A New Curriculum for Teaching Embedded Systems at the University of Ljubljana

Tadej Tuma and Iztok Fajfar
University in Ljubljana
Faculty of Electrical Engineering
Tržaška 25, 1000 Ljubljana, Slovenia, EU
Email: tadej.tuma@fe.uni-lj.si, iztok.fajfar@fe.uni-lj.si

***Abstract*— In recent years the population enrolling university studies in Europe has topped 50%. At the same time we are witnessing an ever faster technological development in the area of embedded systems. All these changes called for an urgent response to make the teaching of embedded systems more attractive and affordable to a wider population. As a result we have completely redesigned our curriculum. Not only have we utilized new technology, we have changed the entire approach to teaching.**

In the past we started with assembly language, which we considered basic, and only later moved to embedded C. From a motivational point of view this was not ideal, since the gap between the assembly language and the students' pre-university computer experiences, which mainly involve using Windows-like applications and Internet, was getting too wide. Higher computer languages were taught in parallel without an immediate connection to embedded systems.

Our new set of courses starts out with JavaScript, which feels very much like C, is free, and motivationally proved an instant success. Next comes the transition to embedded C which is now much less painful and the assembly language level is left for special courses in later semesters, addressing specific hardware issues.

As for hardware platforms, we wanted students to have their personal powerful embedded development system, whose price should not exceed an average textbook. Since professional embedded development systems are far from being cheap, at first that sounded a bit unrealistic. However, we came up with a special concept, which attracted sponsors from industry, so we could raise the necessary funds.

Most courses related to embedded systems now use the same platform with only different add-on boards, which further reduces the per-system price and getting-started overhead. Since the students can do much of their work at home, the after hours spend in the laboratory are less which frees our resources to other activities.

I. Introduction

Most universities in the EU are state funded and consequently their curricula need to be government approved. This ensures a certain quality level and maintains compatibility. The down side, however, are relatively rigid curricula, since any major change needs to pass tedious bureaucratic procedures. The Faculty of Electrical Engineering at the University of Ljubljana, Slovenia [1] is no exception.

The fast developing area of embedded systems is especially affected by the lack of curriculum flexibility. During the past decade the contents of many courses were extended by microcontroller related topics. However, this was done by individual teachers without an overall concept, and only within the approved curriculum structure. The result was a patchwork of different microcontroller related courses on different levels. Accordingly, laboratories were based on all sorts of hardware platforms, so students had a lot of learning overhead to switch back and forth between different microcontrollers. Further more, the lack of coordination necessarily led to overlapping course contents which further reduced the teaching efficiency. There was also a traditionally disjoined relation between teaching general purpose programming languages, which are considered hardware independent, and hardware specific embedded system topics.

Anticipating a major restructure according to the Bologna declaration, which is due in a couple of years, the faculty decided to preempt some urgently needed reforms regarding the embedded system curriculum. Our ambition was to significantly increase the teaching efficiency in this field and remedy all above shortcomings. This may seem rather ambitious, but due to the rigid nature of our curriculum structure it was now or never. Once the enforced Bologna changes are clad in stone, there will be no more room for major restructures.

II. Software Flow

What is the ideal way to introduce embedded system programming to EE students? The answer to this question depends very much on the time asked.

The classic approach definitely starts with the lowest hardware level. The CPU registers, address bus, data bus, and memory are discussed in connection with a basic assembly language instruction set. Sometimes even instruction format, execution cycles, and ALU logic would be explained by way of introduction to microcontrollers. In other words, programming is introduced in parallel with the executing hardware.

Two decades ago this was definitely the way to do it, since the microprocessors of those days were very simple. Also the limited functionality did not allow for a higher language approach. Consequently, the debugging utilities were simple command line based monitors. Higher programming languages would be taught separately, mainly intended for desktop computers and mainframes.

Today, the situation has dramatically changed. Modern microcontrollers are highly sophisticated in design and functionality. The development systems easily implement embedded

C, or even C++ and compilers with user friendly debugging GUI environments. There is no need to start students off on the assembly level. Moreover, it has become extremly difficult to explain the complex machine level. Therefore it makes sense to skip the registers and start at a hardware independent level with a modern C based development system. The assembly language appraoch is by no means superfluous, since this is the only way to show the students what exactly is happening on the machine level. Many times critical real time primitives are still coded in assembly language. However, this is not the approach for a novice any more. The assember language approach has rather become a speciality and should be tackeled in later courses. Having this concept in mind, it becomes also obvious that higher languages are not to be taught separately any longer. Instead, there should be a seamless transition from an abstract language like JavaScript to embedded C, and only later the hardware dependent assembly language approach should be taken on.

