WEB-BASED SUPPORT FOR COMPUTER ENGINEERING

HELMUT BÄHRING, JÖRG KELLER, WOLFRAM SCHIFFMANN

FernUniversität, FB Informatik, 58084 Hagen, Germany
{helmut.baehring,joerg.keller,wolfram.schiffmann}@fernuni-hagen.de

Abstract – Learning about Computer Engineering is a highly constructive process. At our department, computer engineering encompasses three courses and a laboratory session. In order to successfully learn the subject, the students should have hands-on experience with tools. While tools to program (compilers, interpreters, development environments) are widely available for free on the internet, the situation is different for computer engineering. To learn the foundations of digital design, a tool to design and simulate digital circuits is needed. To master microprocessors and microcontrollers, access to assemblers and simulators is needed. To master computer architecture, access to an architectural simulator is needed. The laboratory session requires either an extended on-campus phase or necessitates web access to the devices (processor test cards, oscilloscopes, etc) in the lab. We present an overview of our efforts in the last five years, through which we have provided students with web access to all the tools and devices above.

INTRODUCTION

FernUniversität is Germany's Distance Teaching University, serving also Austria and the German speaking parts of Switzerland. The computer science department offers study programs leading to bachelor, master and doctoral degrees. Students receive printed course texts and assignments, and send their solutions by surface mail. The solutions are then corrected, scored and returned. In order to improve the study situation, FernUniversität provides course portals where students can access the study materials online, and via which a course supervisor can communicate with the students enroled in his course. Furthermore, the opportunities of multimedia and online access are used to improve the course materials themselves.

The field of computer science shares some characteristics with other natural sciences and engineering disciplines. First, the topics taught involve many situations where dynamic processes are explained, e.g. how a sorting algorithm progresses during its execution. This can be supported by providing Java applets with visualizations and simulations. Second, the assignments often involve constructing programs or circuits. This can be supported by giving students access to the appropriate tools, and to provide them with feedback as fast as possible. Third, besides lecture courses, computer science involves a number of laboratory courses. In order to relieve students from long on-campus times (which are difficult to realize because most students work), students can be provided with access to the respective computers and laboratory devices.

The support mentioned above has made great progress in theoretical computer science and the software domain. Many visualizations for algorithms are available, and compilers and software development tools are freely available over the internet for many platforms. Access to servers over the internet via `ssh` and other protocols is standard. In contrast, computer engineering suffers from the fact that CAD tools, simulators, and device simulators are not widespread. In the following sections we will present how we support the computer engineering courses that are part of the computer science curriculum in Germany. First we briefly sketch how applets can help to understand basics of computer engineering. Second, we sketch how students can learn to construct digital circuits in their assignments with the help of CAD tools. Third, we describe how we substitute part of the on-campus phase of our hardware lab course by a virtual lab. Last, we give an outlook on our current research.

COURSE UNITS WITH APPLETS

Many introductory course texts about computer engineering deal with combinatorial or sequential logic. Unfortunately, the operation of those circuits cannot be illustrated by a single picture, because it depends on the great number of different input patterns (combinatorial logic) or also on the internal state of the circuit (sequential logic). Additionally, propagation delay times increase the complexity of the dynamic behavior of these logic circuits. If one can only use textual descriptions and a restricted number of pictures it is very difficult to explain the operation of logic circuits. It is also not useful to include series of pictures for every different state of a circuit.

In order to solve this problem we extended the course texts by links to applets that provide simulations of the logic circuits discussed in the course unit. The applets are referenced at the margins of the text by a Java logo and can be accessed either by means of a link in the corresponding pdf-file or by an extra web page that contains all the link to applets. By means of these applets the students can better grasp the logic and dynamic behavior of the introduced circuits.

Figure 1 shows an example applet that simulates a full adder constructed from two half adders. After the student has read the text he can click on the Java logo and then the corresponding applet will be opened in a web browser. On the left side, a block diagram displays the full adder circuit. The logic assignments of the connecting wires are denoted by colors. Simultaneously, the truth table with the current input pattern being highlighted is displayed on the right side. The student can choose any of the possible input patterns and after pressing the calculate button he immediately sees the full adder's response to this input pattern. Similar applets illustrate the dynamic behavior of sequential logic, e.g. the state sequence of counters or microprogrammable finite state machines.

DESIGN AND SIMULATION OF DIGITAL CIRCUITS

Exercises in computer engineering often comprise the design of logic circuits that transform binary input patterns to output patterns. Exercises of this kind pose some problems for distance learning students because of their isolation from lecturers and fellow students. Thus, they cannot discuss their solutions with others and they are unable to check the correctness of their designs in advance. In conventional paper-based exercises it often takes up to four weeks until they get back the revised and scored exercises. In order to improve the motivation and the learning situation of the students it is important to provide opportunities to test the logic circuit designs in advance. This can be accomplished by means of a CAD tool for schematic entry and simulation. Because commercial CAD tools are expensive, complex and difficult to use they cannot be used for this purpose.

The HADES (HAmburger DEsign System) [3] is a portable, open source and easy-to-use CAD program that accomplishes all the needs for conducting exercises in computer engineering. By means of HADES, students can enter the schematics of any logic circuit (combinatorial or

