMs or newer releases the above mentioned extensions and customizations have to be made.

Therefore, we provide a second alternative implementation where we use a Java class and native methods for interfacing with the hardware part of the system. The main focus is on implementing all necessary functionality for hardware interfacing and reconfiguration in Java. Therefore, the platform specific API of the reconfigurable hardware board can be kept very small. In this case the board API basically consists of native functions to write a value to and read a value from a certain address of the board. These functions are implemented via the *Java Native Interface (JNI)*. That means, all methods for managing the reconfiguration process and execution are implemented in Java and all communication to the hardware board is based on the native implementations of the read and write functions. For communicating with the external board via the PCI bus, a dedicated device driver has been implemented as a kernel loadable module under Linux.

The benefits of this approach are clear. The Java VM does not have to be modified and the hardware interface is clearly defined within the Java language. This means the designer has complete control over all methods for accessing and managing the reconfigurable hardware. The drawback is that the application code has to be modified. The DPGA interface class has to be included in the application source and the designer has to call the appropriate functions for using the DPGA. However, this methodology can be used with any virtual machine. Therefore, it is relatively simple to integrate and test different commercial implementations of the JVM.

References

[1] J. Fleischmann, et. al.: A Hardware/Software Prototyping Environment for Dynamically Reconfigurable Embedded Systems. In Int. Workshop on Hardware/Software Codesign (CODES), 1998.

[2] Transvirtual Technologies, Kaffe Open VM, http://www.transvirtual.com/kaffe.html, 1998.