So the ideal flow of contemporary EE software teaching would roughly be an up side down version of the former flow. At the faculty we decided to implement the following three stage curriculum.

1) General purpose programming languages. In the first semester the students start off on a simple C-like language, preferably a popular web based language so they can immediately enjoy the fruit of their labor by designing dynamic web pages. Our choice fell on JavaScript.
2) Embedded C programming. Already in the second semester the students are taken to an embedded C running on a contemporary microcontroller platform. The transition from JavaScript to embedded C comes natural, without going too much into hardware details. At this stage the hardware target is little more than a motivation factor. Some LEDs, few of buttons and simple LCD are enough to provide incentive.
3) Embedded OS programming. After the two obligatory courses in the core curriculum, the students choose between several specializing curricula branches, each with its own set of specific embedded system related courses. These venture into hardware specifics, assembly language as well as introduce real-time and multitasking environments. Very importantly, most of these courses build on the hardware platform which has been presented in the second semester.

A. General Purpose Programming Languages

Many years' experience with teaching computer languages and architecture to the first-year students show some interesting if not surprising results. Even though we are dealing with electrical engineering students, we have observed a considerable fear of computers on the part of students. For many first-year students to be, the computer is simply a tool for accessing the Internet, and playing games. The rest is wrapped in mystery. As a consequence, trying to teach assembly language or even higher level languages, such as C, has proved more and more difficult through the last half decade. For many students, already unable to grasp the concept and sense of computer programming, a single missing semicolon, producing unreasonable list of errors, could be a big enough frustration to quit any further efforts altogether.

Therefore, we chose a language according to the following criteria:

1) First steps should produce instant results with as little chance of failure as possible. We wanted to introduce a concept of "speaking" to the computer in a form of a simple text. This has proved a very important step for many students. Without the danger of getting tangled in different conditional and loop statements, compiler warning and error messages, students feel free to explore the familiar effect of an unfamiliar language.
   A markup language such as XHTML proved an appropriate candidate for that stage.
2) Very soon students have enough nerve to try and transfer to the computer some of the burdens of tedious typing of XHTML tags. Producing simple tables using JavaScript code is the next logical step. Unobligatory details such as declaring variables and putting semicolons at the end of statements are required by the lecturer whereas the computer is more forgiving. It is a matter of debate whether this is good or bad, but our experience has shown that the allowed sloppiness enabled student in general to focus more on the gist of coding, i.e. explaining the well formed idea to the computer.
3) Also very importantly, our language should resemble C as closely as possible, making the transition to embedded C as seamless as possible. Again, JavaScript has proved the best candidate.

In summary, a combination of elementary XHTML and JavaScript formed an abstract yet practical framework for introducing basic concepts of computer programming.

As an example let us look at a simple code fragment producing a series of thumbnails in a browser window:

```
var i;
for (i = 0; i < num_thumbs; i++)
{
  document.write(thumb[i]);
}
```

Since students are familiar with pages displaying thumbnails, they in general appreciate and understand the benefits of using the for loop for producing such a page. This practical understanding motivates students directly for the deeper study of the logic of for loop itself.

B. Embedded C Programming

In the second semester the students plunge into embedded C programming. They are already quite familiar with a basic syntax, program control structures, concept of calling and defining functions, and devising simple algorithms. The semester starts with pointing out most important syntactical differences between the languages JavaScript and C. For that

purpose we use a hardware platform with very rudimentary input and output, and without an operating system. The most important difference stems from the lack of the operating system: the hardware units used need to be initialized and our program must retain perpetual control over the system. Some other differences we notice at that stage are the consequence of strict typing rules of the C language.

These differences are not very difficult to grasp for the students, and we quickly move on. Next thing they learn is the concept of a real-time programming. The case we study is of course the most simple and intuitive. We use polling with the assumption that all tasks execute within the required time slots.