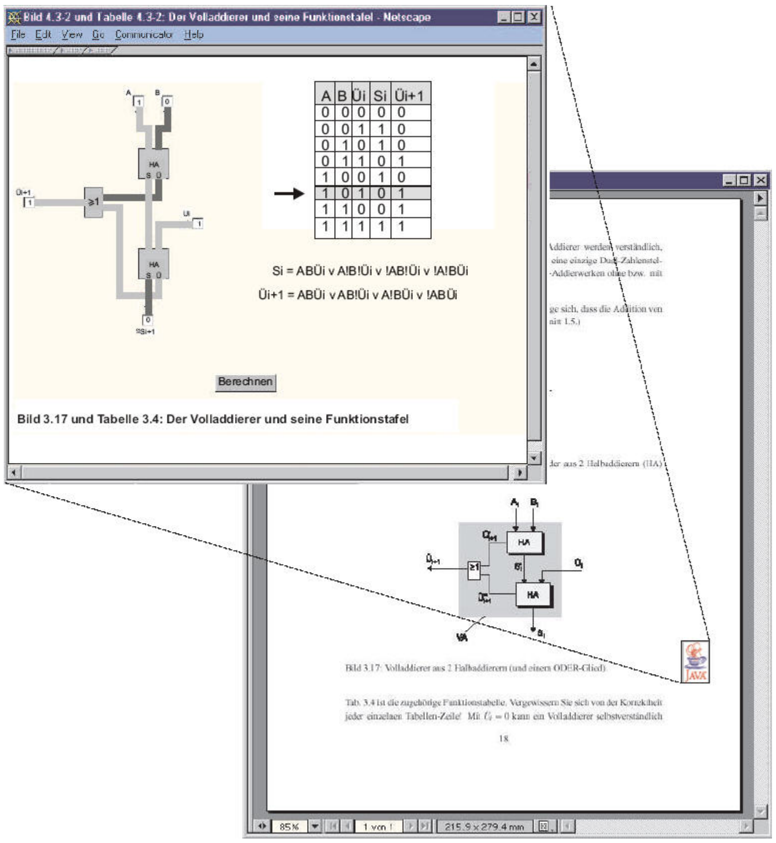


Bild 4.3-2 und Tabelle 4.3-2: Der Volladdierer und seine Funktionstafel - Netscape
File Edit View Go Communicator Help
A
B
Üi+1
Ui
HA
≥1
!Si+1
Berechnen
VA
1 von 1
215.9 x 279.4 mm
85%

A	B	Üi	Si	Üi+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Si = ABÜi v A!B!Üi v !AB!Üi v !A!BÜi

Üi+1 = ABÜi v AB!Üi v A!BÜi v !ABÜi

Bild 3.17 und Tabelle 3.4: Der Volladdierer und seine Funktionstafel

Bild 3.17: Volladdierer aus 2 Halbaddierern (und einem ODER-Glied)

Tab. 3.4 ist die zugehörige Funktionstabelle. Vergewissern Sie sich von der Korrektheit jeder einzelnen Tabellen-Zeile! Mit $Ü_i = 0$ kann ein Volladdierer selbstverständlich

18

Figure 1. Screenshot of an applet that illustrates the operation of a full adder made of two half adders.

sequential). In order to stimulate a circuit and to display the values of its outputs, different kinds of I/O devices are provided. In addition, hierarchical designs are supported to cope with complex designs by means of modularization. Also, in contrast to commercial CAD tools, HADES offers a number of other benefits:

- It can be used on any platform because it is written in Java.
- It is open source. Thus, it can be easily adapted to the specific needs of a course, e.g. the symbols of the gates can be changed to the German DIN format.
- The program size is less than 4 Megabytes. Therefore, it can be downloaded in reasonable time even via a modem connection.
- After the download, HADES can immediately be used and its usage is easy. This feature is most important for the acceptance by the students.
- While simulation in commercial CAD tools requires the start of a separate program, HADES has a built-in simulation facility. Thus, the students do not have to switch from schematic entry to simulation mode and vice versa.

Self Tests by Means of Testbenches

Testbenches are used to check the functional correctness of a design. Students can benefit from testbenches by evaluating their solutions before submission. Later on, the same testbenches can be used for checking the submitted solutions automatically. For this purpose a so-called correction server is provided. The services from this server are requested by the WebAssign platform

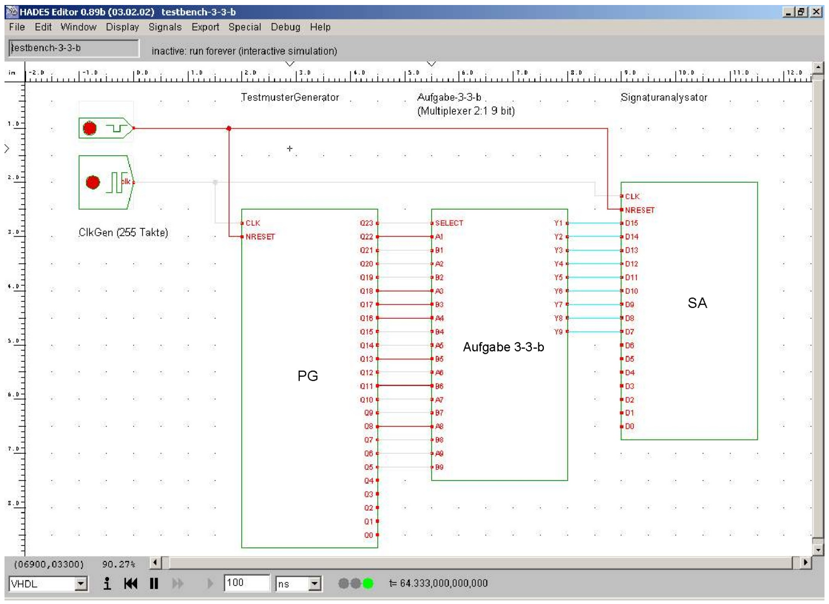


Figure 2. Screenshot of a HADES testbench with PG (left), SA (right) and the student solution (middle).

which is used by the students to submit their solutions via the web (see next subsection). By means of the correction server the WebAssign platform informs the correctors whether the student's solution implements the required functionality or not. Therefore, the human correctors can concentrate on evaluating the design in terms of other requirements e.g. if it conforms to design rules such as naming conventions, or if it contains unnecessarily complex parts.

The testbenches help the students to get a quick feedback while they try to find solutions to the exercises. This improves their motivation because they have a chance to modify their solution until they find a correct one. Our evaluation of the students' performance in the assignments of the summer semester 2003 confirms this assumption [4].

