A Case Study in Efficient Microcontroller Education

Bettina Weiss, Günther Gridling, Markus Proske
Vienna University of Technology,
Embedded Computing Systems Group E182-2,
Treitlstr. 3/2, 1040 Vienna, Austria
Email: {*bw,gg,proske*}*@ecs.tuwien.ac.at*

***Abstract*— Undergraduate education typically is characterized by a large number of students. Therefore, courses must be conducted efficiently and should not only focus on conveying the course material, but must also be oriented towards a maximum transfer of knowledge with a minimum amount of invested time on the instructor's part. At the same time, courses should be flexible to accommodate different student needs.**

In this paper[1], we identify the needs of a practical course in microcontroller programming with respect to course structure and grading, present our solutions, and discuss our experiences.

Index Terms—**embedded systems education, automatic grading, distance learning, pedagogy**

I. Introduction

Embedded systems education is a vital part of the computer engineering curriculum and has gained increasing importance in the last decade [1], [2]. However, it also suffers from problems that are specific to any hands-on course using hardware: To teach the practical skills involved, students must be assigned an appropriate workload, but lab resources are limited. Add to that the increasing demand for embedded systems engineers, which entails an increase in student numbers, and embedded systems education soon reaches its limits with respect to both lab resources and available personnel. It thus makes sense to invest a significant amount of time into the setup of a course, if the course can then be managed with less strain on the university's resources.

The recently introduced bachelor study "Computer Engineering" at the Vienna University of Technology contains several courses which focus on programming embedded systems, among them the introductory course *Microcontroller* [3]. The Microcontroller course gives second-year computer engineering students their first experiences with microcontrollers and hardware-near programming. We expect students to be proficient in C programming and to have basic computer architecture and electrical engineering knowledge. The course is intentionally kept low-level, much like [4], and teaches students both the basics of microcontroller programming (without debugging tools) in Assembler and C and how to control external hardware. For most students, it is their first contact with microcontrollers, with hardware, and with Assembler programming.

[1]This work is part of the SCDL "Seamless Campus: Distance Labs" project, which received support from the Austrian "FIT-IT Embedded Systems" initiative, funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and managed by the Austrian Research Promotion Agency (FFG) under grant 808210. See http://www.ecs.tuwien.ac.at/Projects/SCDL/ for further information.

When setting up this course, we faced the challenge of many (about 150) students in an 8 workstation lab. We also wanted to make the course more accessible to employed and handicapped students, potentially scalable to a large number of students, and suitable for distance learning, which further added to the challenge. This made it very demanding to balance the needs of the students, the demands of the curriculum, and the constraints of available resources. We are teaching the microcontroller course since 2003 and in each year experimented with different mechanisms to attain our goals. This year's course had good student feedback and was conducted very efficiently from our point of view. So we can conclude that it is possible to find a satisfactory solution to this optimization problem, and we believe that our experiences, both positive and negative, gained from setting up this course can be of value to other instructors.

In the following sections, we will first identify the requirements on practical courses in Section II. In Section III we will present the course structure of the Microcontroller course and explain why we structured the course in this way. Section IV is dedicated to our grading scheme, which is essential to achieving our goals. In Section V, we give a discussion of our current course, its advantages and disadvantages, and student feedback. Section VI concludes the paper.

II. Course Goals

When reviewing our expectations for the course, we identified three different and possibly conflicting areas of concern:

Financial Impact:
: The course should be conducted efficiently, with a minimum amount of time and resource efforts. It should be scalable to a large number of students and should be suitable for distance learning. Since the time of a professor is most expensive, whereas tutors are comparatively cheap, the bulk of student supervision should fall to tutors.

Educational Impact:
: The course should be successful, that is, it should have high student retention, a high passing rate, and convey both microcontroller programming skills and understanding of the issues involved.

Motivational Impact:
: The course should motivate students and should result in high student satisfaction. Students should get interested enough to start their own private projects.

