Reprogramable hardware for educational purposes

Michael Gschwind

mike@vlsivie.tuwien.ac.at

Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3-182-2
A-1040 Wien
AUSTRIA

Abstract

This paper presents a novel idea in teaching computer architecture by using programmable hardware. Current teaching models for computer architecture today are either mostly theory-only or implementation oriented. Theory-based architecture courses lack the feedback to show students the effects of their decisions. Implementation-oriented instruction emphasizes the implementation aspects, that is, very low-level implementation strategies, over CPU architecture and forces the usage of very limited CPU designs to reduce complexity. High cost and long manufacturing times are other problems associated with this approach.

We propose to use field programmable gate arrays (FPGAs) to allow fast implementation of chip designs. This allows for a fast debug cycle, as designs can be altered and downloaded in a matter of hours. As FPGAs are pretested, only logic functionality has to be validated, reducing the time to get a workable implementation of a chip considerably.

1 Introduction

Current computer science instruction emphasizes computer architecture theory over implementation. Reasons for this are manifold: Students often lack the engineering background for hardware implementation, as in many countries, computer science curricula were developed from a mathematical background. Implementation of architectures on a transistor basis is arduous, offers little excitement and is error-prone. Thus manufactured chips also require extensive testing effort and multiple iterations to produce working silicon.

Before the advent of VLSI, building architectures was comparatively easier to do in a classroom environment: computers could be assembled from discrete logic using wire wrapping or printed boards. Anybody possessing basic electronic skills could theoretically assemble his own architecture. Also, for these circuits, testing was less involved than it is today for VLSI ICs.

One approach that has been tried is high level compilation, that is the specification of the computer architecture in a high level language and automatic compilation to silicon [GGE91]. However, few tools[1] exist. Moreover, circuit density is suboptimal and often these tools offer only limited parameterization capabilities to already well-defined blocks. This severely limits the architectural space that can be explored with such an approach.

Low level implementation is feasible only for small, limited architectures. These courses generally limit themselves to teaching *how* well-defined architectures can be implemented. Anything going beyond that requires a massive effort on the order of several master's theses.

The results are that computer architecture is often regarded as a very unreal discipline that offers little excitement and few 'real' results. We feel that one way to spark interest in future generations of computer architects is to put the excitement back into computer architectures. This can be done by allowing students to design their own CPUs and actually use them. The concept to facilitate such an undertaking is using programmable hardware. Students can design their chips and program hardware to implement them.

2 FPGAs

User-programmable Logic Devices (PLDs) are a technology which has developed rapidly over the past decade. The original concept was developed about twenty years ago, with PROMs and EPROMs being the first members of this family. Further development brought the implementation of TTL-logic using PLAs and PALs and, finally, field programmable gate arrays.

There are several types of field programmable gate arrays. The main differences are the type of logic cells which are offered (from complex look-up tables to a sea of NAND gates) and how these cells can be programmed (from one-time programmable over EPROM-based to RAM-based). For educational purposes, readily re-programmable circuits are required, as designs need to be readily changeable. For this

[1] One such high-level synthesis tool was Genesil. It was retracted from the market in 1992 by Mentor Graphics.

