A TOP DOWN APPROACH TO TEACHING EMBEDDED SYSTEMS PROGRAMMING*

I. Fajfar, T. Tuma, Á. Bűrmen, J. Puhan

Fakulteta za elektrotehniko, Ljubljana

Key words: embedded systems, higher education, computer programming, higher programming languages

Abstract: Over last decades we have been witnessing an ever faster technological development in the area of embedded systems. At the same time advances in IT and transition to mass higher education worldwide significantly changed the social and cultural environment in which we teach. We had to respond to these changes with updating the courses and making the teaching of embedded systems more efficient, attractive, and affordable.

The classical approach started with computer architecture combined with assembler. Higher level languages were taught independently. From a motivational point of view this has become questionable, since the gap between the assembler and students' pre-university computer experiences, which are mainly Windows-like applications and Internet, was getting too wide.

Our new curriculum starts with JavaScript, which resembles C, runs in environment familiar to any student, needs no additional software, and motivationally proved a success. Transition to embedded C is now much less painful. The assembler level is left for specialised courses in later semesters.

For the purpose of embedded C courses we developed special hardware platform. Each student can purchase their own professional system, at an affordable price. To achieve that, we needed a support from sponsors. Most courses related to embedded systems now use the same platform with only different add-on boards, which further reduces the price and getting-started overhead. That way students needn't spend too much time with unnecessary technicalies and can focus more on the subject. As a result, after only three years we have already received some quite positive feedback.

Pristop k učenju programiranja vgrajenih sistemov z vrha navzdol

Kjučne besede: vgrajeni sistemi, visoko izobraževanje, računalniško programiranje, višji programski jeziki

Izvleček: V zadnjih desetletjih smo priča vse hitrejšemu razvoju na področju vgrajenih sistemov. Obenem je tehnološki napredek in množično visoko izobraževanje v svetovnem merilu drastično spremenil socialno in kulturno okolje, v katerem učimo. Na te spremembe smo se morali odzvati s prenovo učnega pristopa in narediti študij vgrajenih sistemov učinkovitejši, privlačnejši in bolj dostopen.

Klasičen pristop začenja z opisom računalniške arhitekture v povezavi z zbirnikom. Višji programski jeziki, kot na primer C, se poučujejo neodvisno. Z motivacijskega stališča je postal tak pristop vprašljiv zaradi vse večjega razmika med zbirnikom in izkušnjami, ki jih imajo študentje z računalniki pred vpisom v visoko šolo. Te izkušnje zajemajo predvsem uporabo spleta in okenskih programov.

Naš nov pristop začenja z jezikom JavaScript. Ta je podoben jeziku C, deluje v okolju, ki je domače vsakemu študentu, ne zahteva dodatne programske opreme ter se je z motivacijskega stališča izkazal za zelo uspešnega. Naslednji korak je vgrajen C. Prehod na ta jezik je sedaj veliko enostavnejši. Zbirnik smo prepustili specializiranim predmetom v višjih letnikih.

V namen poučevanja vgrajenega Ceja smo razvili posebno strojno opremo. Vsak študent lahko kupi svoj lasten profesionalen razvojni sistem, čigar cena ne presega cene povprečnega učbenika. Cilj smo dosegli s sredstvi, ki so jih prispevali sponzorji. Večina predmetov sedaj uporablja enotno učno okolje, kar dodatno znižuje ceno in zmanjšuje količino uvajalnega dela. Vse to osvobaja študenta nepotrebnega poglabljanja v različne tehnične posebnosti in mu daje več časa za ukvarjanje z vsebino. Tri leta po vpeljavi novega pristopa smo že opazili nekaj zelo vzpodbudnih odzivov.