Naturally, these goals require different techniques, which in turn may have a positive or negative influence on the other objectives. For example, in our first try, we allowed students free access to the labs, gave them an exercise set that was completely voluntary, told them that we were available for questions by email or newsgroup at any time, and conducted one programming exam at the end of the term. So basically, we told them to exercise as much as they thought they needed and then come to the exam. This scored well on the financial axis, moderately well on the motivational axis, and pretty low on the educational axis, since most students put off doing the exercises until it was too late and then dropped the course. Even those that passed lacked some basic knowledge. This taught us that for each of our objectives, we must identify the techniques most beneficial to it, but at the same time must also account for their effects on the other areas. As a result, we went through the different pedagogical measures available to us and estimated their impact, see Table I. In the table, a $+$ represents a positive impact, $-$ a negative one, and values in brackets are setup costs that only occur once. The total column states the overall impact.

Due to limited space, we cannot discuss our reasoning behind the values, but bear in mind that the table represents our personal assessment and values, so other instructors may come to slightly different evaluations.

TABLE I
DIFFERENT TEACHING TECHNIQUES AND THEIR IMPACT IN DIFFERENT AREAS

	Fin.	Edu.	Mot.	Total
autogradable exams	$+++$	$-$	$-$	**1**
essay exams	$---$	$+$	$++$	0
just 1 exam	$++$	$-$	$--$	-1
several exams	$-$	$+$	$++$	**2**
drop result	$-$		$+++$	**2**
flexible grading	$-$	$+$	$+++$	**3**
voluntary homework	$+++$	$---$		0
ungraded homework	$++$	$++$	$---$	1
graded homework	$---$	$+++$	$+$	**1**
flexible workload	$-$		$++$	**1**
extra work counts	$-$	$+$	$++$	**2**
open labs	$+++$	$-$	$++$	**4**
closed labs	$--$	$+++$	$-$	0
supervision	$---$	$++$		**-1**
no supervision	$+++$	$---$	$--$	-2
group work	$++$	$--$		0
lectures	$--$	$++$	$++$	2
electronic lectures	$[---]$	$++$	$+$	**3**
no lectures	$+++$	$--$	$-$	0
script	$[--]$	$+$	$+$	**2**
suppl. material	$[--]$	$+$	$++$	**3**

With this table in mind, we then did a complete redesign of our course, employing the mechanisms indicated by bold numbers in the total column of the table, and had far better success with it while still keeping the workload of the professor acceptably low. We can thus conclude that it is possible to get good educational and motivational results with only moderate concessions on the financial part.

III. MICROCONTROLLER COURSE

This year's course got very good student ratings. The structure we describe here already incorporates our experiences gained this year and will be first employed in the summer term of 2006. Since the changes intended for 2006 are minor and to a large extent supported by student and tutor feedback, we expect the course to have a similar (if not better) impact on students as this year's course.

A. Educational Material

Since students should be able to work at home, we needed to get the lab to the students. Although it would have been more cost effective to use simulators like in [5], simulators cannot capture all the problems that arise when using real hardware [6]. Furthermore, working with real hardware instills into students the confidence that they know how to handle hardware, and often encourages them to start their own projects. Therefore, we decided to use real hardware in the course, which had to be cheap to be suitable for "mass" production. Unfortunately, we did not find any suitable existing hardware and hence proceeded to develop our own lab hardware.

One important lesson we learned about student hardware is that it is vital to let students connect the hardware themselves. In our first course, we used a commercial HCS12 evaluation board with fixed connections, and we ended up with a lot of students who simply did not understand how a LED and a microcontroller pin go together. They had just learned that "writing 0x0F to port X turns on LED0-3". So our own hardware has no fixed connections, students must do all the wiring.

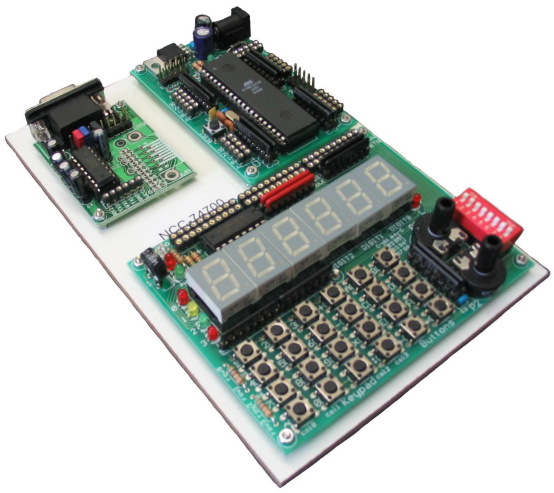

Fig. 1. Basic microcontroller hardware

We decided to use an Atmel microcontroller because they are common and fairly inexpensive, and because their RISC architecture is easy to understand and to program. In order