SIGSCE 94- 3/94, Phoenix, Arizona,USA

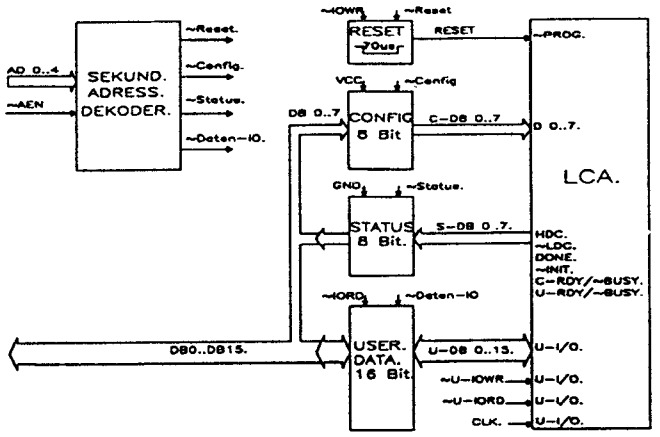

Figure 1: Block diagram of the the FPGA/AT-bus interface

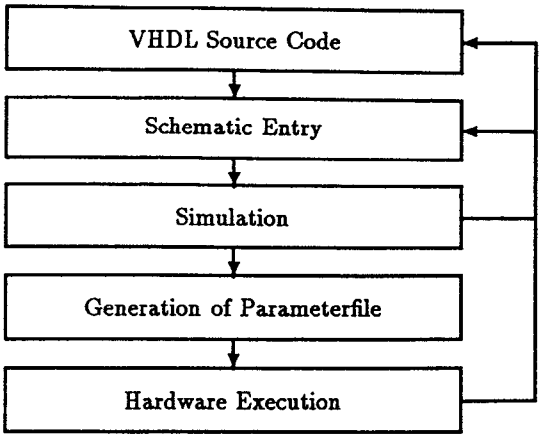

Figure 2: Programming hardware

reason, we chose Xilinx Corp.'s FPGAs [Xil92]. These are SRAM based and can be re-programmed at any time by asserting the RESET signal and latching the new configuration information. Xilinx's FPGAs use logic cells which basically contain three lookup tables and two flip flops each.

3 Our Platform

To make FPGA technology a 'user-friendly' programmable hardware platform, we decided to design a PC board and associated software that would simplify programming of and interaction with the hardware[Hub92].

The PC board consists of two programable FPGA chips, a Xilinx XC4005 and a Xilinx XC3020, and glue logic to facilitate communication with the PC's CPU over the AT bus. The glue logic implements several I/O space mapped status registers (see table 1). These registers allow user software to reset the FPGAs, program them with a new configuration and communicate with the programmed hardware (see figure 1).

4 How to Program Hardware

To program the FPGA hardware, the circuit design is entered on a schematic level, using ViewLogic. Several macro libraries exist, offering TTL components and other macro cells. After design entry, the design can be simulated, using the ViewLogic Simulator, and, once the design has been verified, a parameter file is generated for the FPGA. This file is transferred to the host for the FPGA board, from where it is down loaded to the FPGA board. A MS Windows interface offers a user-friendly method of controlling the hardware.

In future we hope to add an additional layer on top of the ViewLogic schematic entry, which allows using VHDL source to specify the design (see figure 2).

5 Usage for Education

Our approach with programable hardware allows designs to be entered on a schematic level, using tools such as ViewLogic. With the appropriate libraries, different implementation approaches can be taught, such as TTL logic or using larger macro libraries. Once the design has been entered, it can be simulated on a workstation, or a parameter file for downloading on the hardware can be generated.

The advantage of this approach is that no re-wiring is required if the circuit is changed. A simple compile run as is customary in software design suffices to modify the hardware. This facilitates experimentation with different approaches and reduces the reluctance to modify the design, as little additional work is necessary. Also, it increases the number of exercises that can be done in a given time period, as startup time normally necessary for teaching the use of tools and overhead for re-wiring, or generating printed boards is optimized away.

Because changing the circuit is as easy as a software edit-compile-run cycle, debugging is very easy. Suspected bugs can be fixed immediately and the testing can be continued. Also, instrumentation for debugging is easy: for example, if a particular bus is suspected to carry a faulty signal, it is extremely easy to add a latch to store the value carried by the bus. The host computer can then be used to read the latch and once the problem has been corrected, the hardware added for debugging purposes can just as easily be removed.

An additional benefit of FPGA-based architecture instruction is that if the logic design works, the hardware is guaranteed to work. Thus, potentially cumbersome electrical problems are removed, allowing to concentrate on computer architecture, not on debugging its implementation.