Connecting external simple hardware devices such as keys, sensors, and stepper motors is the next important step we take. Apart from some basic physical phenomena like bouncing, students learn that from the programmers point of view the problem of elementary controlling of such devices is in fact trivial.

Even more control over hardware is possible when we learn binary coding principles and direct addressing of the hardware registers.

The next example shows, how the problem of rotating a stepper motor is surprisingly similar to the problem of displaying an array of thumbnails we met in previous section:

```
int i;
for (i = 0; i < rot; i++)
{
  outportb(switchseq[i % 4]);
  delay(20);
}
```

One just needs to replace the array of references to thumbnails with the array containing a four-step switching sequence. Due to the mechanical limitations a small delay between switches is also required.

C. Embedded OS Programming

The curriculum at the Faculty of Electrical Engineering in Ljubljana, Slovenia, basically consists of four common semesters covering all fundamental EE topics followed by several specializing curricula branches [1]. The later can be roughly divided into four groups: Automatics, Electronics, Power Engineering and Telecommunications. All four groups include microcontroller based courses focusing on specific embedded applications. Typically, these would involve systems for control in robotics, power transmission, RF electronics, etc. So the notion of real-time multi-task programming is introduced at different levels. Either the courses discuss respective programming techniques or they build on embedded operating systems like μSmartX, which was developed by one of our post graduate students, and is freely available on the web [3].

Of course all advanced courses engage students in practical project work. So far these projects have been based on arbitrary microcontrollers so there always was the typical getting-started-overhead. Also, the specific expensive equipment required the students to work in the laboratories on campus. With our new approach, the overhead is almost nil. Moreover, since the students have their own development boards, a considerable part of the project work can be done at home.

However, it is of utmost importance that the development board be powerful enough and flexible enough to allow the docking of any advanced hardware boards. This has been achieved by an inventive concept, as explained in the following section.

III. Hardware Platform

We are teaching embedded system knowledge in general but when it comes to giving students practical skills one necessarily needs to resort to one specific microprocessor. This is just like getting a driving license. The goal is to acquire the skill of driving a car, any car. But you have to practice on one specific model. Although you will drive different cars in your life, we believe it is inefficient to switch back and forth between different car models while still in drivig school. With teaching embedded systems it is no different, we need an affordable and robust workhorse to practice.

In the previous section we have already hinted at the idea of constructing a common hardware platform for second and third stage. The design specifications are tough. The development system obviously needs to be very flexible in order to accommodate simple user friendly sessions in the second semester as well as all semi professional requirements of higher level courses.

On top of all this we want students to have an opportunity to buy their very own development system right from the beginning. Using the comparison to the driving school once more, it is clear that a student having his/her own car right from the beginning will be higher motivated, will be able to work after hours and will keep driving the same car after passing the license test.

A. Development Board

Looking for an all around workhorse between contemporary microcontrollers, we decided to take our chances with the ARM7 core by Philips. We're speculating that this technology will be around for at least one decade. In order to keep cost as low as an average textbook and still meet professional standards we had to get sponsorship backup right from the beginning. However, in order to attract the attention of potential sponsors we had to present a faculty wide support for the project. This was a classic chicken-and-egg situation since the enthusiasm of participating teachers on the other hand very much depended on the price/performance of the development tool. After much negotiations on both sides a strong consortium of six companies was ready to develop and finance our new ARM7 development board.

According to the three stages identified in section II we designed the basic development module as depicted in Fig. 1.