to be more flexible, we designed several separate boards: The *microcontroller board* only contains the microcontroller, an ATmega16, and its supportive logic (power jack and regulator, oscillator, reset button, programming connector) and single connectors to its I/O pins instead of a fixed connector like for example in [7]. As a side benefit, the microcontroller can be exchanged freely, and we are in fact working on several additional controller boards (HCS12, ARM, MSP430) to allow good students to broaden their knowledge by switching to a different controller during the course. The *simple I/O board* contains 8 LEDs, 8 buttons, 8 switches, a 6-digit numeric display, a 4×4 matrix keypad, 2 potentiometers, and a light sensor. Most of the exercises can be done with the controller board and the simple I/O board. For communication to the PC, we also offer an *RS-232 board* which contains a serial interface. All three boards are mounted on a wooden plate, see Figure 1, and can be connected via normal wires. For more challenging projects, we offer additional boards like a motor board with dc and stepper motors, see Figure 2, an interface board with serial, parallel, and CAN interfaces, or a chip-card reader.

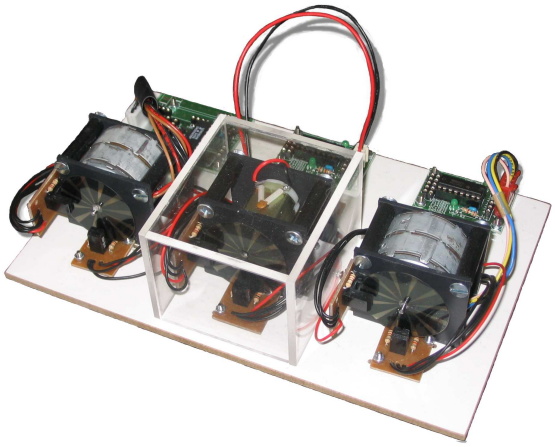

Fig. 2. Motor board with dc and stepper motors

The lab kit allows our students to work in their own time according to their own schedules. They only have to synchronize with us for homework submissions and the exams. Student feedback for the kit is very positive, and we feel that the lab kit especially helps slower students to catch up and acquire the necessary programming skills. Students could also keep the lab kits if they desired, and this offer was taken by over 70% of the students, showing us that (a) many students got interested into the topic during the course, and (b) the lab kit is useful to them beyond the course.

This year is the first in which we offered take-home lab kits for a deposit of about 70 Euro. The kit consists of the microcontroller, simple I/O and RS-232 boards, wires, cables, a power supply, and a Knoppix CD with our lab environment, see Figure 3.

Fig. 3. Contents of the lab kit

Apart from the hardware, we also offer some instructional material. Although most of the course is lab work, there are some accompanying lectures that give students an introduction into the microcontroller and the hardware. To allow distance learning and to remove the lecture load from the instructor, lectures are not held in class, but will in the future be provided electronically as slides and/or video with audio track. A lecture script complements the electronic lectures. Finally, manuals for the hardware are available, and we provide sample programs to get students started.

B. Course Structure

Basically, our course employs a mixture of exams, light compulsory workload and flexible voluntary workload that allows different work styles while still yielding comparable grades.

The course is divided into three parts. The first part is intended as an introduction into the programming environment, the microcontroller architecture, the hardware, and assembly language. The second part is dedicated to learning how to program the basic features of the microcontroller like timers and the analog module, how to handle interrupts, and how to program typical hardware like matrix keypads and numeric displays. The third part is concerned with interfaces, motor control, and application programming. Both second and third part are in C. To ensure that the students learn the most important concepts, they have to do a couple (3-4) of compulsory exercises in each part. These exercises cover the microcontroller's features in simple tasks designed to convey basic knowledge about how the feature works and how it is programmed. We decided to focus on simple tasks here because in our experience complex tasks cause students to concentrate on getting the program to work instead of on getting it right. To deepen their knowledge, students may select additional voluntary exercises from a significantly larger (20-30) exercise pool. Some of these exercises cover special details of the microcontroller or the hardware in depth, others are

applications. Depending on the quality of the work and the complexity of the exercise, students get a certain amount of "bonus" points per completed exercise, which are added to their course result.

