

# A Tool for Teaching and Research on Computer Architecture and Reconfigurable Systems

Christof TEUSCHER<sup>1</sup>    Jacques-Olivier HAENNI<sup>1</sup>    Francisco J. GÓMEZ<sup>2</sup>  
Héctor Fabio RESTREPO<sup>1</sup>    Eduardo SANCHEZ<sup>1</sup>

<sup>1</sup>Logic Systems Laboratory, Swiss Federal Institute of Technology  
CH – 1015 Lausanne, Switzerland  
E-mail: {name.surname}@epfl.ch

Fax: ++41 21 693 37 05, URL: <http://lslwww.epfl.ch>

<sup>2</sup>Escuela Técnica Superior de Informática, Universidad Autónoma de Madrid  
E – 28049 Madrid, Spain  
E-mail: [francisco.gomez@ii.uam.es](mailto:francisco.gomez@ii.uam.es)

## Abstract

*Labomat 3 is a reconfigurable platform for teaching and research purposes developed by our laboratory. In this paper we describe in details the hardware and software of the board as well as three application domains: logic design, computer architecture, and codesign.*

*The main features of the board are: a microprocessor associated with two mid-range FPGAs; a set of powerful software tools going from a real-time operating system to a JavaVM; easy to use design and simulation tools, and last but not least a network interface.*

*The combination of these features makes Labomat 3 a unique teaching tool. Nevertheless, the board may also be used for advanced research: several boards connected together can form a powerful reconfigurable parallel system.*

## 1 Introduction

It is commonly admitted that the reconfigurable feature and the increasing density of FPGA circuits [10] are carrying out a true revolution in the world of digital circuits. After being initially reserved in an almost exclusive way for prototyping tasks, they were very quickly involved in the heart of a great number of different kinds of applications.

Naturally, teaching has also taken advantage of all these possibilities and, as a consequence, several platforms exist. In certain cases the circuit's manufacturers offer these platforms [9, 7]. Strangely, these development systems currently ignore one of the most promising fields: the codesign – basically, the decision

of which parts of an application are to be designed as software and which shall be designed directly as hardware [1].

In the classical university curriculum there is a hard distinction between software and hardware design: the former is under the responsibility of the computer science department, whereas the latter is taught at the electrical engineering (or computer engineering) department. However, with codesign this clear-cut frontier is dissolving, requiring a fundamental change in the engineering curriculum.

The board presented in this paper, Labomat 3<sup>1</sup>, has been developed for use by students in any kind of hardware design courses, from elementary digital circuits to codesign, going through computer architecture and microprocessors.

Indeed, Labomat 3 presents some unique characteristics:

### 1. Hardware:

For the microprocessor and codesign courses, a Motorola 68360 microprocessor and two FPGAs are available on the board. In our computer architecture course, we use the XC4013, a relatively complex FPGA, with which students are able, for example, to implement a complete processor with a simple instruction set and a 5-level pipeline including hazard correction.

### 2. Software:

The board runs a real-time operating system (RTEMS) and a wide set of software tools, including a Java machine. TCP/IP networking is already integrated in the operating system.

---

<sup>1</sup><http://lslwww.epfl.ch/labomat>

### 3. Applications:

A special software—taking advantage of the partial and dynamic reconfiguration characteristic of the XC6200 FPGA family—was developed for the teaching of digital systems. The circuit is directly configured based on a schematic diagram designed in a “traditional” way. There is no need to compile, place, and route. It is also possible to read the values of all the signals and to display them directly in the diagram. This system thus combines the facilities of simulation with the completeness of emulation.

The combination of these features makes Labomat 3 a unique teaching tool. Nevertheless, the board may also be used for advanced research: several Labomat 3 boards connected together can form a powerful reconfigurable parallel system.

The rest of the paper is organized as follows: In part I we give a complete description of the Labomat 3 hardware, software and its communication interfaces. In part II we present three application domains: logic design, computer architecture, and codesign. We conclude our paper by pointing out some considerations on future work.

## Part I

# Labomat 3 Description

The Labomat 3 board is a simplified and modified version of the RENCO (Reconfigurable Network Computer) [2, 8] board developed by our laboratory. Our aim is to offer students a powerful, all-round platform that is easy to understand and simple to use.