1. Introduction

An embedded system is a non-general computing system which comprises a microcontroller to implement some of its functionality. The area of embedded systems has undergone tremendous advances over the last few decades. Microcontrollers are getting cheaper and more powerful and are practically everywhere. It has been estimated that an average American came into contact with more than 100 microprocessors per day already a decade ago /1/, and this number is constantly growing. Apart from that, systems have become more sophisticated, running tens of thousands lines of code, and handcrafted approaches to developing small microcontroller applications of the past are no longer successful. More structured design methodology has become a must. All this rapid changes has made our embedded system curriculum dangerously outdated.

But before we plunge into designing a new curriculum we must not ignore the social and cultural changes, whose understanding is crucial to developing a successful educational programme. A higher education worldwide is facing a problem of transition from elitism to mass education

* Part of this research was presented at the 7th International Conference on Information Technology Based Higher Education and Training, held from 10th to 13th July, 2006, in Sydney, Australia

/2, 3, 4/. In Europe a significant growth of mass higher education has been observed over the past few decades, and Slovenia is no exception /5/. From the increased numbers stem many problems, like that of funding, organization, and of much changed conditions of teaching students with quite different motivation and academic talents.

Still another problem observed in all advanced societies is brought about by immense advances in IT development and extensive use of all kinds of multimedia technologies, especially TVs and computers, already from early childhood. These new video generations are less able to read, concentrate, and memorize /4/, which further influences the capacity of coping with abstract mathematical notations that university lecturers traditionally use to model real life systems. New means of communication has changed the very way information is absorbed and processed in students' heads, which is another challenge for traditional university teaching methods.

Paradoxically, already much increased number of students has at the same time intensified a struggle to attract more and more students to engineering sciences. Especially in western societies these sciences are amongst less popular, which too calls for an urgent change in teaching to attract women as well as some other underrepresented groups /6/.

All these changes are very hard to follow, in some part due to the inflexible European university systems. 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. 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, compilers and languages. Furthermore, 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.

Alongside a major restructure according to the Bologna process the Faculty of electrical engineering at the University of Ljubljana decided to carry out some needed reforms regarding the embedded system curriculum. Our ambition was to significantly increase the teaching efficiency in this field and remedy as much of the above mentioned shortcomings as possible. 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.

In the paper, we describe the design of new embedded systems curriculum, trying to cover the advances in the field itself as well as taking into account changed social and cultural environment. The course consists of three parts: introduction to general programming using JavaScript language, introduction to embedded systems using C language, and subject specific embedded system topics. We give some emphasis on developing embedded operating systems and the hardware development system we have developed for the educational purposes. At the end of the paper we comment some observations and experiences we have had over the last three years.

2. History, analysis of situation and preliminary steps

The first microprocessor course at the Faculty of Electrical Engineering in Ljubljana started already in the late 70s, short after an 8-bit Motorola 6800 was released. Because the 6800 was relatively simple yet complex enough to demonstrate all the basic principles of the microprocessor architecture and programming, and because of it's widespread usage, the 6800 (and some of it's improved versions like 6802 and HC11) was the core processor in almost all of our microprocessor oriented courses. The exception was an introductory course in the first year, where a so called hypothetic computer was used. It was an extremely simplified model of a real microcontroller using only 14 instructions and existed only as a simulated device running on a DOS. At that time that was quite a successful approach through which students got a basic idea of the happenings in an ALU, hardware registers, memory, and on a system bus when running a program. Later, this system was replaced by a real development board based on HC11, which was designed at our faculty. Before that, the same system was already used for sometime in higher level courses, but soon there emerged a need to replace old and increasingly uninteresting hypothetical computer. For that purpose we developed a special IO hardware and IDE with a graphical user interface, which enabled 1st year students to quickly write, compile, run and debug some simple programs. The programming language used was without exception assembly language.

There were also some lectures teaching higher languages, mostly C, but that was used for programming mainframes or PCs and many students saw little if any connection between both words. It even wasn't very rare that a student who was quite successful in mastering assembly language didn't come even close to passing the exam in C language and vice versa - that who had no difficulties with C language, was quite lost with assembler.