Since there cannot be any guarantee that students do their homework all alone, each part is concluded by a programming exam (consisting of two exam tasks) and a theory exam. Attendance at all exams is not compulsory, but students must fulfill some minimum requirements concerning their exam results for a passing grade. If students come to all three exams of a type, their worst result on these exams will be dropped.

C. Lab Organization

Although closed labs seem to be more effective [8], they are hard on resources and not flexible w.r.t. working hours. Even though we concur with [8] that having a closed lab in the initial phases of the course would probably be a good thing, we cannot do this due to resource problems, except if we make students work in groups. However, after two years of experience with group work in a closed lab setting (3 persons/group), we cannot affirm the results of [9] that students learn more in group work. We suspect that the effect of group work depends on the tasks posed to the students: Large and difficult tasks that cannot be done by one member alone make for good teamwork and allow students to learn more in the team; simple tasks, however, are more conducive to one student doing the work and the others just watching. The student doing the work possibly benefits, but the onlookers will lack essential practical programming and debugging skills at the exam. Since we focus on simple tasks that capture the essentials, we observed that group work is detrimental to overall student performance and also brings the danger of students over-assessing their skills – after all, they put in a lot of hours watching and feel that it should suffice if they know how to do it in theory. For many students, the exams are real eye-openers in this regard.

So closed labs and group work are out. On the other hand, we also have experience with an open lab and voluntary individual work, which resulted in students putting off the work as long as possible and then getting into trouble with the exams due to lack of practice. Also, retention in this course was terrible, only about 30% of the students remained until the end of the course, because most failed to acquire the necessary skills in time. Furthermore, the lack in supervision resulted in exercise solutions that appeared to work, but were pretty convoluted and sometimes even wrong. Yet, students thought that their solutions were right and without feedback had no way of knowing otherwise. So just doing open labs was not a viable option either.

To get the best of both worlds, we decided on a mixture of open and closed labs: Students are expected to work at home, but each student is assigned to a supervised lab hour once a week. Attendance is not compulsory, but highly recommended. The lab supervisor (tutor), a senior student, is assigned 5 students per hour, and is there to answer their questions, help with debugging problems, check whether students are on schedule with their homework, and generally lend an ear in case of problems and keep up the students' motivations. We found that the tutors have a tremendous influence on the motivation and performance of the students and are in fact the pivotal factor in determining the motivational impact of a course.

Since the lab hour is not for doing work, but for discussing problems and asking questions, it could easily be conducted in a distance learning setting via Webcam. The important thing here is that the student is encouraged to regularly report about his or her progress, and that the tutor keeps track about each student's progress and motivation and can react to problems at an early stage.

D. Personnel

In order to minimize personnel costs and to provide scalability, the course uses a hierarchical system for student supervision: The course is managed by one assistant professor, one teaching assistant, and $\lceil n/15 \rceil$ tutors, where n is the number of students in the course[2].

Tutors work for three hours a week and meet with 5 students per hour. In this time, they talk with their students about their progress, check their work, answer questions, and determine how well the students have mastered the material so far. They should also do a preliminary grading of their students' homework (check whether the programs work and whether the protocols are in order). The work is very challenging for the tutors, so this job requires highly motivated students. It is important to be on the look-out for such students during the term, and to actively interest them in a tutor job for the following year.

The teaching assistant, who works for 18 hours per week, is responsible for the lab equipment, for organizing the exams, and for the final grading of the homework (after checking for plagiarisms). The TA is also the first address for tutors with questions or problems.

The assistant professor is responsible for the course contents and organization, for the course materials, for the exam contents, and for exam grading. Since we use electronic lectures which only have to be done once, after course setup is complete the course poses only a minimal load on the professor.

IV. Grading

For grading, we must ensure that diligent students can use homework to balance bad or missing exam results, while at the same time not allowing dishonest students to pass with somebody else's work. To this aim, we do both theory and programming exams, and we require that all students should get at least two out of the six programming exam tasks right. Students who fail to meet this requirement do not get a passing grade, no matter how many points they attain. This ensures that students who pass the course have some basic microcontroller programming skills. For an excellent grade, students must also

[2]This maximum allowable number of tutors is prescribed by the computer science faculty of the TU Vienna.

show that they have a grasp of the theory, so attaining at least one third of the theory points is a prerequisite for an excellent grade.