## 2 Hardware

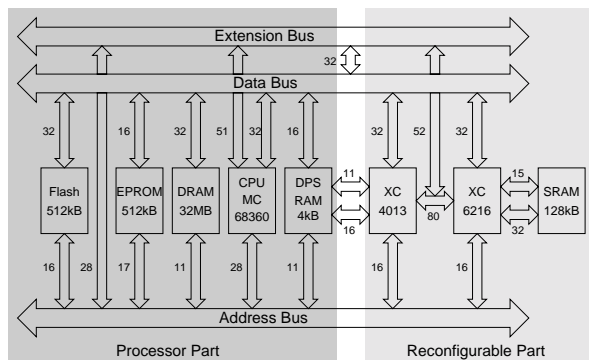


Figure 1: Labomat 3 architecture

The board is divided into two main parts (Figure 1):

1. **the processor part** is built around a Motorola MC68EN360 processor [5]. It is connected to 512 KB of boot EPROM and to a maximum of 32 MB of DRAM (SIMM). An additional flash memory (512 KB) can store FPGA configurations or user data. A 4-KB dual-port SRAM facilitates data sharing between the processor and the reconfigurable part. Ethernet 10Base-T and RS-232 ports are used to communicate with the outside world.

The MC68360 processor was chosen for its integrated communication capabilities. In addition, there is a large amount of free software tools available for this platform.

2. **the reconfigurable part** is built around a XC4013E and a XC6216 FPGA from Xilinx [11]. The XC6216 is connected to 128 KB of additional SRAM which is not directly accessible from the processor. The XC4013E is connected to the dual-port SRAM, that in turn is connected to the processor.

The processor can directly access the FPGAs as peripherals through its data and address buses. A large bus of 80 signals interconnects both FPGAs and a subset (52 I/Os) is directly available on 10-pin connectors. The extension bus also holds the global data, the address bus, and the control bus. Interrupt and bus controlling is done by a programmable logic chip (MAX7128). We managed to hide the timing specifications of bus and FPGA configuration cycles.

JTAG and BDM interfaces serve the dual purpose of programming the on-board bus controller and providing powerful low-level hardware debugging tools.

Every FPGA receives two different clock signals and each signal can be assigned to a different source. An on-board programmable clock generator provides a wide range of clock frequencies. The step-by-step button allows manual clock generation which is particularly useful for debugging and testing designs. Clock signals may also be generated by the CPU. The system's default clock frequency is 25 MHz.

Bootstrap code is executed from the EPROM. This code simply downloads the complete operating system and applications via the Ethernet interface (see section 4).

The Labomat 3 board is implemented on a 6-layer PCB (Figure 2). The XC6216 and the microprocessor are assembled on the back side of the PCB.

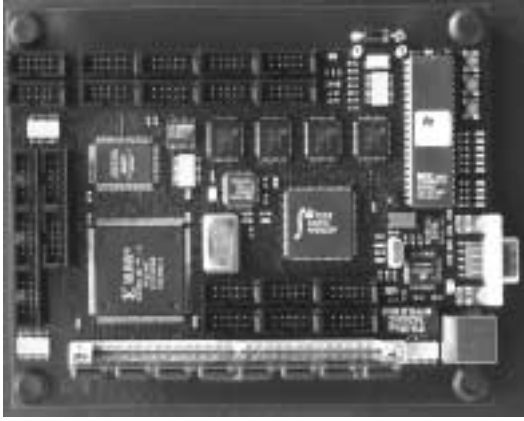


Figure 2: Labomat 3 board

### 3 Software

In order to achieve versatility and user friendliness, we decided to use a whole operating system instead of just a simple abstruse environment.

After examining many possibilities, RTEMS [6], a pre-emptive multitasking operating system with modest memory requirements, was chosen. In addition it contains the drivers for Ethernet and provides a TCP/IP stack. Finally, it has already been adapted for the 68360 processor and its source code is available for free.

Board-specific code, collected together in the *Custom Hardware Library (CHL)*, includes utility functions for accessing the board resources.

On top of this operating system, a Java virtual machine was set up. Java was chosen because it is simple to use, has a standardized API, is robust, network-ready, and allows minimization of platform-dependent code. We chose Kaffe [4] since its source code is available for free and because it has already been ported to the 68000 processor, therefore reducing the amount of work needed to set it up.

