Application Experiences with a Real-Time Java Processor

Martin Schoeberl [*]

[*] *Institute of Computer Engineering, Vienna University of Technology, Austria (e-mail: mschoebe@mail.tuwien.ac.at).*

Abstract: In this paper we present three different industrial real-time applications that are based on an embedded Java processor. Although from different application domains all three projects have one topic in common: communication. Today's embedded systems are networked systems. Either a proprietary protocol is used due to legacy applications or for real-time aspects or standard Internet protocols are used. We present the challenges and solutions for this variety of protocols in small, memory constraint embedded devices.

1. INTRODUCTION

Embedded systems in the control and automation domain become more and more complex. In the desktop and server domain the increasing complexity of systems can be handled by advanced programming languages. Java has been seen as a very productive object-oriented language. In this paper we show that Java can also be used in the embedded domain. We present three real-world embedded applications written entirely in Java. The applications are deployed on the Java processor JOP.

Furthermore, most embedded systems are implemented as distributed systems and even very small and memory constraint devices need to communicate. In control applications this communication has to be performed under real-time constraints. We show in this paper different communication systems that are all based on simple communication patterns.

The paper is organized as follows: in the remainder of the introduction we present the Java processor JOP and related work on Java processors. Section 2 gives an overview of the three industrial projects. In Section 3 we describe the challenges and solutions for different communication requirements. We extract common patterns from the three applications and present them in Section 4 followed by a discussion in Section 5 and the conclusion in Section 6.

1.1 The Java Processor JOP

All described projects are based on an embedded Java processor, called JOP, see Schoeberl (2005, 2007a). JOP executes Java bytecodes, the instruction set of the Java virtual machine (JVM) native. Therefore, no compiler with the associated memory overhead or an interpreter with the runtime overhead is necessary. The main focus of the design of JOP is on time-predictable execution of those Java bytecodes. As a result the execution time is known cycle accurate and JOP is an easy target for worst-case execution time (WCET) analysis as has been shown by Schoeberl and Pedersen (2006) and Harmon and Klefstad (2007).

JOP is implemented in a field-programmable gate array (FPGA). The logic resource consumption is configurable in the range of 2000–3000 logic cells (LC). That size is 1/3 of the soft-core RISC processor LEON, see Gaisler (2002), that is used by ESA for space missions.

Implementation of a processor in an FPGA is a little bit more expensive than using an ASIC processor. However, additional application logic, such as a communication controller or an AD converter, can also be integrated into the FPGA. Integration of the processor and the surrounding logic in the same reprogrammable chip is a flexible solution: one can even produce the PCB before all logic components are developed as the interconnection is programmed on-chip and not routed on the PCB. For low-volume projects, as those presented in this paper, this flexibility reduces development cost and therefore outweighs the cost of the FPGA device. It has to be noted that low-cost FPGAs, that are big enough for JOP, are available at $11 for a single unit.

1.2 Related Work

Several embedded Java processors are available from industry and academia. aJile's JEMCore, see aJile (2000) and Hardin (2001), and the Cjip processor, see Halfhill (2000) and Imsys (2004), are ASIC versions of Java processors. The first Java processor available was Sun's picoJava, see O'Connor and Tremblay (1997). The ASIC version of picoJava was not a real success and Sun released the Verilog source as open-source. Puffitsch and Schoeberl (2007) have implemented picoJava in an FPGA. As the design was originally targeted for an ASIC and not an FPGA it is quite big: 27500 LCs or about 9 times bigger than JOP.

Kreuzinger et al. (2003) present Komodo, a research Java processor for multithreaded real-time systems. Komodo is now renamed to jamuth, see Uhrig and Wiese (2007), and targeted for commercial embedded applications with Altera FPGAs. Beck and Carro (2003) designed FemtoJava, a research project to build an application specific Java processor. Komodo and FemtoJava are quiet similar to JOP: simple bytecodes are supported by the processor

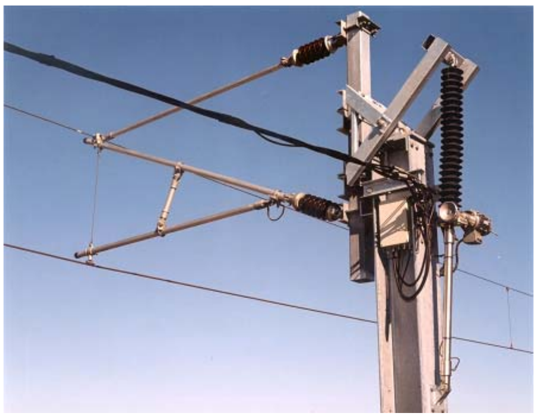
Fig. 1. A *Kippfahrleitung* mast in down position

pipeline, more complex are implemented by the execution of microcode or with a software trap. Their resource usage is also in the same range.