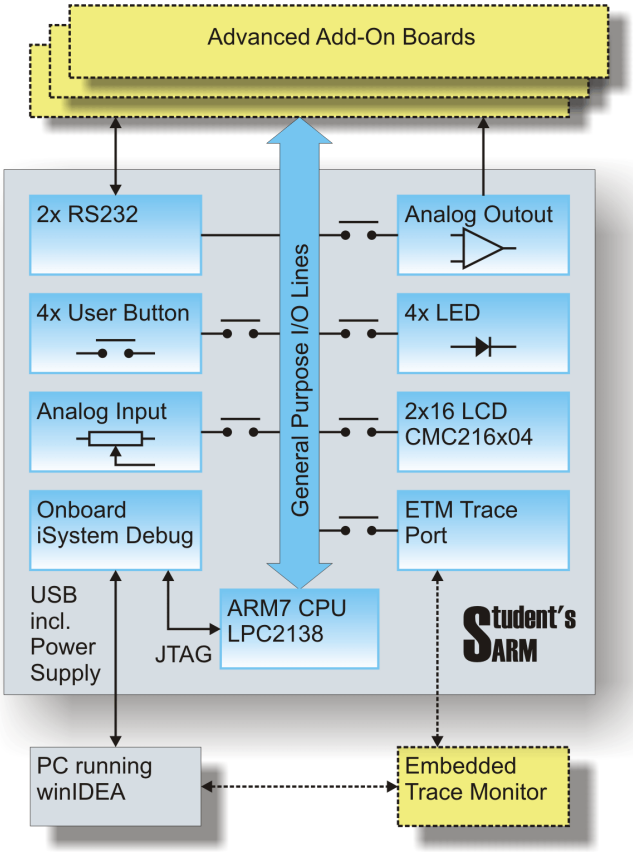

Fig. 1. ARM7 Development system overview.

The highlight is the integrated on-board debugging hardware linking the ARM7 CPU to the well known professional development environment winIDEA^{TM} by iSystem [2] which is running on any standard personal computer. The PC is connected via USB and is providing the necessary power supply as well.

In this way we can offer full functionality of the entire development system to our students. The proprietary software on the PC is locked to the on-board debugging hardware in order to prevent unauthorized profesional use of the system. This is an original concept protecting the copyright of winIDEA^{TM} and giving the students full development power at the same time.

The development board in fig. 1 obviously has powerful debugging capabilities but very limited input/output devices. This is because we need to keep the initial costs as low as possible. Remember, the system is introduced in the second semester in support of teaching embedded C. It should provide just basic incentive for novice students. To this end we have included several very simple I/O devices. There are four keys, four small LEDs, a potentiometer at an A/D input, a general purpose opertional amplfier on a D/A output, a pair of RS232 serial ports and the facilities to mount a standard LCD piggyback. This is more than enough for a beginner course, advanced level course on the other hand require specific devices.

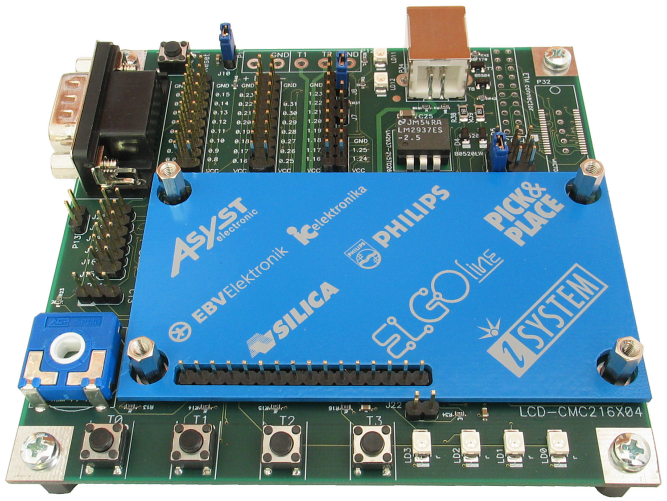

Fig. 2. Student's ARM7 Development board.

To accommodate these needs we have provided respective connectors of all CPU ports. Any number of sophisticated add-on boards can be attached to these connectors. For minimal interference with professional add-on equipment all on-board I/O devices except for the serial ports can be disconnected by jumper settings. Individual teachers are designing add-on boards for their specific needs in smaller quantities. Senior students are encouraged to experiment with add-ons in their project work. Many master theses are based on development and testing add-ons.

Optionally, an external embedded trace monitor can be connected to a special port, enabling students to trace their programs in real-time. This, however, requires relatively expensive additional hardware.

From a physical point of view the developemnt board is manufacutred in SMD technology, based on a four layer 10 by 10 cm PCB as seen in Fig. 2. In front, the four buttons and LEDs are visible. All ICs are covered by the blue plate, which serves for protection and for sporting the sponsor logos. The LCD piggyback is mounted over this area as well.

Thanks to our sponsors we are able to offer this board, including USB cable and winIDEA^{TM} software to our students at a price less than 40 EUR. In fact the presented development board has become quite popular, so our sponsor iSystem is now offering it world wide as their LPC2138 evaluation board [2], demonstrating the capabilities of winIDEA^{TM}.