In addition to the standard Java API, the user has at his disposal a board-specific API which provides classes and methods for accessing the board resources.

Figure 3 shows the different software layers used in Labomat 3.

The layered structure offers a high level of abstraction and thus helps students to become familiar with the board in a short period of time.

### 4 Interfaces

The Labomat 3 platform offers various means to configure and control the reconfigurable logic and the processor. The user may choose one of the below described interfaces (see Figure 4):

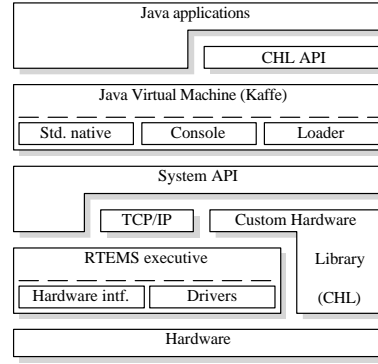


Figure 3: Labomat 3 software architecture

1. The board can be connected by an RS-232 interface. A standard terminal software displays a simple monitor that allows to configure the FP-GAs and to access directly the hardware.
2. Labomat 3 hosts a telnet server offering the same monitor as used for the RS-232 interface. In fact, every Labomat 3 board has assigned a unique hard-coded hardware address that can be mapped to any IP address (domain name).
3. In the future, functionality of the web-server currently running on RTEMS will be increased in order to control all Labomat 3 hardware via a web-based interface. A board connected to the internet could then be configured and programmed from anywhere in the world. Thus, students could work at home and communicate with a board installed in our laboratory.

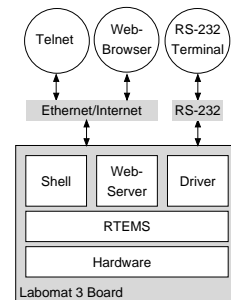


Figure 4: Labomat 3 interfaces the world

Other communication applications could be easily implemented since all basic components, like sockets and the Java virtual machine Kaffe, already exist.

FPGA configuration files are currently read from a tftp server. At the moment, only standard Xilinx formats (.hex and .cal) are supported. Configuration file download is initiated by the means of one of the

aforementioned communication interfaces or directly form C or Java code.

## Part II

# Applications

## 5 Logic Design: A Schematic and Simulation Tool

### 5.1 Introduction

The development of a digital circuit using FPGA technology can be broken down into several stages as shown in Figure 5: (1) the circuit is designed using the editor tool; (2) a logical simulation may then be carried out; (3) the logical elements are mapped to the FPGA and (4) then located and connected (place and route).

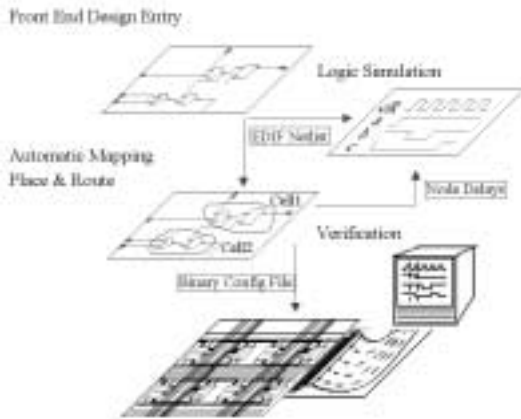


Figure 5: Logic design flow using Xilinx software and a standard schematic editor

The XC6200 FPGA family from Xilinx offers the possibility to configure independently each basic cell, as well as to read and modify the state of the cells.

Our goal is to offer a simple and intuitively to use graphic tool that directly communicates with the FPGA where the placed and routed design is going to be tested (Figure 6).