The jHISC project, see Tan et al. (2006), proposes a high-level instruction set architecture for Java. However, the resulting design is probably not very well balanced. The processor consumes 15500 LCs compared to about 3000 LCs for JOP.

2. THE PROJECTS

Since the start of the development of JOP in late 2000 it has been successfully deployed in several embedded control and automation systems. The following section highlights three of those projects.

2.1 The Kippfahrleitung

The first commercial project where JOP had to prove that a Java processor is a valuable option for embedded real-time systems was a distributed motor control system.

In rail cargo, a large amount of time is spent on loading and unloading of goods wagons. The contact wire above the wagons is the main obstacle. Balfour Beatty Austria developed and patented a technical solution, the so-called *Kippfahrleitung*, to tilt up the contact wire. Fig. 1 shows the construction of the mechanical tilt system driven by an asynchronous motor (just below the black tube). The little box mounted on the mast contains the control system. The black cable is the network interconnection of all control systems. In Fig. 2 the same mast is shown with the contact wire tilted up.

The contact wire is tilted up on a distance of up to one kilometer. For a maximum distance of 1 km the whole system consists of 12 masts. Each mast is tilted by an asynchronous motor. However, the individual motors have to be synchronized so the tilt is performed in a smooth way. The maximum difference of the position of the contact wire is 10 cm. Therefore, a control algorithm has to slow down the faster motors.

Hardware Each motor is controlled by its own embedded system (as seen in Fig. 1) by silicon switches. The system measures the position of the arm with two end sensors and a revolving sensor. It also supervises the supply voltage and the amount of current through the motor. Those values are transmitted to the base station.

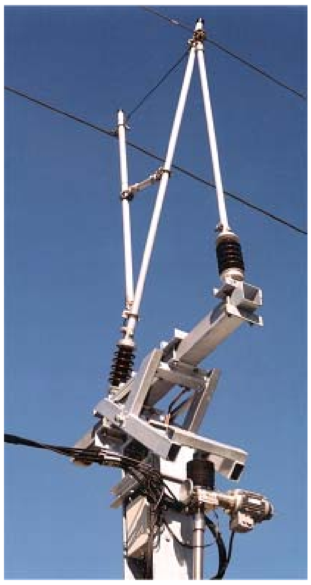
Fig. 2. The mast in the up position with the tilted contact wire

The base station provides the user interface for the operator via a simple display and a keyboard. It is usually located at one end of the line. The base station acts as master and controls the deviation of individual positions during the tilt. In technical terms, this is a distributed, embedded real-time control system, communicating over a shared network. The communication bus (up to one kilometer) is attached via an isolated RS485 data interface.

Although this system is not a mass product, there are nevertheless cost constraints. Even a small FPGA is more expensive than a general purpose CPU. To compensate for this, additional chips for the memory and the FPGA configuration were optimized for cost. One standard 128 KB Flash is used to hold FPGA configuration data, the Java program and a logbook. External main memory is reduced to 128 KB with an 8-bit data bus.

Furthermore, all peripheral components, such as two UARTS, four sigma delta ADCs, and I/O ports are integrated in the FPGA.

Five silicon switches in the power line are controlled by the application program. A wrong setting of the switches due to a software error could result in a short circuit. Simple logic in the FPGA (coded in VHDL) can enforce the proper conditions for the switches. The sigma-delta ADCs are used to measure the temperature of the silicon switches and the current through the motor.

Software Architecture The main task of the program is to measure the position using the revolving sensor and communicate with the base station under real-time constraints. The conservative style of a cyclic executive was chosen for the application. At application start all data structures are allocated and initialized. In the mission phase no allocation takes place and the cyclic executive loop is entered and never exited. The simple infinite loop, unblocked at constant time intervals, is shown in Listing 1.