6 Experiences

Tests with our prototype have largely been satisfactory. Our tests included the implementation of a full microprocessor,

Port	Function	Description
300H	CReset	Writing to this address will reset the FPGA.
302H	CConfig	Used to download the configuration file.
304H	CStatus	Status Register, used during configuration and for handshaking by user applications
306H	CIO	Bidirectional register for transferring 16 bit data between FPGA application and CPU.

Table 1: Communication registers of the FPGA board are mapped into the CPU's I/O space.

the PIC16C57 ([GJ92]), with debugging control, and of a stack machine.

Currently, we use ViewLogic to enter designs at the schematic/function block level. The ViewLogic environment offers a wide array of cell libraries, including macros modeling TTL parts and more advanced function blocks. Thus, students can concentrate on architecture design, without having to spend a large portion of their time designing low-level function blocks. The TTL emulation library allows student familiar with conventional board design to work much more efficiently, as their existing expertise in this design style can be applied.

For teaching hardware description language and logic synthesis classes, we intend to use VHDL compilers (we currently have both the Synopsys and Racal Redac compilers available at our site) interfacing to ViewLogic (see figure 2).

7 Case Study: The Design of a Stack-Based Microprocessor

To maximize the understanding of the interaction of all levels of computer design (hardware, compilers, OS), we emphasize integration of system design consideration in student designs. Students are free to choose their own processor architecture and implementation strategy, generating considerable enthusiasm for the implementation of *their* processor. In the remainder of this section, we shall discuss the design and implementation of a stack processor, as designed by one of our students ([Mau93]).

The student who did this project had already completed a compiler design class ([Bro92]), where he implemented a compiler for a stack machine. This machine specification was then used as the base design specification for the computer architecture project (see figure 3).

The original design had variable length instruction formats, ranging from 8 to 24 bits, with a unified instruction and data memory space. At the project supervisor's suggestion, for simplicity, this was altered to use separate 24 bit instruction and 16 bit data memories. This way all immediate constants could be encoded in single length instructions. The design was going to be non-pipelined, with a finite state machine (FSM) controlling the data path. The finite state machine was modelled using a microprogram-like mnemonic representation (see figure 4). Through the use of C macros, this representation was compiled to bit vectors, representing the values of the data path control signals in each state ([Gsc93]).

The design of the data path was straight-forward, using Xilinx-supplied macros, the TTL emulation library and our own, generic bit-slice ALU. The FSM was implemented using the Xilinx `memgen` tool. This allows automatic generation of ROM- and RAM-like structures.

Integration of the design was seamless. The design was simulated using ViewSim. The simulation was largely successful, but exhibited occasional unexpected behavior, like erroneous incrementing of the PC – this was tracked down to hazards in the automatically generated ROM. The student had stabilized the control signals by latching the current state, allowing hazards to propagate to all functional units in the data path. By latching the control signals of the current state instead, these hazards were masked out. After this final verification, the original compiler was adapted to reflect the changes made to the architecture at the beginning phase of the project. Thus, a fully functional microprocessor environment was available, including a compiler and a hardware prototype, implemented on one Xilinx XC4005 FPGA.

8 First Impressions

After implementing several chips using our current FPGA board, we are now considering designing a new board with enhanced functionality.

We were able to identify several points to be improved on in this new generation:

- Using several interconnected FPGAs will allow to implement larger, more complex designs:
- Logic density is low for some types of specialized, very technology-dependent blocks, such as ALUs. A possible solution to this problem is the inclusion of a dedicated ALU chip on the board (for example, a member of AMD's bit-slice ALU series), which can be integrated into FPGA based designs. Thus the logic density of data paths would no longer pose a problem. A model of this unit would be created for the design entry system (ViewLogic in our case), such that the complete design can be entered and simulated in ViewLogic.
- We have noticed a lack of random access memory. Traces cannot be gathered in 'real time', as no sufficiently large memory is available for storing the traces.