A. Programming Exams

Our programming exams consist of two tasks each. Students get two hours time, starting with a half hour design phase without computer, followed by the programming (and testing) phase on the actual lab hardware. To attain points for an exam task, students have to write a correct program. Since no partial credit is given, the tasks are very easy and generally consist of at most 20 lines of code. Students are provided with a framework, consisting of an Assembler or C skeleton program, possibly header files, additional object files, and a Makefile, and just have to add their code to the skeleton to complete the task. This allows us to place our exam tasks within more complex settings while still giving simple student tasks. For example, an exam task could be:

> **Task:** Create a program which acts as a frequency measurement routine. Configure the *Input Capture Function* of Timer/Counter 1 to determine the frequency produced on pin PD7.
>
> The frequency is in a range between 10 Hz and 99 Hz, so use prescaler 256. Display the ten's place of the measured frequency value on the display, using the provided function *display_result()*.

Here, the students just have to set up the input capture feature of a particular timer and compute the frequency. All other functionality, like displaying a number or generating the input function that should be measured, is provided by the framework.

Tutors are available during the exams to help students with hardware trouble. Since hardware-near programming is prone to simple but hard-to-find bugs, the tutors are also allowed to help with debugging problems, but only if the student can demonstrate that he or she understands the material and just suffers from a blackout.

Right now, the correctness of the student programs is checked manually by the tutors. To help the tutors assess whether the register initializations of the timer and other components are correct, we add some functionality to the provided framework to regularly check the register configurations and put the results on the LEDs. Thus, the tutor can see at a glance whether the register initializations are correct, or whether there are missing and/or superfluous bits. In addition, we are currently working on an automatic external test system which can provide the microcontroller with specific test stimuli and verify its responses. Since the software check outputs its results on the LEDs, these results can easily be taken over by the external test system, which can thus perform a completely automatic correctness check. We expect to have this system working by next summer.

Doing the practical exams on a computer with the actual hardware has both advantages and drawbacks. On the positive side, students can try out their programs and have the opportunity to debug them. This in turn allows us to use an all-or-nothing grading, which definitely would not be accepted by students were this just a paper exam. As another benefit, the exam results are instantly available, no later grading session is necessary. On the negative side, since students generally pass on their exam tasks to the colleagues that come after them, one either requires a lot of PCs or a lot of different exam groups. Since coming up with good yet simple exam tasks is quite hard, the first is preferable if the computers and hardware are available.

To allow a distance learning setting, we need to give students, who are sitting the exam at some external location, remote access to the automatic test system. We should also provide them with remote tutor help via Webcam. We are currently working on this problem in the SCDL project. Note that as soon as the exam can be conducted remotely, a large number of students can sit the exam concurrently. The persons supervising them do not have to be trained, only the tutors at the university giving the debugging help do.

B. Theory Exams

For our theory exams, we decided to use exams with yes/no questions instead of the traditional essay exams. Our decision was led by our desire for a fair grading scheme and by the fact that such exams can be graded automatically. Hence, we conduct our theory exams on computers, and the evaluation is done automatically as well. Originally, our exam questions were classical multiple choice questions in the form of a root (question) and several stems (answers) [10], but we gradually shifted away from this exam type and now use topics with several independent yes/no questions each. First of all, it broadens the range of material that can be covered with the exam (not every topic can be covered by a question with multiple plausible answers). As a nice side effect, we found that students like this kind of exam better as well.

Correcting the exams is done automatically. A correct answer is worth $+1$ points, a wrong answer gets -1, and no answer get 0 points. In consequence, if the maximum number of attainable points is N, then a student can get between $-N$ and $+N$ points on the exam. A negative exam result is set to zero. Guessing is neither punished nor rewarded by our system: On the average, a student will neither gain nor lose with guessing. However, we strongly discourage students from guessing. At this time, we do not feel the need to actively punish guessing by making the penalty for a wrong answer any larger, but we could of course do so. However, since this also punishes students who did not guess but simply got the answer wrong, we decided to refrain from using this means to curb guessing as long as we get the impression that (most) students try not to guess anyway.