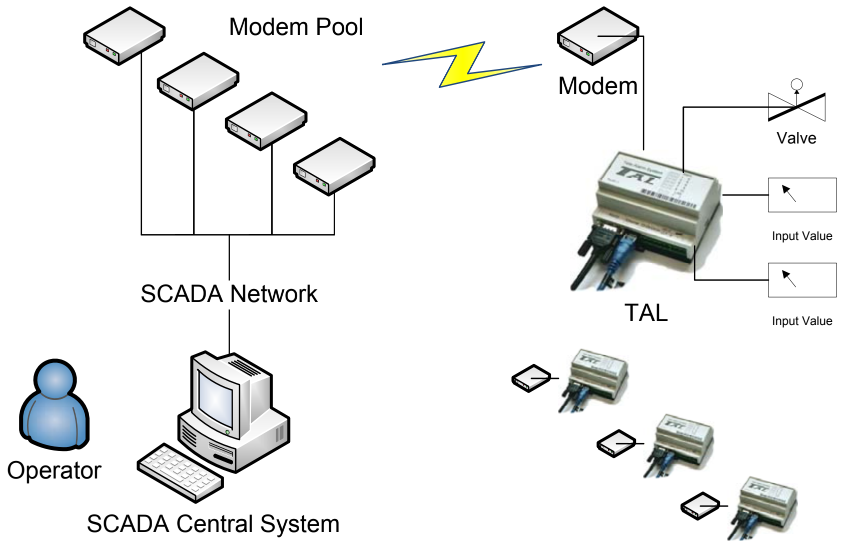

Fig. 3. EVN SCADA system with the modem pool and TALs as remote terminal units

```
private static void forever() {

    for (;;) {
        Msg.loop();
        Triac.loop();
        if (Msg.available) {
            handleMsg();
        } else {
            chkMsgTimeout();
        }
        handleWatchDog();
        Timer.waitForNextInterval();
    }
}
```

Listing 1. The cyclic executive (simplified version)

At the time the application was developed no static WCET analysis tool for Java was available. The actual execution time was measured and the maximum values have been recorded regularly. The loop and communication periods have been chosen to leave slack fur unexpected execution time variations. However, the application code and the Java processor are fully WCET analyzable as Schoeberl and Pedersen (2006) have shown later.

No interrupts or direct memory access (DMA) devices that can influence the execution time are used in the simple system. All sensors and the communication port are polled in the cyclic executive.

Communication Communication is based on a master/slave model. Only the base station (the master) is allowed to send a request to a single mast station. This station is then required to reply within bounded time. The master handles timeout and retry. If an irrecoverable error occurs, the base station switches off the power for all mast stations, including the power supplies to the motors. This is the safe state of the whole system.

In a master/slave protocol no media access protocol is needed. In the case of a failure in the slave that delays a message collision can occur. The collision is detected by a violation of the message CRC. Spurious collisions are tolerated due to the retry of the base station. If the RS485 link is broken and only a subset of the mast stations reply the base station, the base station switches of the power supply for the whole system.

On the other hand the mast stations supervise the base station. The base station is required to send the requests on a regular basis. If this requirement is violated, each mast station switches off its motor. The local clocks are not synchronized. The mast stations measure the time elapsed since the last request from the base station and locally switch off based on a timeout.

The maximum distance of 1 km determines the maximum baud rate of the RS485 communication network. The resulting 12 masts on such a long line determine the number of packets that have to be sent in one master/slave round. Therefore, the pressure is high on the packet length. The data is exchanged in small packets of four bytes, including a one-byte CRC. To simplify the development, commands to reprogram the Flash in the mast stations and to force a reset are included. Therefore, it is possible to update the program, or even change the FPGA configuration, over the network.

2.2 The SCADA Device TeleAlarm

TeleAlarm (TAL) is a typical remote terminal unit of a supervisory control and data acquisition (SCADA) system. It is used by the Lower Austria's energy provider EVN (electricity, gas, and heating) to monitor the distribution plant. TeleAlarm also includes output ports for remote control of gas valves.