9 Future Development

Currently, the board's application domain is limited as this board is still in a prototype phase:

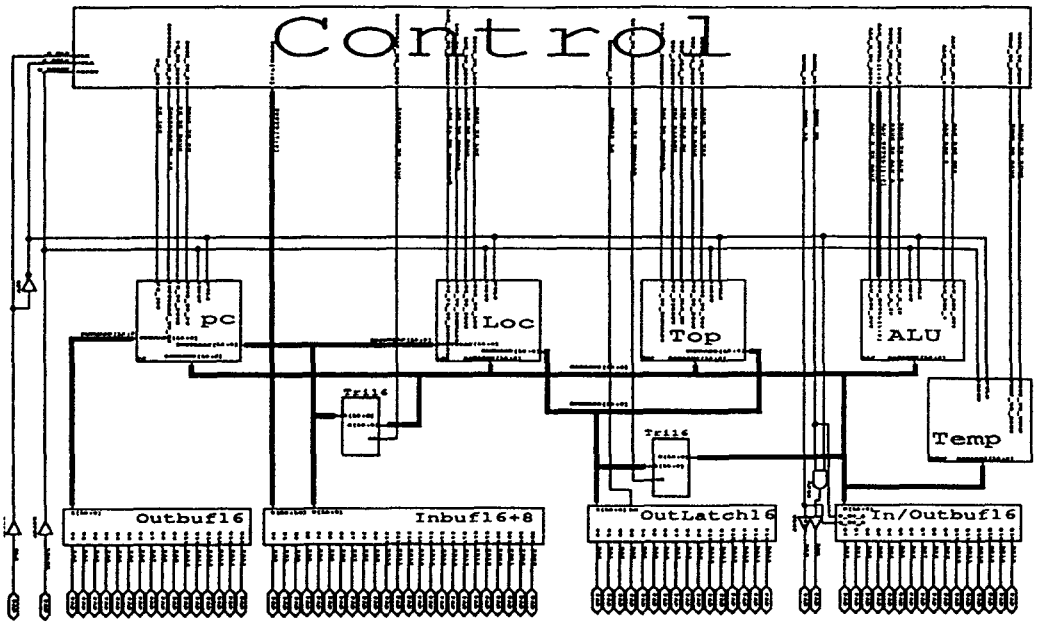

Figure 3: Block level diagram of stack machine

```
LABEL(ADD)
  COM( top_to_dmemadd | dmemadd_le | dmem_rd | dbus_to_alu_a )
  COM( top_direct_down | top_clk_en )
  COM( top_to_dmemadd | dmemadd_le | dmem_rd | dbus_to_alu_b )
  COM( top_to_dmemadd | dmemadd_le | dmem_wr | alu_e_to_dbus | alu_cntrl_add | pc_inc | fetch )
```

Figure 4: FSM code for adding the top two stack elements

- Applications may only require a moderate number of gates (max. 5000 gates for the XC4005).
- The currently used glue logic implements only part of the AT bus standard: the board is a passive device, which cannot generate interrupts. All communication with the PC and its devices has to be done through polling by the master CPU. This includes main memory access, which cannot be initiated by the board, but rather has to be routed through the CPU by using a polling strategy.
- The AT bus is too slow to allow high speed communication with the master CPU, main memory and attached devices.

10 Future Directions

Possible (and planned) future enhancements are using multiple FPGAs which would be programmed by software that partitions complex circuits to fit into several FPGAs. Local (static) memory should provide sufficient bandwidth for the FPGA cluster operating at higher speed. Also, we are planning to use a high speed bus to connect this cluster to the master CPU, main memory and the attached devices. One such emerging technology is the VL local bus.

A different approach for a high speed connection to the CPU is using the SIMM memory interface. This leads to a computing paradigm which is especially well suited for systolic array applications, but suffers from the same drawback as our current board: it only can be used as a passive device accessed through polling.