In order to enable automatic tests we extend the logic circuit by two additional components which represent the testbench. While the first component, called the pattern generator (PG), provides stimuli signals to the logic circuit, the second component traces the response of the logic circuit to those stimuli signals. To generate the stimuli a linear feedback shift register (LFSR) is used. The LFSR has a word width of 16 to 32 bits and is initialized with a bit pattern called *seed*. Due to the feedback the LFSR produces a sequence of bit patterns that are used as stimuli signals. The response signals of the device under test are fed to a signature analyser (SA) that is realized like an LFSR. Additionally, the outputs of the test circuit are combined with the feedback in order to generate the follow-up states of the flipflops. After a given number of clock cycles a specific bit pattern will be generated in the SA. This so-called *signature* depends on the properties of the PG (seed, feedback structure), the functional behavior of the device under test and the properties of the SA (seed, feedback structure). The signature summarizes the function of the test circuit by a single (hexadecimal) value. It is important to notice that the signature is invariant to details of the realization of the device under test (so-called *black box test*). Also, it is extremely unlikely that an incorrect realization will generate the correct signature. Thus, we can use the signature to test the functionality of a logic circuit designed by the students.

For each exercise a specific testbench is provided, but to avoid confusion all have identical feedback structures for PG and SA and identical seeds for the PG. Only the seed of the SA is varied. The students are provided with a seed-signature pair to test their designs at home. Another seed-signature pair is used by the correction server to check the submitted designs.

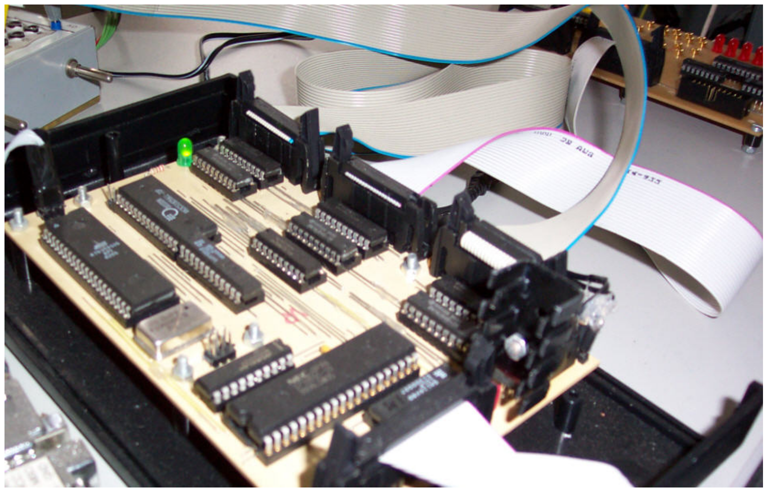


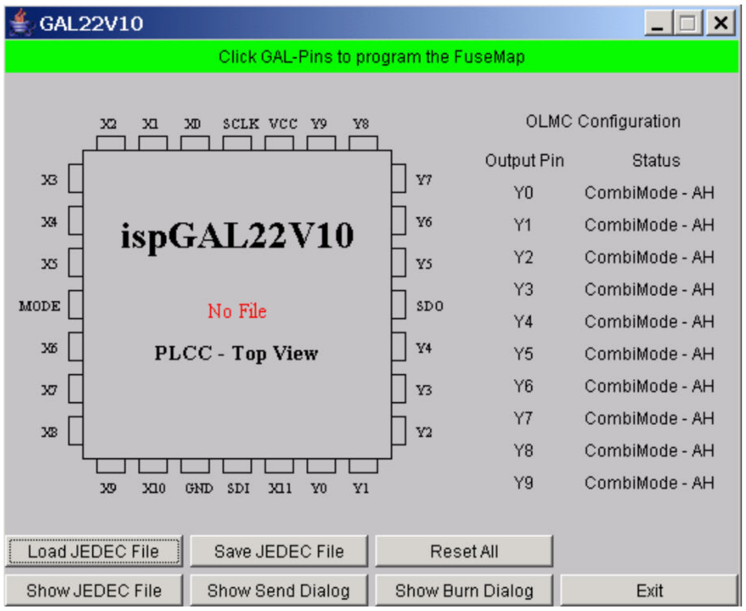
Figure 3. The Experimental Control Unit (ECU).

WebAssign for Conducting Exercises

WebAssign is a platform that supports the interaction of the parties involved in conducting internet-based exercises [1]. One of the objectives of WebAssign is to shorten the processing times by using the internet instead of conventional mail. In the past, the exercises were printed and sent to the students via yellow mail. When the students solved the assignments they returned them to FernUniversität, from where the submissions were distributed to the correctors who are spread all over Germany and the neighboring countries. The corrected exercises were returned to FernUniversität and finally forwarded to the students. This was complicated, time-consuming, error-prone and expensive (because of the mail charges).

WebAssign offers a web-based access to a database for the parties concerned: lecturers (or teaching assistants), students and correctors. The lecturers can enter assignments and sample solutions for their courses. They are also able to establish deadlines for the submission of the assignments and the release of solutions, to determine the distribution of submissions among the external correctors and to access the scores of the correctors for the students' solutions. Students can access assignments and upload their solutions. When the deadline for submitting solutions is passed they also can access the sample solutions to the assignments. In accordance to the distribution given by the lecturer, WebAssign allocates the submitted solutions to the correctors. WebAssign also returns the corrected submissions to the students via email.

In order to support the correctors we extended WebAssign by a correction server that automatically checks the correctness of the submitted solutions by means of the testbenches described in the previous subsection. For each assignment the students must download a specific testbench and insert their solution. Figure 2 shows a screenshot of a testbench together with the solution module that was inserted by a student. As already described, the students can test their designs at home by using the testbench together with a given seed-signature pair. After they have entered the seed value in the SA, a simulation run of the complete testbench is performed. If the resulting signature in the SA coincides with the given one students know that their design solves the assignment. In the same way the submitted solution is automatically checked by means of the correction server. The result of this test is stored in the database of WebAssign and forwarded to the corrector together with the student's solution.