Hardware The TAL device consists of a CPU FPGA module and an IO board. The FPGA module contains an Altera Cyclone device, 1 MB static memory, 512 KB Flash, and 32 MB NAND Flash. The IO board contains several EMC protected digital input and output ports, two 20 mA input ports, Ethernet connection, and a serial interface. Furthermore, the device performs loading of a rechargeable

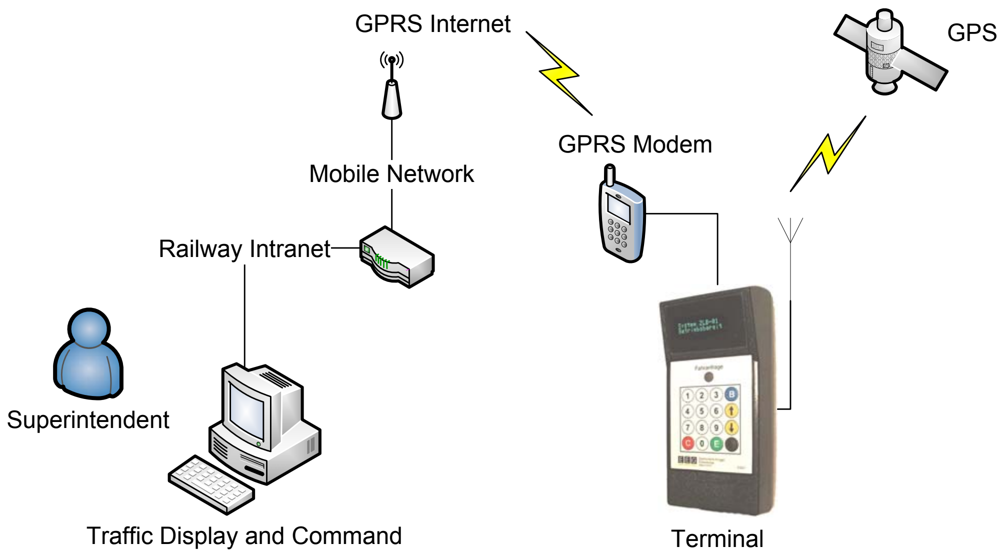

Fig. 4. Support system for single track railway control for the Austrian railway company

battery to survive power down failures. On power down, an important event for a energy provider, an alarm is sent. The rechargeable battery is also monitored and the device switches itself off when the minimal voltage threshold is reached. This event is sent to the SCADA system before the power is switched off.

The same hardware is also used for a different project: a lift control in an automation factory in Turkey. The simple lift control software is now used as a test case for WCET tool development.

Communication The communication between the TAL and the main supervisory control system is performed with a proprietary protocol. On a value change TAL sends the new data to the central system. Furthermore, the remote units are polled by the central system at a regular base. The TAL itself also sends the actual state regularly. TAL can communicate via Internet/Ethernet, a modem, and via SMS to a mobile phone.

EVN uses a mixture of dial-up network and leased lines for the plant communication. The dial-up modems are hosted by EVN itself. For safety and security reason there is no connection between the control network and the office network or the Internet.

Fig. 3 shows the SCADA system setup at EVN. Several TALs are connected via modems to the central modem pool. The modem pool itself is connected to the central server. It has to be noted that there are many more TALs in the field than modems in the pool. The communication is usually very short (several seconds) and performed on demand and on a long regular interval. Not shown in the figure are additional SCADA stations and other remote terminal units from other manufacturers.

2.3 Support for Single Track Railway Control

Another application of JOP is in a communication device with soft real-time properties – Austrian Railways' (ÖBB) new support system for single-track lines. The system helps the superintendent at the railway station to keep track of all trains on the track. He can submit commands to the engine drivers of the individual trains. Furthermore, the device checks the current position of the train and generates an alarm when the train enters a track segment without a clearance.

At the central station all track segments are administered and controlled. When a train enters a non-allowed segment all trains nearby are warned automatically. This warning generates an alarm at the locomotive and the engine driver has to perform an emergency stop.

Fig. 4 gives an overview of the system. The display and command terminal at the railway station is connected to the Intranet of the railway company. On the right side of the figure a picture of the terminal that is connected to the Internet via GPRS and to a GPS receiver is shown. Each locomotive that enters the track is equipped with either one or two of those terminals.

It has to be noted that this system is not a safety-critical system. The communication over a public mobile phone network is not reliable and the system is not certified for safety. The intension is just to *support* the superintendent and the engine drivers.

Hardware Each locomotive is equipped with a GPS receiver, a GPRS modem, and the communication device (terminal). The terminal is a custom made device. The FPGA module is the same as in TAL, only the IO board is adapted for this application. The IO board contains several serial line interfaces for the GPS receiver, the GPRS modem, debug and download, and display connection. Auxiliary IO ports connected to relays are reserved for future use. A possible extension is to stop the train automatically.