To improve the quality of our exams and to increase the students' trust in our questions, we conduct a statistical evaluation of the exam papers prior to grading the exams, and we make students aware that we do this. For each question, our statistic calculates the percentage of students who got

the answer right, the percentage who got it wrong, and the percentage who abstained from answering. Questions with a high percentage of wrong answers or abstentions are carefully examined for their appropriateness, and are removed from the exam if we belatedly find them controversial. With this method, we have caught a fair share of ambiguous questions which slipped by our initial scrutiny but were identified by the number of students who had trouble with it. As a side benefit, this method also points out errors in the question database. We highly recommend this procedure for anyone who wants to do automatically evaluated exams. It does not cost much time, but makes the exams fairer and increases student satisfaction.

The theory exams can be graded very efficiently. Even with the additional effort of the statistic, we grade all exams (about 100) in a couple of minutes, and would have no problems at all to grade 1000+ exams either. A general advantage of our electronic exams is that they can be conducted anywhere where a computer and network are available. So it is no problem if such an exam is conducted at another university, or even at a high-school, as long as a trusted person is supervising the exam. As a drawback, one needs either many computers or a lot of different exam questions.

C. Homework

In addition to exams, students do some exercises from a larger set as homework. In order to get credit for their work, they have to submit a working program and a protocol describing their work and answering some questions about the subject.

The homework is the only part of the grading that does not scale well and is easily exploitable by dishonest students. However, it greatly increases the flexibility of the grading system, is beneficial to the student's development of practical skills, and thus vastly improves student motivation and satisfaction. Therefore, we include graded homework despite all its problems.

To lessen our workload, we need homework that can be corrected mostly by tutors. Here, the microcontroller course benefits from its low-level approach: The exercises generally are not complicated. Thus, a well-trained tutor can check whether a program works, and can also give pointers if the solution is not programmed well. The same goes for the protocol: The tutor can check whether the answers to the questions are correct, and whether the protocol is well-written. This takes the bulk of the load off the instructors, who only have to verify the first assessment of the tutors to assign the grade.

For example, a simple homework task would be to explore the effects of a floating pin:

> **Task:** Connect SW1 to pin PA4, set PA4 to input and do not activate the pull-up. In the main loop, display register bit PINA4 on LED0 and bit PORTA4 on LED1.
>
> **Questions:** Turn SW1 to OFF. Wave your hand closely over the board, tap or wiggle the wire connecting SW1 to the controller. What do you see on the LEDs? Why?
> Set SW1 to position ON. Repeat the actions of the previous question. What do you see on the LEDs now?

This exercise demonstrates the effects of a floating pin (when switch SW1 is turned OFF, the line floats; when it is ON, the line is grounded). Students observe the volatile nature of the power levels on the wire when they wave their hand over it or when they touch it, and are encouraged to speculate on the technical reasons for the observed effects. The exercise helps them identify floating problems, which occur quite frequently, in future programs.

Our most complex exercise task is to measure and plot the speed curve (acceleration and deceleration phases) of a dc motor for different PWM ratios. The students have to program the microcontroller to gather the data, send it over the serial interface to the PC, and create a 3D-plot with gnuplot. In their protocol, they have to interpret the resulting plot. This exercise forces students to put thought into program design, induces them to reuse previously written code, and thus gives the students a taste of complex application program development.

The major problem of homework is plagiarism. This can either take the form of copying (student A copies the work of student B, possibly slightly modified), or substitution (student A writes the homework for student B). Unfortunately, there is no sure-fire means to verify whether a student has done the work by himself. Even oral exams may not do the trick, since the student could have been tutored by the author of the code. Plagiarism detection mechanisms [11] can be used, but they too cannot identify cases of substitution. Therefore, our main means to catch such students is our requirement that students complete two exam tasks for a positive grade. This makes sure that students passing the course can write simple programs by themselves. A similar approach has been taken for example by [12]. Of course, this does not catch students who know how to program and are just too lazy to do the work by themselves. Therefore, we additionally check for similarities in the program codes and the protocols. Right now, we do this manually, which is extremely time-consuming. We thus strongly recommend to use an automatic tool for this purpose, and we are currently looking for suitable tools.

With our lab kits, homework is naturally suited for distance learning settings, since students can work anywhere. Electronic submission should not be a problem either, but in this case we should also provide some means for automated program verification, both to relieve the tutors from the load of having to check the correctness of all student programs and to allow the students to check the correctness of their programs prior to submission. To allow students to get feedback from their tutors on their submitted work, tutors should be available via Webcam to discuss the program and the protocol. So students can submit their work anytime and can get feedback from their tutor in his or her next (virtual) lab hour.