B. Integrated Development Environment

As mentioned in previous section, we use winIDEA^{TM} as an Integrated Development Environment. Since the software is locked to the on-board debugger, we are able to distribute a full version of the environment. It turned out that students quite appreciate the fact that they can work on a fully professional system at home. This is a strong motivational factor for them as well as for sponsors. They understandably expect that many electrical engineers will want to use exactly the same software

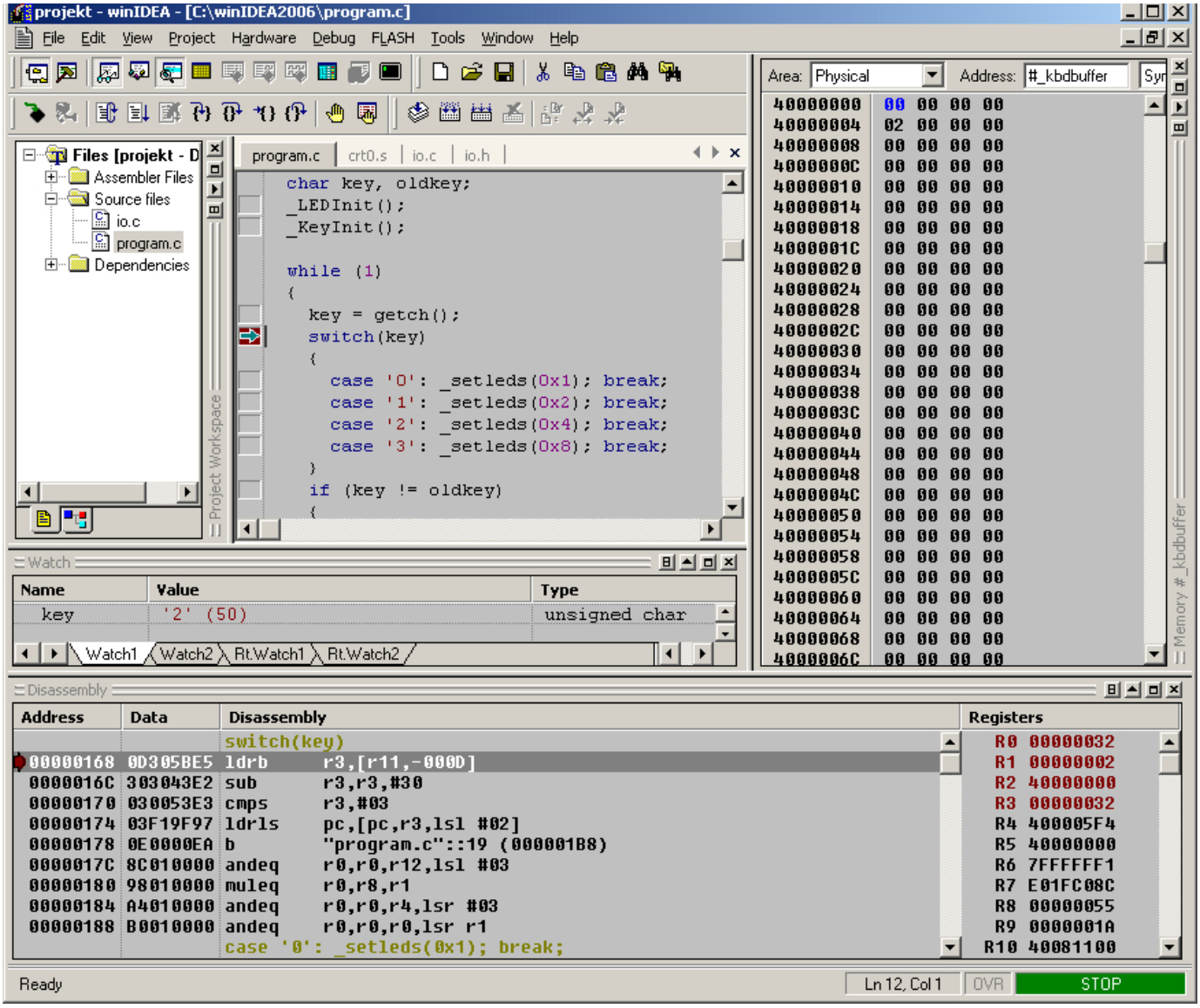

Fig. 3. The winIDEA^{TM} integrated development environment.

in their professional life after graduation. This belief is secured by the saying that old habits die hard.