Communication The current position of the train is measured with GPS and the current track segment is calculated. The number of this segment is regularly sent to the central station. To increase the accuracy of the position, differential GPS correction data is transmitted to the terminal. The differential GPS data is generated by a ground base reference located at the central station.

The exchange of positions, commands, and alarm messages is performed via a public mobile phone network (via GPRS). The connection is secured via a virtual private network that is routed by the mobile network provider to the railway company's Intranet. The application protocol is command/response and uses UDP/IP as transport layer. Both systems (the central server and the terminal) can initiate a command. The system that sends the command is responsible for retries when no response arrives. The deadline for the communication of important messages is in the range of several seconds. After several non-successful retries the operator is informed about the communication error. He is than in charge to perform the necessary actions.

Besides the application specific protocol a TFTP server is implemented in the terminal. It is used to update the track data for the position detection and to upload a new version of the software. The flexibility of the FPGA and an Internet connection to the embedded system allows to upgrade the software and even the processor in the field.

3. COMMUNICATION

Although we described embedded systems from quite different application domains we have been facing similar challenges. All systems are distributed systems and therefore need to communicate. Furthermore, they are real-time systems (at least with soft deadlines) and need to trust the communication and perform regular checks.

3.1 Challenges

Communication is not per se reliable. The RS485 link at the Kippfahrleitung operates in a rough environment and electromagnetic influences can lead to packet loss. The TAL system can suffer from broken phone lines. The single track control system operates on a public mobile phone network – a network without any guarantees for the GPRS data traffic.

Therefore, we have to find solutions to operate in a safe and controlled manner the distributed system despite the chance of communication errors and failures.

3.2 Solutions

Reliable communication is usually provided by the transport layer, TCP/IP in the case of the Internet. However, the timeouts in TCP/IP are way longer than the communication deadlines within control systems. The approach in all three presented projects is to use a datagram oriented protocol and perform the timeout and retransmission at the application level. To simplify the timeout handling a simple command and response pattern is used. One partner sends a command and expects the response within a specific time bound. The command initiator is responsible for retransmission after the timeout. The response partner just needs to reply to the command and does not need to remember the state of the communication. After several timeouts the communication error is handled by an upper layer. Either the operator is informed (in the SCADA and the railway control system) or the whole system is brought into a safe state (in the motor control project).

4. COMMON PATTERNS

The issues in the design of embedded real-time systems are quite similar in the three described projects. We found that several design patterns are used over and over and describe three of them in this section.

4.1 Master/Slave Designs

Developing safe embedded systems is an exercise in reducing complexity. One paradigm to simplify embedded software development is the master/slave pattern. Usually a single master is responsible to initiate commands to the slaves. The single master is also responsible to handle reliable communication.

The master/slave pattern also fits very well with the command/response pattern for the communication.

4.2 Dealing with Communication Errors

Communication errors are either transient or longer lasting. Transient communication errors are lost packets due to network overload or external electromagnetic influences. In a command/response system the lost packets (either the command or the response) is detected by a timeout on the response. A simple retransmission of the command can correct those transient errors.

A longer network failure, e.g. caused by a wire break, can be detected by too many transmission retries. In such a case the system has to enter some form of safe state. Either the power is switched off or a human operator has to be informed.

The individual timeout values and the number of retries depend, similar to thread periods, on the controlled environment. In the Kippfahrleitung the maximum timeout is in the millisecond range, whereas in the SCADA system the timeout is several minutes.

4.3 Software Update

Correction of implementation bugs during development can be very costly when physical access to the embedded system is necessary for a software update. Furthermore, a system is usually never really finished. When the system is in use the customer often finds new ways to enhance the system or requests additional features.

Therefore, an important feature of a networked embedded system is a software and parameter update in the field. In the first project the software update is performed via a home-made protocol. The other projects use the Internet protocol to some extent and therefore TFTP is a natural choice. TFTP is a very simple protocol that can be implemented within about 100 lines of code. It is applicable even in very small and resource constraint embedded devices.

5. LESSONS LEARNED

Writing embedded control software in Java is still not very common due to the lack of small and efficient implementations of the JVM. Our Java processor JOP is a solution for some embedded systems.