V. Discussion

This year's course was on the whole well received by our students. Student favorites, in descending order, were the quality and friendliness of our tutors, the take-home lab kits, the interesting and diverse hardware, the possibility to get bonus points for additional work, the large exercise set with its varied exercises (and its availability from the start of the course), and the flexible grading system.

Students criticized the fact that we do programming exams, their all-or-nothing nature, and that we demand two completed programming exam tasks for a positive grade.

Suggestions for improving future courses included additional and more complex hardware, and to be able to take home all the hardware (the motor boards were only available in the lab this year).

We do of course try to improve criticized areas, but cannot do much about our programming exams. Neither do we really want to change our procedures. After all, our exam tasks are kept fairly simple to ensure that prepared students have sufficient time, and the tutors are allowed to help to prevent students from failing just because of a simple bug they were too nervous to find. Our impression is that the students take the exams too lightly and do not appreciate the fact that just getting a homework exercise to run with the help of the tutor and a lot of trial-and-error does not make them proficient microcontroller programmers. An unfortunate aspect of our on-target programming exams is that students are generally convinced that they were "just five more minutes" away from a correct solution, which makes them feel the lack of partial credit all the more[3]. We believe that we can overcome this problem with better exam preparation, and thus in the future will ask our tutors to assess the students' current level of knowledge regularly.

Since the homework is beneficial to students, we allow it even though plagiarism may occur. Of course, we try to identify plagiarism, and in future courses will emphasize more why plagiarism is detrimental to performance. A study conducted on patterns of plagiarism [13] indicates that plagiarism is mostly done by weaker students, possibly because of problems with handling the work. We hope that regular supervision and support by the tutor can help weaker students overcome their problems without resorting to plagiarism.

To improve our motivational impact without too much financial backlash, we strove to shift as much supervision load as possible to tutors. It turns out that tutors can handle the bulk of the student supervision, as long as they in turn are well supported. Here, we still lack a few tools that would help tutors a great deal, foremost a tool to manage the student data (student name, progress, homework submissions, submitted voluntary work, ...). We are currently working on such a tool in our SCDL project and expect it to be ready by next year. The tool should also be able to handle electronic submissions and check for plagiarism.

It is also important to prepare tutors properly for their work. They must of course be trained on the exercise material, but they must also be made aware of their social duties. For instance, they must understand why they should monitor the performance of their students and play an active role in their development. We must also call to their attention the problems of plagiarism and their obligation to try and prevent such things among their students. This is of particular importance because tutors, as senior students, are liable to have "helped" their own friends one time or another, and are thus vulnerable to turning a blind eye when encountering such activities among their own students. So care must be taken to explain to tutors why plagiarism hurts the person that plagiarizes.

We never really addressed these issues in former years, but always chose tutors that we believed were liable to have a good impact on students. By actively making tutors aware of their social role and explaining their impact on student motivation and grades, we hope to improve tutor motivation even more.

Since student satisfaction with our course is very good, student retention is quite high (70%). Of the students who remained in the course, 73% passed. These are acceptable values, but we hope to increase these numbers even more by improving the quality of support. The hardware in its current form is well suited to convey the course contents. To increase the fun aspect of our course, we plan to introduce a line-following robot into the third part. It will be used to teach motor control and power-efficient programming issues.

Although we are currently not conducting the course in a distance learning setting, we could easily do so. Due to the lab kits, homework can be done at home. All course material is contained on the Knoppix CD that is part of the lab kit, so students have all information they need at home. The supervised lab hour can easily be offered via Webcam. The theory exams are already conducted via Internet, and since the exam supervisor does not have to help students, any exam room and supervisor would suffice as long as it is guaranteed that students cannot get illegal help during the exam. The programming exams currently need skilled supervisors to help with debugging problems, but this might also be achievable with a Webcam, so as soon as our test system to automatically evaluate the correctness of student solutions is available, the programming exams can be conducted via Internet as well.

Although due to some hardware problems we do not yet have experience with the electronic lectures, some students already told us that they think it is a great idea. We will try to make each lecture as self-sustaining as possible, complete with indications on which other lectures it relies, to allow other courses to use our lectures as necessary. In turn, we believe it would be useful to generate a faculty-wide database of such lectures, possibly even with exam or self-assessment questions about the material, so that instructors can easily assemble material students should brush up on for their course.