Fig. 3 shows a running winIDEA^{TM} environment. We can see some basic elements of the environment such as source code and watch windows. The execution of the loaded program has stopped at a breakpoint and the user is able to observe the value of the variable *key*. The important fact is that the program is running on the target board. After two single steps through the code one is able to observe the third LED lighting as the consequence of the execution of the statement `_setleds(0x4);` This is extremely illustrative for an average first year student who still has difficulties grasping the sequential cause-and-effect concept of computer programming.

IV. First Experiences

It has been almost a year since we introduced the approach described in the paper, and we already have some qualitative and quantitative results regarding its success. The first thing we notice is drastical increase in students' interest in the subject already during the first semester. All of a sudden, out of their own initiative, many students are seeking further explanations and discussions on the subject even during the lecture breaks.

In exams, especially oral, where we test higher levels of abstraction according to Bloom's taxonomy, we noticed drastically raised levels of understanding of basic concepts that we have been teaching for more than two decades. This subjective observation was also partially confirmed in numbers. Table I shows percentage of students that passed the exam during the first examination period, i.e. during the first month after the end of the lectures, over the last five years. We observe a

Year	2001	2002	2003	2004	2005
Passed (in % of total)	28	33	29	34	55

TABLE I

PERCENTAGE OF STUDENTS PASSING THE EXAM DURING THE FIRST EXAMINATION PERIOD. WHEN WE INTRODUCED THE NEW CONCEPT, A SIGNIFICANT RAISE IN SUCCESS RATE WAS OBSERVED (YEAR 2005)

drastical increase in year 2005, when our new approach was introduced.

The results, however, are not surprising. Starting out on a too low level, which over the past years assembly language definitely has become, gives little motivation to the students. The gap between their experiences of everyday life and low level computing has simply become too wide. On the other hand, many students, already quite familiar with Internet, discover instant application of JavaScript in real life problems. This motivational factor is strong enough to lead many students effortlessly through the first, and for many the most difficult, part of learning computer programming. The next step, embedded C programming, turned out to be a natural sequel to the basic JavaScript programming.

V. CONCLUSION

The recent redesign of embedded systems curriculum at the University of Ljubljana is discussed. The reorganization is based on a three point action plan. First we aimed for a higher language based approach, which would integrate most microcontroller courses as well as classic hardware independent programming. Secondly, we wanted each student to have his/her own microcontroller development board right from the first year in order to be better motivated and to be able to work more efficiently. The third point was to involve industrial partners in the project by using professional tools and getting respective sponsorships.

The three components mutually depend on each other: without uniting a critical mass of teachers, no sponsor could be attracted. Without a generous sponsorship we could not offer embedded boards for each student. Without students having their own board throughout their studies it was not possible to have a wide cooperation between teaches. And so the loop is closed. In fact, the uniqueness of our new curriculum lies in our ability to break this dead loop by implementing all three actions simultaneously.

The new curriculum is in effect for only about a year so we don't have any long term feedback yet. The first experience however, is very encouraging. The only down side we can see so far is the fact, that our embedded system curriculum is strategically dependent on a single microprocessor architecture and a single development system.

ACKNOWLEDGMENT

The authors would like to thank the Ministry of Education Science and Sport (Ministrstvo za Šolstvo, Znanost in Šport) of the Republic of Slovenia for co-funding our research work through program P2-0246, Algorithms and Optimization Methods in Telecommunications.

REFERENCES

[1] Faculty of Electrical Engineering at the University of Ljubljana, Slovenia *www.fe.uni-lj.si/welcome-E.html*, 2006
[2] iSystem AG *www.isystem.com*, 2006.
[3] μSmartX, the free real time operating system for the ARM7TDMI platform *usmartx.sourceforge.net*, 2006.
[4] B. S. Bloom, *Taxonomy of Educational Objectives*, Handbook I: The Cognitive Domain. New York: David McKay Co. Inc., 1956.
[5] D. R. Krathwohl, B. S. Bloom and B. M. Bertram, *Taxonomy of Educational Objectives, the Classification of Educational Goals* Handbook II: Affective Domain. New York: David McKay Co. Inc., 1973.