OLMC Configuration

Output Pin	Status
Y0	CombiMode - AH
Y1	CombiMode - AH
Y2	CombiMode - AH
Y3	CombiMode - AH
Y4	CombiMode - AH
Y5	CombiMode - AH
Y6	CombiMode - AH
Y7	CombiMode - AH
Y8	CombiMode - AH
Y9	CombiMode - AH

Figure 4. Starting page of the GAL Editor.

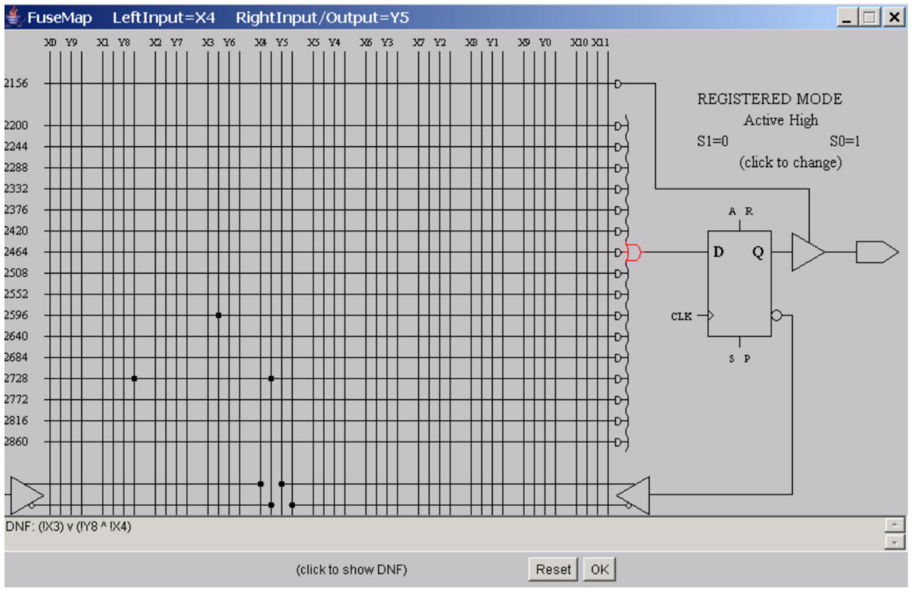
Connecting HADES with a GAL device

In order to connect and to control external hardware from the HADES graphical user interface, we developed a cheap and easy-to-use measurement device, the *experimental control unit (ECU)*, see Fig. 3. It can be connected to a parallel port of any PC and it is based on an ATMEL microcontroller. By means of this device one can easily generate test patterns and sample the response of external hardware components. We also developed a device driver for integration in HADES. In this way one can combine hardware and CAD designs (similar to hardware/software codesign). The ECU is mainly used to connect an in-circuit programmable GAL (Gate Array Logic) device 22V10 to the HADES system. This device consists of a circuit with up to 22 inputs and 10 outputs that can be used either in combinatorial or registered mode. The students can program this GAL chip by means of a Java application that allows a graphical programming of a matrix with approximately 6000 fuses. As shown in Fig. 4 the students select the output pin that should be programmed and the GAL editor displays the corresponding part of the fusemap that now can be easily programmed by clicking the intersection between vertical input (or feedback) wires and an AND-term of the output function (see Fig. 5). After programming of all the output lines the final fusemap can be uploaded via the Internet to the remote lab in Hagen where it will be used to configure the GAL chip which is connected to the HADES system by means of the ECU. The students are able to control this HADES system by means of a VNC server (Virtual Network Computer) that can be accessed by any Java-enabled webbrowser.

REQUIREMENTS FOR VIRTUAL AND REMOTE MICROPROCESSOR LABS

When students want to get familiar with microprocessor programming and deployment, they must have access to a software development environment and a target system. Our computer engineering lab course[1] is divided into four phases. First of all, the students work through a course text that describes the basics for the experiments. For instance, the processors to be programmed are described, and the pitfalls and fallacies of assembler programming are explained.

[1] In the German curriculum, a basic computer engineering education is part of the computer science programme.


FuseMap LeftInput=X4 RightInput/Output=Y5

REGISTERED MODE
Active High
S1=0 S0=1
(click to change)

DNF: (!X3) v (!Y8 ^ !X4)

Figure 5. Configuring a registered GAL output.

In the second phase they use development tools to practice machine language programming. The development system consists mainly of an editor and an assembler for compiling programs written in assembler language into the microprocessor's object code. These tools — cross-assemblers in particular — are widely and freely available, and are either installed locally at the students' computer or accessed remotely. In the third phase the students test their program on a target system and watch it running there. For this purpose, we had to develop an appropriate target system that operates and looks like the real microprocessor system which is used in our laboratory on campus.

The target system should serve as a remote control for a real microprocessor system in Hagen, so that the execution of students' programs can be observed in real-time. In the simulator mode the students can use the target system as a *virtual* laboratory for testing the object code at home. Although they can test the functionality of their programs in this phase by simulation, they are not able to either check the real-time behavior, nor can they control external hardware that is connected to a real microprocessor system. In order to provide these facilities the students should be able to connect via the Internet to a real target microprocessor system located at the lab on the Hagen campus, i.e. they use a *remote* laboratory. By switching into the remote control mode the students will be able to remotely control an experimental setup in the lab, consisting of the processor itself and some additional hardware. By means of additional measurement devices that can also be controlled remotely the students can conduct experiments with external hardware (e.g. a traffic light panel or a radio frequency controlled clock). In addition to the software tools, web cams provide live pictures of how the setup in the lab behaves, e.g. which lights are on or off.