11 Related Work and Conclusion

Intel Corp. used 14 Xilinx-based Quickturn RPMs[2] to fully simulate its current top-of-the-line Pentium™ microprocessor as part of the Pentium™ pre-silicon validation process([KNZB93]). The simulated Pentium™ microprocessor achieved an emulation speed of 300 kHz and booted all major operating systems for Intel's x86 processor family.

Athanas et al. use a reconfigurable coprocessor based on FPGAs to speed up C code. Their approach is to identify C functions which can efficiently be implemented in FPGAs and have the coprocessor execute these functions. They report speedups of factors up to 26, but the type of functions that can be implemented is *very* limited [AS91]. van den Bout et al. use cascaded FPGAs to increase the gate capacity and static RAMS to increase memory capacity of their Anyboard. They have implemented a circuit partitioner to distribute circuits over several FPGAs [vdBMT+92]. Mathisen et al. use an FPGA board to provide a reconfigurable I/O interface[MU92].

With this work we have demonstrated that FPGAs al-

[2]Each Quickturn RPM contains approximately 50k gates using Xilinx FPGAs.

low hardware design to have a debug cycle comparable in speed to that of software, while still employing *real hardware* instead of software simulation for testing. This allows the design of real-life architectures which can be tested in real time and keeps the cost of exploring different design options low.

Our experience shows that it is feasible to have every computer science student design, implement and test a CPU using standard libraries targeted to FPGAs. Targeting at silicon would require more extensive testing, development of test vectors, test methods etc. – tasks which take several moths in their own right. While valid concerns, these tasks add little to the understanding of computer architecture.[3] With guaranteed-to-work, pretested FPGAs, every student can design his own CPU in little over one month, witnessing the full cycle of architecture definition, logic design and verification and to actually see the processor working.

12 Acknowledgement

Credits are due to Alexander Jaud and Ernst Huber for building and testing the board. Christian Mautner designed the stack-based microprocessor presented as case study.

References

[AS91] Peter M. Athanas and Harvey Silverman. An adaptive hardware machine architecture and compiler for dynamic processor reconfiguration. In *International Conference on Computer Design*, 1991.

[Bro92] Manfred Brockhaus. Übersetzerbau. Vorlesungsskriptum, TU Wien, 1992.

[GGE91] H. Grünbacher, M. Gschwind, and W. Eder. The design of a RISC controller based on a load/store architecture. In *Proc. of the Second EuroChip Workshop on VLSI Design Training*, Sep. 1991.

[GJ92] H. Grünbacher and A. Jaud. JAPROC - an 8 bit microcontroller and its test environment. In *Proc. of the Second International Workshop on Field-Programmable Logic and Applications*, Aug. 1992.

[Gsc93] Michael K. Gschwind. Automatic generation of finite state machines for data path control. Technical report, TU Wien, 1993.

[Hub92] E. Huber. *Eine Einsteckkarte für den IBM-PC/AT zur Programmierung von Xilinx FPGAs.* Diplomarbeit, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, Sep. 1992.

[KNZB93] Wern-Yan Koe, Harish Nayak, Nazar Zaidi, and Azam Barkatullah. Pre-silicon validation of Pentium CPU. In *Hot Chips V - Symposium Record.* TC on Microprocessors and Microcomputers of the IEEE Computer Society, August 1993.

[Mau93] Christian Mautner. Entwurf und Implementation eines Stack-basierten Mikroprozessors. Course report, TU Wien, 1993.

[MU92] Jan Anders Mathisen and Lisbet Utne. A multi-purpose I/O-board. In *Proc. of the Second International Workshop on Field-Programmable Logic and Applications*, Aug. 1992.

[vdBMT+92] Dave van den Bout, Joe Morris, Douglas Thomae, Scot Labrozzi, Scott Wingo, and Dean Hallman. AnyBoard: an FPGA-based, reconfigurable system. *IEEE Design&Test of Computers*, Sept. 1992.

[Xil92] Xilinx. *The XC4000 Databook.* Xilinx Corp., 1992.

[3] These problems are covered by VLSI design lectures.