[3]It is interesting to note that in the cases where we followed up such claims, it always turned out that the student solution had one or more serious errors and was far from being complete.

VI. Conclusion

The course in its current form is successful, student satisfaction is high. Our course structure has proved effective, students like the take-home lab kits, the flexible grading scheme, and the homework exercises. The course can easily be exported to other universities or can even be held in a distance learning setting.

To keep the course scalable and efficient, we employ a hierarchical structure for supervision, with 1 tutor per 15 students and 1 TA for all tutors. Lectures are provided electronically, so after the setup phase of the course, the professor only has to handle the course organization and assign the final grades.

Final scores are composed of programming exam results, theory results, light compulsory homework, and voluntary bonus homework. A certain minimum score on the programming exams is necessary to attain a positive grade to prevent unprepared students to pass by plagiarizing their homework.

There are still some open tasks which can make the course more self-supporting. First of all, we intend to expand our set of electronic lectures to include video introductions to the hardware, and generally introductions to all subjects students have problems with (e.g., C programming). Second, we will continue to develop our automatic test system, which is currently only available as a prototype, and work on the web cam approach so we can eventually conduct all exams remotely and without trained supervisors on site. We are also currently looking for suitable plagiarism detectors to strengthen the value of our homework exercises, which can in theory be exploited by lazy students to get a better grade. Finally, we need a suitable course management tool that also supports tutors in their work.

References

[1] W. Wolf and J. Madsen, "Embedded systems education for the future," *Proceedings of the IEEE*, vol. 88, no. 1, pp. 23–30, Jan. 2000.

[2] B. Haberman and M. Trakhtenbrot, "An undergraduate program in embedded systems engineering," in *18th Conference on Software Engineering Education and Training)*, Apr.18–20, 2005, pp. 103–110.

[3] MCLab, "Microcontroller homepage," 2005, *www.ecs.tuwien.ac.at/lehre/Microcontroller/MCLab.shtml*. [Online]. Available: http://www.ecs.tuwien.ac.at/lehre/Microcontroller/MCLab.shtml

[4] C. E. Nunnally, "Teaching microcontrollers," in *26th Annual Frontiers in Education Conference (FIE'96)*, vol. 1, Nov. 6–9, 1996, pp. 434–436.

[5] M. Amirijoo, A. Tešanović, and S. Nadjm-Tehrani, "Raising motivation in real-time laboratories: The soccer scenario," in *35th SIGCSE Technical Symposium on Computer Science Education*, Mar. 2004, pp. 265–269.

[6] J. W. McCormick, "We've been working on the railroad: A laboratory for real-time embedded systems," in *36th SIGCSE Technical Symposium on Computer Science Education*, Feb. 23–27, 2005, pp. 530–534.

[7] R. Bachnak, "Teaching microcontrollers with hands-on hardware experiments," *Journal of Computing Sciences in Colleges*, vol. 20, no. 4, Apr. 2005.

[8] A. N. Kumar, "The effect of closed labs in Computer Science I: An assessment," *Journal of Computing Sciences in Colleges*, vol. 18, no. 5, pp. 40–48, May 2003.

[9] L.-K. Soh, A. Samal, S. Person, G. Nugent, and J. Lang, "Closed laboratories with embedded instructional research design for CS1," in *36th SIGCSE Technical Symposium on Computer Science Education*, Feb. 23–27, 2005, pp. 297–301.

[10] B. Weiss and G. Gridling, "Creating and grading multiple choice tests," Vienna University of Technology, Institut für Technische Informatik, Research Report 63/2004, Dec. 2004. [Online]. Available: http://www.vmars.tuwien.ac.at/frame-papers.html

[11] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, "Shared information and program plagiarism detection," *IEEE Transactions on Information Theory*, vol. 50, no. 7, pp. 1545–1551, July 2004.

[12] K. W. Bowyer and L. O. Hall, "Experience using "MOSS" to detect cheating on programming assignments," in *29th Annual Frontiers in Education Conference (FIE'99)*, vol. 3, Nov. 10–13, 1999, pp. 13B3/18 – 13B3/22.

[13] C. Daly and J. Horgan, "Patterns of plagiarism," in *36th SIGCSE Technical Symposium on Computer Science Education*, Feb. 23–27, 2005, pp. 383–387.