Students thus develop their programs locally and test them on a simulator. If they do not have the rights to install software on the computer they are using (because of missing software licences), they can also use a development system on a server on campus via the Internet (typically using VNC). When a student is confident that his program works, then he accesses the remote lab, uploads his program to the target system, and conducts his experiment on real hardware. If an error occurs or the real-time behavior differs from what is expected or required, he corrects the program until he is satisfied. In the end, he submits his results and his program via the WebAssign system to the correctors for grading. In this manner, the virtual lab mode saves

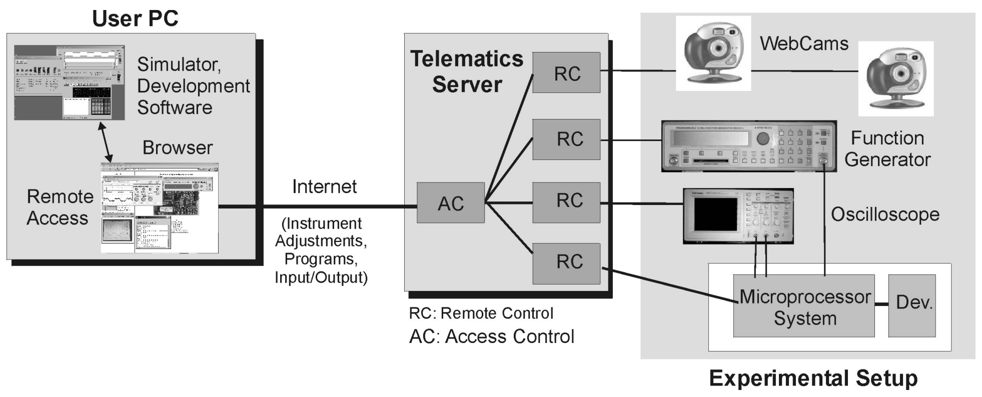


Figure 6. Structure of the microprocessor lab.

valuable online time for many students, as they can develop at home in an offline mode. This also distinguishes our approach from online labs.

Despite the two modes of operation, students should not be required to learn two sets of user interfaces, as this would incur a lot of additional learning time. Hence, the virtual lab and the remote lab should share a common user interface. For example, a student should be aware whether he is running code on a simulator or on real hardware, but the controls for starting the program run should be the same.

The remote laboratory is realized by a client-server approach. Figure 6 outlines the scenario of our microprocessor lab. Students who want to use the remote lab have to download a number of client programs that serve as remote controls for the microprocessor system and the measurement equipment, such as an oscilloscope or a web cam. These clients connect to server processes that run on a server computer (named telematics server) in the laboratory at Hagen. Some of those clients also have simulator functionality (e.g. the microprocessor client), so students can seamlessly switch between the virtual lab and the remote lab. The server computer in turn is connected to the hardware devices in the lab: microprocessor card, oscilloscope, web cam, and so on. The microprocessor card itself is connected to some additional hardware such as a traffic light kit or the RC clock (see next subsections). The server also provides a central installation of the development software and the simulator for students who are not allowed to install software on their local computer. Those students may access the server either via a remote shell or session, or web-based via VNC.

To accomplish more complex experiments with attached devices, such as a radio controlled clock (see next subsections), the simulator establishes a TCP/IP-connection via the internet to the server at our laboratory, thus allowing students to switch seamlessly to the remote lab. After authentication, the user gets access to the remote control software of different components of the experimental setup. These components include one or more oscilloscopes, function generators and mainly the microprocessor system hardware. Attached to this system will be different devices, e.g. the above mentioned radio controlled clock or a model of a traffic light.

The user is able to upload his programs to the remote microprocessor system via the user interface and to start them there. He can adjust the already mentioned instruments remotely, perform different measurements, and gets back the results. To give him the possibility of observing the experiments, and thus the feeling of being in the laboratory room, all instruments and devices — but also the whole laboratory — are accessible by controllable web cams. (E.g., look at Fig. 11 where the control applet of an oscilloscope is shown in a window on the left and the web cam image of the oscilloscope in another window on the right.)

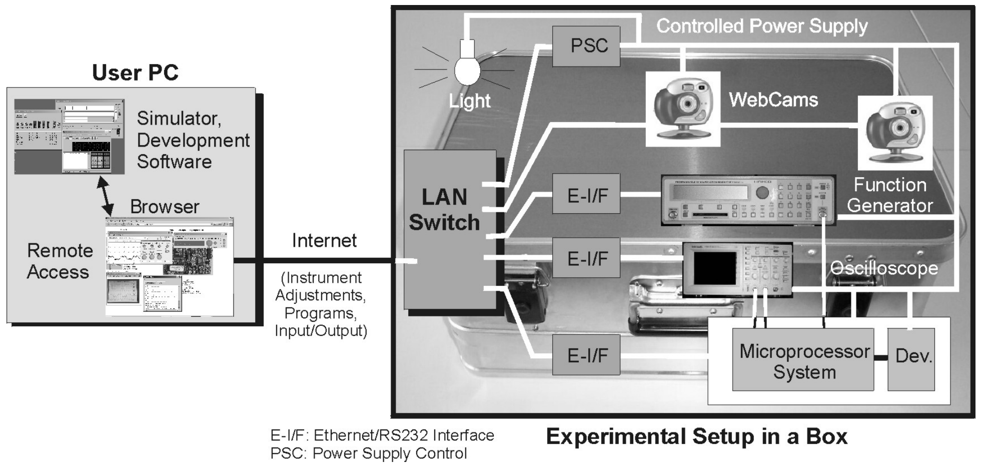

User PC
Simulator, Development Software
Browser
Remote Access
Internet
(Instrument Adjustments, Programs, Input/Output)
Controlled Power Supply
PSC
Light
WebCams
LAN Switch
E-I/F
E-I/F
E-I/F
Function Generator
Oscilloscope
Microprocessor System
Dev.
E-I/F: Ethernet/RS232 Interface
PSC: Power Supply Control
Experimental Setup in a Box

Figure 7. Microprocessor lab in a box.

The client programs are preferably programmed in Java. The software is programmed in two parts: one part that is independent of the particular device for which the client is the frontend, and one part which realizes the particular device's frontend. The former part can be re-used for all devices, the latter part is programmed in a way that realization of an additional device can be done with the least possible effort. Using Java also had the advantage of being more independent from the type of computer the students have locally, than with any other realization.

Of course, the implementation of the lab is not restricted to a particular microprocessor card. There are cards with a universal microprocessor, a microcontroller and a digital signal processor, respectively, that can be accessed in the same manner. We will demonstrate this by two examples in the next subsections.