### 5.2 XC6216 Hardware

Xilinx XC6200 is a family of fine-grained FPGAs: each logic cell can be regarded as a combination of a D flip-flop with a two-control-variables multiplexer. The XC6200 family uses a hierarchical routing system: each level of the hierarchy (unit cells, 4x4 cell blocks, 16x16 cell block, 64x64 cell block, etc.) has its own

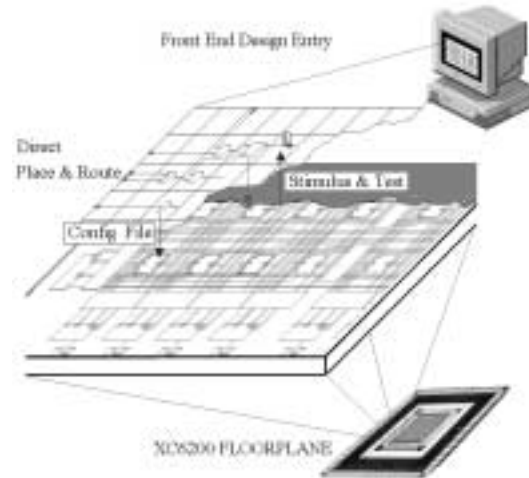


Figure 6: Logic design, online configuration and verification

associated routing resources. At the lowest level, each cell communicates with 4 of its immediate neighbors via 2 uni-directional lines.

All the configuration bits of the circuit of the XC6200 family are organized to a public format at the interior of a SRAM: to modify a cell or its interconnections, it suffices to write the appropriate word onto this memory. Furthermore, it is possible to reach, via this SRAM, all the flip-flops of the logic cells. This SRAM is, in our case, directly mapped into the address space of the 68360 microprocessor.

### 5.3 The Design Tool

#### 5.3.1 General

The *Graphical User Interface (GUI)* of our design tool is similar to other schematic editor tools. It offers the usual functions present in GUIs, such as: cut, copy, paste, zoom-in, zoom-out, and drag-and-drop.

A general layout of the program interface is shown in Figure 7. The window background displays a grid representing the matrix of the 64x64 basic cells of our XC6216 FPGA.

#### 5.3.2 Operation Modes

There exist three different modes in the tool:

**Add Component Mode** allows the insertion of a basic component from a library. The component is selected from a dialog box, then, using the drag-and-drop option, it is placed at the desired position in the design window. The number of cells needed for the selected component are indicated by a rectangle on the grid. A hierarchical cell design with basic cells

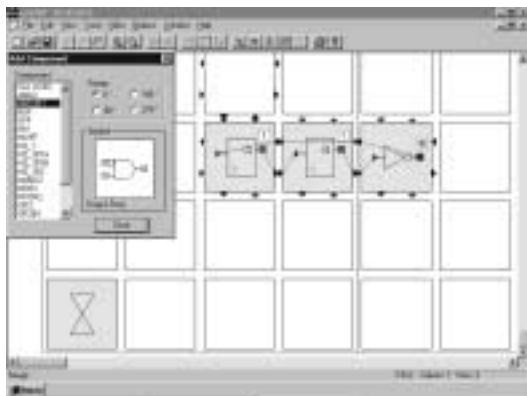


Figure 7: Graphical user interface of the schematic editor

is also possible since the user can create new library elements based on already created cells or basic cells.

**Net Mode** is used to connect the different components. Connections can be made using the contact points defined in each component. Limitations are given by the internal hardware of the basic cell of the XC6216 family and the routing resources of contiguous cells. A component using only a single cell can be connected through the North, South, East, and West connections, with the four neighbor cells. Each 4x4 block of cells also has rapid connection lines (FastLane 4) in the North, South, East, and West directions.

**Normal Mode** is where FPGA configuration and circuit test is initiated. In this mode, component selection, drag-and-drop, cut, copy, paste and delete are also enabled.

### 5.3.3 Communication with the FPGA

The program only communicates via Ethernet with the Labomat 3 board. This allows remote control of the hardware from anywhere in the world (you only have to know the IP address of the board).

The most innovative feature of this tool is the exact reproduction of the configured hardware—the XC6216—in the schematic on the screen. It is not necessary to use any simulation tool since the output values of each component in the design are verified automatically by reading the status of each individual cell (accessible through the register embedded in each cell). The signal values of a cell may also be directly imposed by the user.

## 5.4 A 4x4-bit Multiplier: A Simple Example

As an example, let us show how to design a pipelined 4x4-bit multiplier. A multiplication unit is

replicated in a regular structure intercalating registers to form a sort of pipeline.

**First step:** define the basic 1-bit multiplier as a new library element. It is implemented with a logic AND-gate and a full adder. After the basic components are placed, connections should be made as shown in figure 8. To create a new component, all cells that need to be included have to be selected. With the *Create Component* option, a name and a symbol is associated to the group of selected cells. Automatically, input and output pins are chosen from among the pins that have been left unconnected on the selected group of cells. The new element is then added to the current library and can later be used as a standard element.