Using Java as the implementation language was a pleasure during programming and debugging. We did not waste many hours to hunt for pointer related bugs. The stricter (compared to C) type system of Java also catches many more programming errors at compile time. However, when using Java in a small embedded system one should not expect that a full blown Java library is available. Almost all of the code had to be written without library support. Embedded C programmers are aware of that fact, but Java programmers are new in the embedded domain and have to learn the difference between a PC and a 1 MB memory embedded system.

Up to date FPGAs in embedded control systems are only used for auxiliary functions or to implement high-performance DPS algorithm directly in hardware. Using the FPGA as the main processor is still not very common. However, combining the main processor with some peripheral devices in the same chip can simplify the PCB layout and also reduce the production cost. Furthermore, a field-reprogrammable hardware device offers a great deal of flexibility: When some part of the software becomes the bottleneck, an implementation of that function in hardware can be a solution. Leaving some headroom in the logic resources can extend the lifetime of the product.

6. CONCLUSION

In this paper we have presented three industrial applications implemented in Java on an embedded, real-time Java processor. All projects included custom designed hardware (digital functions) and the central computation unit implemented in a single FPGA. The applications are written in pure Java without the need for native methods in C. Java proved to be a productive implementation language for embedded systems.

The presented applications contain several nodes and form a distributed embedded system with real-time constraints. In a distributed system reliable communication is essential. We have extracted common patterns (master/slave and command/response) that enable implementation of reliable communication in a constraint environment.

We have attached JOP to a time-triggered network-on-chip, see Schoeberl (2007b). It would be an interesting exercise to implement a JOP based node in a time-triggered distributed system as proposed by Kopetz and Bauer (2003). The combination of a real-time Java processor and a real-time network can ensure real-time characteristics for the whole system.

REFERENCES

aJile. aj-100 real-time low power Java processor. preliminary data sheet, 2000.

Antonio Carlos Beck and Luigi Carro. Low power java processor for embedded applications. In *Proceedings of the 12th IFIP International Conference on Very Large Scale Integration*, December 2003.

Jiri Gaisler. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, page 409, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1597-5.

Tom R. Halfhill. Imsys hedges bets on Java. *Microprocessor Report*, August 2000.

David S. Hardin. Real-time objects on the bare metal: An efficient hardware realization of the Java virtual machine. In *Proceedings of the Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*, page 53. IEEE Computer Society, 2001. ISBN 0-7695-1089-2.

Trevor Harmon and Raymond Klefstad. Interactive back-annotation of worst-case execution time analysis for java microprocessors. In *Proceedings of the Thirteenth IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, August 2007.

Imsys. Im1101c (the cjip) technical reference manual / v0.25, 2004.

Hermann Kopetz and Günther Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

J. Kreuzinger, U. Brinkschulte, M. Pfeffer, S. Uhrig, and Th. Ungerer. Real-time event-handling and scheduling on a multithreaded Java microcontroller. *Microprocessors and Microsystems*, 27(1):19–31, 2003.

J. Michael O'Connor and Marc Tremblay. picoJava-I: The Java virtual machine in hardware. *IEEE Micro*, 17(2): 45–53, 1997. ISSN 0272-1732.

Wolfgang Puffitsch and Martin Schoeberl. picoJava-II in an FPGA. In *Proceedings of the 5th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2007)*, pages 213–221, Vienna, Austria, September 2007. ACM Press.

Martin Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, doi:10.1016/j.sysarc.2007.06.001, 2007a.

Martin Schoeberl. *JOP: A Java Optimized Processor for Embedded Real-Time Systems*. PhD thesis, Vienna University of Technology, 2005.

Martin Schoeberl. A time-triggered network-on-chip. In *International Conference on Field-Programmable Logic and its Applications (FPL 2007)*, pages 377 – 382, Amsterdam, Netherlands, August 2007b.

Martin Schoeberl and Rasmus Pedersen. WCET analysis for a Java processor. In *Proceedings of the 4th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2006)*, pages 202–211, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-544-4.

Y.Y. Tan, C.H. Yau, K.M. Lo, W.S. Yu, P.L. Mok, and A.S. Fong. Design and implementation of a java processor. *Computers and Digital Techniques, IEE Proceedings-*, 153:20–30, 2006. ISSN 1350-2387.

Sascha Uhrig and Jörg Wiese. jamuth: an ip processor core for embedded java real-time systems. In *JTRES '07: Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems*, pages 230–237, New York, NY, USA, 2007. ACM Press.