A Lab in a Box

In a number of experiments no server machine is needed to control the offered measurement equipment and microprocessor systems with their attached components. For those experiments we develop a remote lab with a smaller form factor which can be implemented in a box of moderate size. Its structure is shown in Fig. 7. In this lab all instruments, the microprocessor system, the web cams and even the power supply and the light are connected to the internet via a LAN/Ethernet switch. The power supply socket and the web cams have their own built-in Ethernet interfaces. The other devices are equipped by small Ethernet/RS232 interfaces (called XPORTS, denoted as E-I/F in Fig. 7) which are connected to their serial interfaces. These XPORTS offer an integrated web server and 384 kbyte of flash memory, enough to implement comfortable control programs for the attached units. The total lab has only two interfaces to the outside – one for the power supply and one for the internet. This kind of implementation allows the cost– and space-saving realization of remote labs: The small boxes can be stacked and placed in small laboratory rooms, allowing the simultaneous use by many students. Their modular realization eases maintenance, repair and replacement. The integrated control of the power supply saves energy and increases the lifetime of all components because they are switched off when not used.

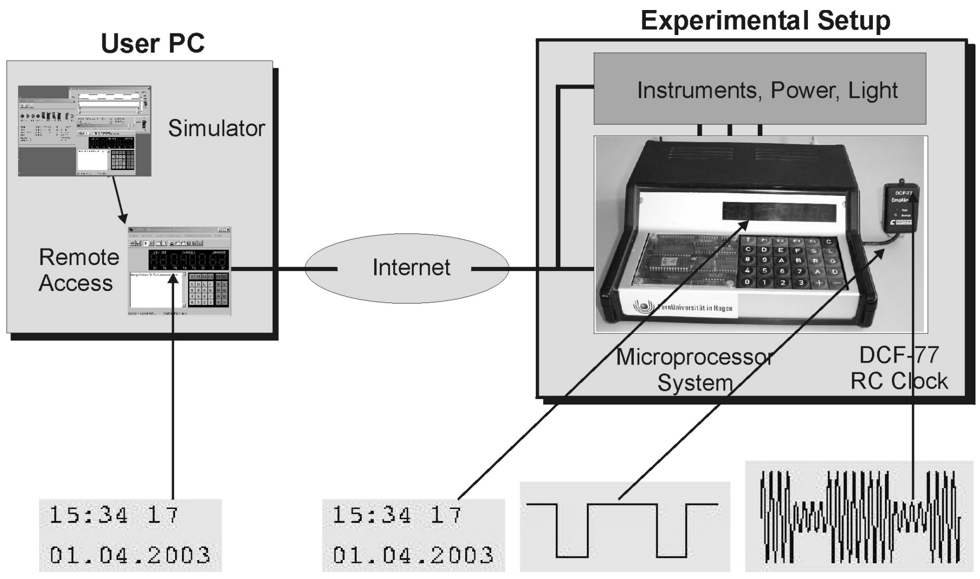


Figure 8. Lab for a universal microprocessor.

A Lab for Universal Microprocessors

In this subsection we demonstrate how the virtual and remote lab can be used by the students to study fundamentals of microprocessor technology. In our basic practical course we use a universal microprocessor to get the students acquainted with the basic processor and instruction set architectures and the common interface and control units. The structure of this lab is shown in Fig. 8. The microprocessor system consists of a custom-designed microcomputer, integrating an 8-bit processor, a keyboard and a seven-segment display. As peripherals it offers two parallel ports, two serial interfaces and three 16-bit timers. A small operating system (monitor program) allows to store programs or data and to control the display, the keyboard and the peripherals. The RS232 Interface is used to communicate with a PC or an XPORT (see above).

In basic experiments the students have to write a number of rather small programs concerning all the above mentioned microcomputer components. For ”higher” experiments different devices can be attached to the microcomputer. In Fig. 8 you can see a radio frequency controlled clock (DCF-77) which shows the exact mean time of central Europe. The objective in this experiment consists of writing a program that should evaluate the signals of this radio controlled clock. The students can watch these signals remotely with the help of an oscilloscope. There they see the analog signal received by the antenna of the clock module and the generated digital signal (see Fig. 8). Their job is to write a program which analyzes this digital signal and derives the demanded time and date.

When the students have worked through the course text, which describes the basics for the experiment, they can conduct the experiment first with the help of the simulator. Figure 9 shows that the graphical interface of this simulator resembles very much the appearance of the real microcomputer, so the student can easily operate both. Besides, this graphical interface can also be used as a remote control during the remote laboratory experiment. When – in this mode – the user operates the clear key C of the graphical interface a special signal is sent to the real microcomputer which is thereby reset into its initial state. By means of a pull down menu the user can switch between the two modes: simulator – remote control. For the remote mode the IP of the remote lab server must be entered. Access control (see Section on Gridlabs) avoids conflicts due to simultaneous access by several students.

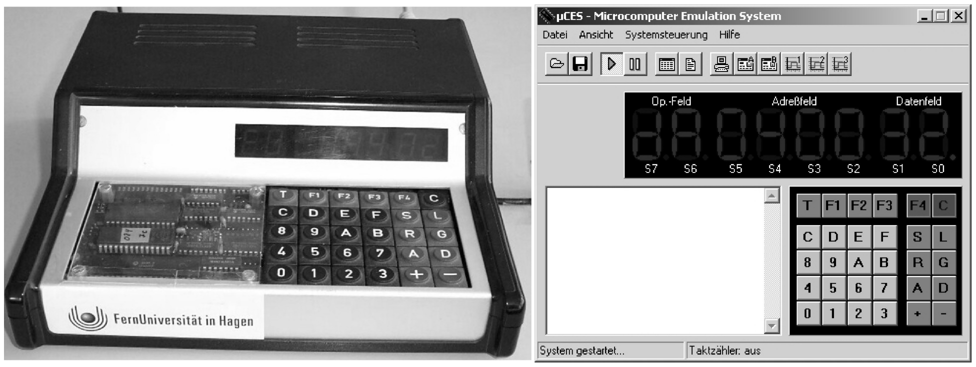


Figure 9. The real microcomputer (left) and the graphical interface of its simulator/remote control (right).