**Second step:** to obtain a 4x4-bit multiplier, the 1-bit multiplier needs to be connected as shown in figure 8. The previous steps do not require a connection with the Labomat 3 board.

**Third step:** *Connect* establishes a connection with the Labomat 3 to configure and test the design on the board. The *Send Data to Xilinx* option generates a configuration file in the CAL format (Xilinx). This ASCII file is a sequence of addresses and data, that configures the multiplexers of each functional unit. The configuration file is sent via Ethernet to the Labomat 3 board.



Figure 8: Pipelined 4x4-bit multiplier

**Fourth step:** once the FPGA is configured, successive clock pulses can be sent to the FPGA. At each clock cycle, the logical value of the basic cell's output is updated and shown directly on the schematic. It is thus a very convenient way to debug the design.

## 6 Computer Architecture: A Simple Pipelined Processor

### 6.1 Introduction

This chapter presents the implementation of a simple pipelined processor that is entirely run on the Xilinx XC4013E FPGA. Students are not only able to

simulate the processor, they may go a step further and implement their own design using real hardware. The aim is not to implement a high performance processor but rather to examine difficulties such as hazards and exceptions, that appear naturally when using a pipelined processor architecture. As such, it is more of an academic application.

## 6.2 Specification

The processor specification and implementation are partially inspired by the DLX architecture [3]: it is an 8-bit load-store architecture with four internal 8-bit general purpose registers, named R0, R1, R2, and R3; unlike the DLX architecture, the value of our register R0 is not always zero. There are no floating point registers available and the only supported data types are 8-bit integers.

The following eight instructions have been defined:

```

LOAD      Rd ← M[addr]
STORE     M[addr] ← Rs
MOVE      Rd ← Rs
SUB       Rd ← Rs1 - Rs2
ADD       Rd Rs1 ← Rs2
CMP       flag ← Rs1 - Rs2
BRANCH   if flag then PC ← PC + offset
JMP      PC ← PC + offset

```

Rd, Rs, Rs1 or Rs2 are any of the four general purpose registers.  $M[addr]$  stands for memory access at address  $addr$ . No instruction has more than one result.

A 5-stage pipeline is used and every pipeline stage is executed in one clock cycle:

```

IF      Instruction fetch cycle
ID      Instruction decode cycle
EX      Execution cycle
MEM     Memory access cycle
WB      Write back cycle

```

## 6.3 Instruction Format

Like most recent RISC machines, instructions have a typical, fixed-length encoding for pipeline efficiency. The only available addressing mode is absolute addressing for load and store instructions and PC-relative addressing for branch and jump instructions (Figure 9). All instructions have a 3-bit opcode and the addressing mode is directly encoded in the instruction.

Any of the four general purpose registers may be loaded or stored without restriction.

## 6.4 Processor Architecture

To avoid pipeline stalls and to simplify input-output mapping of data to the real I/O-ports, instruc-

Instr.	OPC			Rd	Rs	Rd	Rs1	Rs2	offset
	15	14	13						
LOAD	0	0	0	Rd					address
STORE	0	0	1	Rs					address
MOVE	0	1	0	Rs	Rd	0	0	0	0
SUB	0	1	1	Rd	Rs1	Rs2	0	0	0
ADD	1	0	0	Rd	Rs1	Rs2	0	0	0
CMP	1	0	1	0	Rs1	Rs2	0	0	0
BRANCH	1	1	0	0					offset
JMP	1	1	1	0	0				offset

Figure 9: Instruction format

tion memory and data memory are separated (Harvard architecture). Instruction fetch (IF stage) and memory access (MEM stage) can be done simultaneously without stalling the pipeline. This greatly simplifies the hazard detection unit.

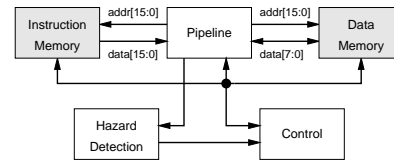


Figure 10: Processor architecture

The hazard detection unit works closely together with the control unit. The five functional blocks in figure 10 are implemented as separate architectures in VHDL.