A Lab for Digital Signal Processors

In this subsection we describe the second remote lab which we have already realized. Its main subject is the architecture of digital signal processors (DSP). As our clients study computer science and not electrical and electronic engineering, our DSP lab does not aim at the theory and practice of digital signal processing itself. Consequently our students are only confronted with the function of digital filtering by FIR filters (finite impulse response) – surely the simplest function of digital signal processing. Figure 10 shows the assembly of our remote DSP lab.

To motivate the students and to convince them of the elegance of DSP solutions they first have to study in simple experiments the behavior of analog filters (i.e. circuits of resistors and capacitors) in the time and frequency domain. By the help of a remotely controllable switching unit they can select between the outputs of a low-pass filter, a high-pass filter and a band-pass filter with the same characteristic frequency and watch it on the oscilloscope. The input signal of the filters is generated by a remotely controlled function generator and can be a square wave signal, a sinus signal or a triangular signal with selectable frequencies.

In the next step the students have to realize a microprocessor solution for the above filter problem. Therefore they can select — by the switching unit — a system consisting mainly of a digital signal processor (DSP) and a codec for the conversion of analog to digital signals and vice versa. They have to write programs using a professional software development environment via VNC and to upload them into the DSP. As an example, Fig. 11 shows the graphical user interface of the DSP lab. On the left side you can see the remote-control interfaces for the instruments used: the function generator at the top and the oscilloscope below. At the top of the right side you can see the window to select and upload programs from the client PC to the DSP. At the bottom is shown a window displaying the streaming video of the oscilloscope screen captured by a web cam. For experts only: both the remote-control interface and the web cam image of the oscilloscope show a 1500-Hz square-wave signal as input (upper channel) and the typical low-pass filtered signal as output (lower channel).

A Computer Architecture Lab

The purpose of the previous laboratories was the use of a certain device, e.g. learn how to program a microprocessor in assembler. In contrast, the computer architecture lab deals with questions about the architecture of the microprocessor under study, e.g. to determine the optimal cache size for a given set of benchmark programs. Obviously, these questions can be solved by simulations, as is usual in computer architecture research. However, processor simulators tend

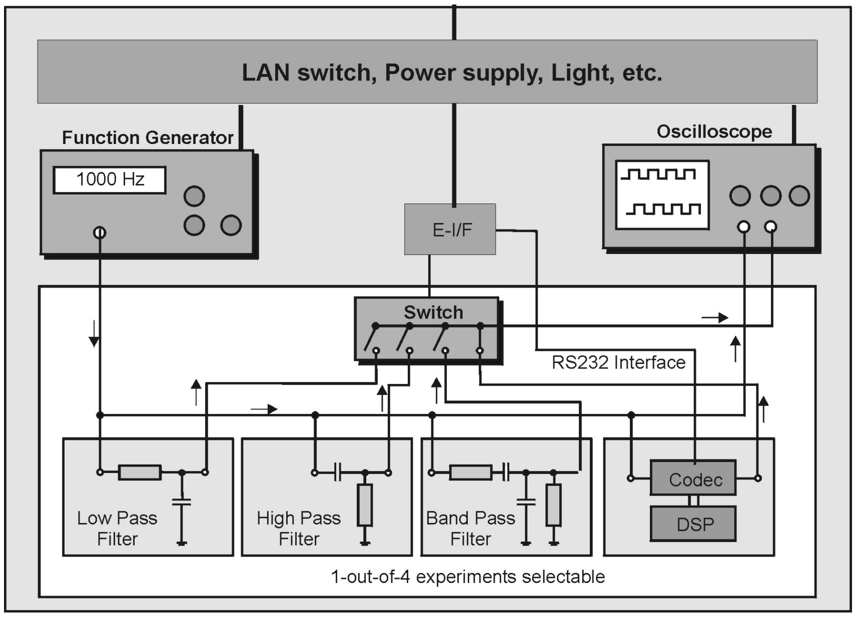


Figure 10. Structure of the DSP Lab.

to be either focussed on a particular instruction set and/or microarchitecture, see e.g. WinDLX [5], or are more general but complex to install and maintain, like SimpleScalar [2]. Moreover, a simulator like SimpleScalar demands for a powerful machine to be executed on, in order to deliver results in a reasonable time.

Our *Computer Architecture Lab (CA Lab)* resolves this conflict by providing an installation of SimpleScalar on a powerful server, and by giving access to this installation via a web page. Via the web page, the command line parameters of SimpleScalar can be set, e.g. the size of the instruction cache. In this manner, the students get access to a state-of-the-art tool in computer architecture research, but are freed from the burden of managing its details. If several users access the web page concurrently, a separate instance of SimpleScalar is invoked for each of them. In order not to overload the server, and thus provide a reasonable response time, the number of users can be restricted.

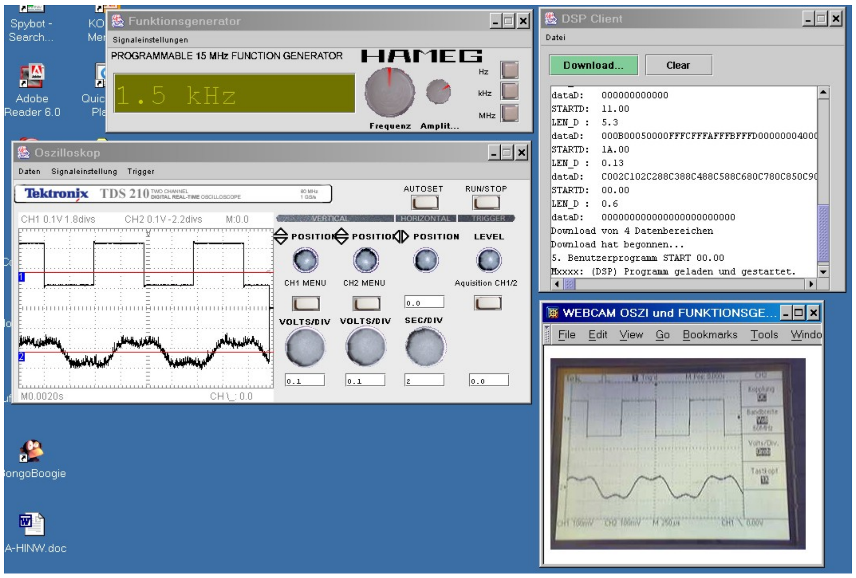
The web page can be further customized in order not to set all parameters but only those that are relevant for the task at hand. In this manner, a complete suite of pages can be constructed quickly for all assignments in a course on computer architecture[2]. It is also easy to extend the system to host several installations of SimpleScalar, each targeting a different one of the supported instruction sets. In this way, also comparative studies are possible.