## 6.5 Pipeline

In order to proceed at the rate of one clock per instruction (CPI), the pipeline has to be able to execute any combination of instructions. To avoid structural hazards, the pipeline shown in Figure 11 has no resource conflicts: every instruction can execute simultaneously and overlap with any other instruction. One out of the four internal registers (R0, R1, R2, R3) is selected by an internal multiplexer for each output (oa, ob, oc). Register values are modified on the raising clock edge during the write-back stage (WB) and read during the falling clock edge. Each temporary register bank holds the values between the clock cycles (dark gray registers in Figure 11). These registers are synchronously loaded (control signals LREGIF, LREGID, LREGEX, LREGMEM) on the falling clock edge if no pipeline stall occurs.

As a result of the CPI being 1, the PC has to be incremented in the instruction fetch stage. Three multiplexers (MA, MB, MC) allow data bypassing. D1 and D2 are decoders which directly control the operations of the ALU and of the MUXWB. The operations depend on the instructions in the IR registers. The sign extension block in the ID-stage extends the offset and the absolute address coded in the instructions on eleven bits to a 16-bit address.

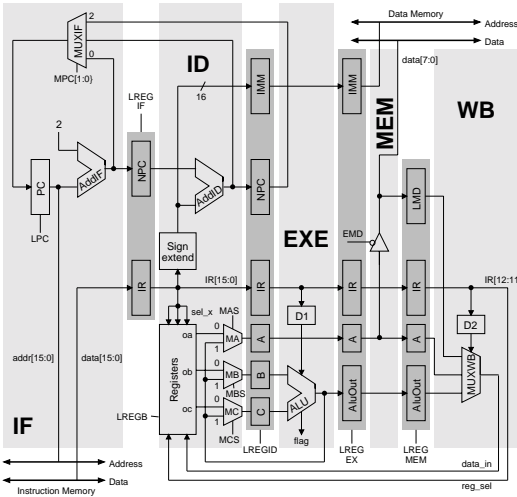


Figure 11: Pipeline

## 6.6 Hazard Detection

The hazard detection unit only accepts input from the instruction fetch register. Since the pipeline has no structural hazards, only data and control hazards have to be detected. Detecting control hazards is simple: a *jump* instruction always stalls the pipeline for one clock cycle, a *branch* instruction for two clock cycles, independently of whether the branch is taken or not.

In order to detect data hazards, a very simple solution has been adopted: the number of a destination register (which will be modified at a later stage in the pipeline) is entered during instruction fetching in a four-word FIFO queue. For example, consider the following *move* instruction:

```
MOVE R2 ← R3
```

This instruction will modify the general purpose register number 2 in the write-back stage. Therefore the number 2 is entered in the FIFO queue. Elements in the queue are moved at each clock cycle. Each time a new instruction is loaded into the instruction fetch register, a test is carried out to see if the number of the register used as the source (3 in the above *move* instruction) is present in the FIFO queue. If it is present, the pipeline stalls until the concerned number leaves the FIFO queue (in other words, the register has been modified). If it is not present, instruction fetching can continue without stalling. In the worst case a data hazard causes a 4-cycle stall, as illustrated by the following example:

```
MOVE R2 ← R3
MOVE R0 ← R2
```

The second instruction can not be carried out before the first instruction has completed. The number 2 (register modified by the first instruction) will remain in the FIFO queue for 4 clock cycles. Clearly, a more

sophisticated bypassing could improve performance.

The bypass in figure 11 is implemented, but currently not used by the control unit (Figure 10).

## 6.7 Implementation

The processor is completely implemented in VHDL using the FPGA-Express synthesizer. A simulation of the instruction memory simplifies access cycles. A more complicated implementation respecting access timings could interface the dual-port SRAM connected to the XC4013E FPGA and use it as instruction memory. In this case, the microprocessor may write and change the instruction code executed by the RISC processor. Data memory is mapped to eight Labomat 8-bit extension buses connected to switches and LEDs. Using the step-by-step button, debugging and verification of the program code can easily be carried out.

The implementation on a XC4013E FPGA uses 296 CLBs out of 576: enough hardware resources are still available for future extensions and improvements. The design runs without problems at a clock frequency up to 25 MHz.

## 7 Codesign: A Floating-Point Coprocessor

This section describes the design of a simple floating point coprocessor (FPU). Students are asked to design a floating-point multiplier using logic gates, to simulate and finally test it on Labomat 3's hardware (Figure 12). This implies the design of an FPGA configuration that implements the multiplier and interfaces with the processor. In addition, some code for the processor has to be written to interface and test the FPU.

The processor can access the FPGA just like a peripheral using its data and address buses. The FPGA does not have to generate complex acknowledge signals because the processor is interfaced by glue logic hiding the bus operations.

The processor will write both operands at given addresses in the FPGA, will wait for the calculation to be completed (either by polling a status register of the FPGA or using the interrupts), and will read the result from a given address.

The FPU is developed using the WorkView Office environment on Windows-NT PCs. It is mainly composed of combinatorial logic controlled by some state machines responsible for the sequencing of micro-operations. We can distinguish three basic modules: (1) the computation of the sign, (2) the mantissa and (3) the exponent.

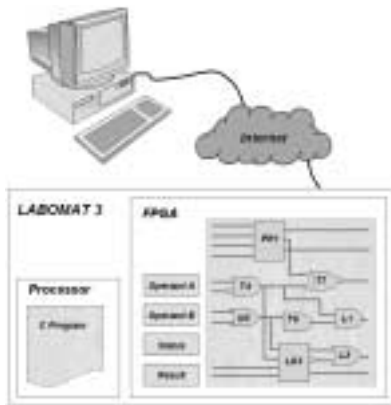


Figure 12: Labomat 3 as a codesign tool

The software, written in C, is simply a built-in routine call to configure the FPGA, then the processor enters a loop, reading two operands from the user terminal and sending them to the FPU. Once the floating-point computation has completed, the result is read back and written to the user terminal.

## 8 Conclusion

The combination of programmable resources, high-level software tools and networking, Labomat 3 provides a system that can be used to experiment with the design of dedicated systems comprising both hardware and software, as well as to explore different interfaces between the two parts.

Labomat 3 can be used to develop and validate automatic codesign tools, and—since its architecture is conceptually simple—the designer can concentrate his or her effort on the partitioning and synthesis without having to deal with the complexity of the target platform.

Our students' laboratory is currently equipped with 50 Labomat 3 boards and they have been successfully used for student exercises. It is possible for students to go beyond simulation, testing their designs with real hardware, thus allowing them to face problems that do generally not appear in simulation. Furthermore, they are able to experience the satisfaction of obtaining a *bona fide* hardware system.

For a near future, we are working on tools to assemble and use all the 50 boards in a huge reconfigurable cluster (100 FPGAs) for high performance computing experiments. Communication between the boards is possible through Ethernet or fast direct communication lines that connect the boards in a given structure (ring, star, etc.). Experiments with digital neural networks, genetic algorithms, and artificial life are also planned.

## Acknowledgments

We are grateful to André Badertscher for the board photograph and assembly. This work was supported in part by a grant from the Werner Steiger Foundation.

## References

- [1] L. Garber and D. Sims. In pursuit of hardware-software codesign. *IEEE Computer*, 31(6):12–14, June 1998.
- [2] J.-O. Haenni, J.-L. Beuchat, and E. Sanchez. Renco: A reconfigurable network computer. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 288–289, Napa USA, April 1998. IEEE.
- [3] J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1990.
- [4] Kaffe. *A free virtual machine to run Java code*. <http://www.kaffe.org>.
- [5] Motorola. Quad integrated communication controller. Technical report, Motorola, 1993.
- [6] Oarcorp. *RTEMS Real-Time Executive for Multiprocessor Systems*. <http://www.rtems.com>.
- [7] Z. Salcic and A. Smailagic. *Digital systems design and prototyping using field programmable logic*. Kluwer, 1997.
- [8] E. Sanchez, M. Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer, and A. Perez-Urbe. Static and dynamic configurable systems. *IEEE Transactions on Computers*, Special Issue on Configurable Computing, June 1999.
- [9] D. van den Bout. *The practical Xilinx designer lab book*. Prentice Hall, 1998.
- [10] J. Villasenor and W. H. Mangione-Smith. Configurable computing. *Scientific American*, 276(6):54–59, June 1997.
- [11] Xilinx Inc. XC6200 Advanced product specification, The Programmable Logic Data Book 1997.