GRIDLAB: WORLDWIDE SHARING OF REMOTE LABS

We have already discussed that laboratory experience represents a valuable component of distance learning. The students can conduct experiments while they study the written course materials and this helps to improve the comprehension of the subject matter.

We admit that it is costly to develop remote labs. Also, each experiment is still only used to full capacity during specific periods of time, although the utilization of a virtual lab is typically higher than that of a local lab. Hence it may be useful to share remote labs among several organizations, that can be distributed worldwide. The idea that *virtual* organizations share re-

[2]To be more exact: processor architecture.



```
dataD:   000000000000
STARTD:  11.00
LEN_D :  5.3
dataD:   000B00050000FFFCFFFAFFFBFFFD00000004000
STARTD:  1A.00
LEN_D :  0.13
dataD:   C002C102C288C388C488C588C680C780C850C90
STARTD:  00.00
LEN_D :  0.6
dataD:   000000000000000000000000
Download von 4 Datenbereichen
Download hat begonnen...
5. Benutzerprogramm START 00.00
Mxxxx: (DSP) Programm geladen und gestartet.
```

Figure 11. The graphical user Interface of the DSP Lab.

sources is essential to the emerging field of grid computing. In our current research we adopt this idea to develop a web portal that provides a marketplace for remote labs of different disciplines. This portal, called *GridLab*, will enable the users to search for appropriate experiments, book the remote laboratory for a specific period of time, conduct the experiment together with other students and get support from a tutor on demand. Furthermore, an economic model for laboratories' providers and users will be developed. The providers should get free access to other remote labs to the same extent they provide labs by themselves. Alternatively, users can also purchase credits for remote experimentation time by paying money.

The GridLab web portal, illustrated in Fig. 12, will be based on an existing CSCL portal (Computer Supported Collaborative Learning) that mimics the concept of a real university building. Starting from an entrance hall there are virtual learning rooms that can be accessed by means of virtual keys, which are only valid during the booked period of time. In contrast to the existing CSCL portal, GridLab will provide *augmented* virtual learning rooms that contain remotely controllable devices and that can be observed by web cams. When a student leaves a learning room the current state of the experimental setup is stored. In this way one can continue experimentation later on.

We will also provide support for setting up new experiments by reusing the concept of a lab in a box (see appropriate section). Physically the experiments will be located in boxes of four different sizes. Inside each box there is a light source that will be switched on together with the other measurement devices when a user enters the learning room, to enable experimentation at night time. Internet web cams will be mounted on the sides of the box to observe the experimental setup and to verify the data displayed on the remote panels of the measurement devices. To enable users to easily integrate all the components into the GridLab portal, an API will be provided.

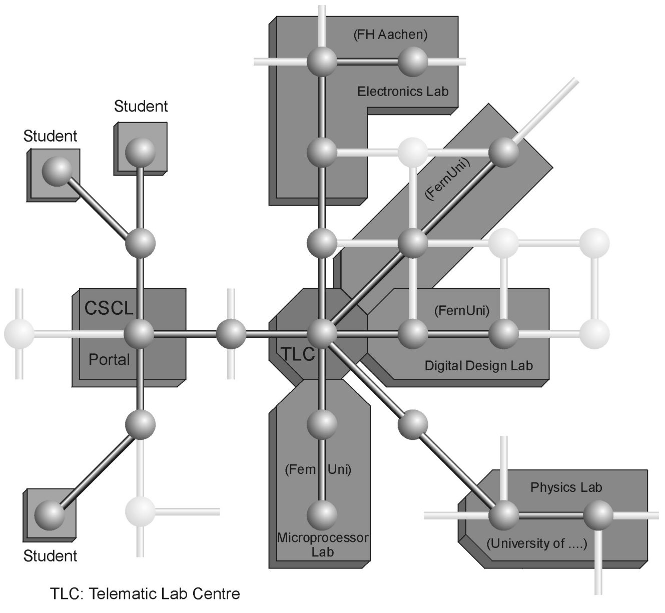


Figure 12. Example structure of GridLab.

REFERENCES

[1] BRUNSMANN J., HOMRIGHAUSEN A., SIX H.-W., VOSS J. (1999) *Assignments in a Virtual University – The WebAssign-System,* Proc. 19th World Conference on Open Learning and Distance Education, Vienna/Austria, June 1999

[2] BURGER, D.C., AUSTIN, T.M. (1997) *The SimpleScalar Tool Set, Version 2.0,* Computer Architecture News, 25 (3), pp. 13-25, June, 1997

[3] HENDRICH N. (1998) *HADES: The Hamburg Design System,* ASA'98, European Academic Software Award/Alt-C Conference, Oxford, September 1998

[4] HÖNIG, U., KELLER, J., SCHIFFMANN, W. (2003) *Web-Based Exercises in Computer Engineering,* Proc. Int. Conference on Networked e-learning for European Universities, Granada, Nov. 2003, `http://pv.fernuni-hagen.de/docs/cevu-final.pdf`

[5] GRUENBACHER, H., KHOSRAVIPOUR, M. (1996) *WinDLX and MIPSim Pipeline Simulators for Teaching Computer Architecture,* Proc. IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS'96), pp. 412–419, 1996