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 off on the assembly level. Moreover, it has become extremely 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 approach is by all means necessary, 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. Instead, the assembler language approach has become a speciality and should be tackled in later courses. Having this concept in mind, it becomes also obvious that higher languages are not to be taught separately any longer.

At first we thought that starting with a high level language such as C would not be a problem since the accessibility of computers should have contributed to generally higher level of computer literacy in students entering the electrical engineering studies. But surprisingly, as many years' experience with teaching computer languages and architecture to the first-year students has shown, students show a considerable "fear" of computers. 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. It was easier to teach students programming a decade or two ago from scratch than now when they have certain (misleading) predispositions about what a computer is.

Understanding the needs and abilities of students is fundamental to designing an efficient learning environment. Many groups of students find computer science and even programming obscure and unattractive, and they cannot see any connection to the real life. It has been shown /5/ that women and other underrepresented groups are more likely to engage in discipline if they understand its connection to society. First steps for the student should therefore be concrete and simple, and they should produce (with as little effort as possible) quickly observable results the average student would appreciate. Embedded systems turned out to be too abstract for many beginners to be able to see their applicability in real life. We chose the area of computer engineering all students are familiar with from the user point of view, that is Internet and web based applications.

Apart from that, as we have learned, many students were so much frustrated by the failure at the first unsuccessful attempt to install an IDE that they gave up on programming altogether. Some of those that by some lucky coincidence have no problems installing an IDE were frustrated when, after first compiling a program, a single missing semicolon produced the list of error messages longer than a code itself. For lots of students that was enough to quit the further attempts altogether. To start out, we somehow wanted to avoid the need of compilers. Ideally, student should do with software already installed on his/her computer, e.g. a simple text editor and a browser.

According to those findings we decided to design a curriculum starting from the following prerequisites:

1. Computer architecture and assembly language courses should not be taught independently from the higher, general purpose languages.
2. Introductory course should start out with a higher language, supported with a lot of simple real-life problems that students will instantly recognise as familiar and therefore important.
3. Hardware architecture, programming model, assembly language, and lower software techniques such as bitwise logical operations or bit masking should be introduced gradually in later stages, again supported with solutions of realistic problems.

3. The curriculum - first year level

3.1. General purpose programming

The main objective of our curriculum is that after the first year students master basic algorithm development, programming and coding skills, learn the most basic elements of computer architecture, get some idea of system approach, and start to appreciate the role and importance of embedded systems in modern society. Apart from those subject-specific topics, we have made a significant effort to try and achieve some of the more general teaching goals. Above all we find it important that a student after a first year should to the certain extend be able to notice and solve problems, to think critically and to reflect, develop abstract and system-based thinking, to analyse and to draw syntheses, and gain some self-confidence and spirit of enterprise and activity.

Since our final goal after all is embedded system programming, we need to choose appropriate language to back that up. C language is definitely one of the most appropriate candidates, if nothing else for its widespread usage. C and its derivatives are by far the most widely used programming languages today. The most heard argument against C as a teaching language, that it is ugly - which by the way it is - can be avoided by following some simple rules like indenting code and strictly writing curly braces at every beginning end ending of a block, even when syntactically that is not required.

We have chosen 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 a 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. Even the most unenthusiastic students get some kick when they produce a first document which actually shows in a browser without much danger of something getting terribly wrong.

A mark-up 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 a beginning 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, to avoid the problem of unnecessary learning overhead when switching between languages and make 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 (in view of embedded C programming) yet practical framework for introducing basic concepts of computer programming.

Having once eliminated as much of problems annoying for beginners as possible, we can focus on real problems right away. Motivation too is not an issue anymore, since practically every student has experience using browser and is keen to producing his/her own pages. Students quickly want to write more and more complex web pages and soon realise why programming is so important. That way, we can emphasize principles of programming like program development and stepwise refinement more efficiently rather than just present and explain sample programs. Students become very motivated to further develop programs that have been devised to certain stage in lectures. For example, we can present and demonstrate students a simple program that moves a window, and then tell them they can shake a window by multiple moves in different directions in a for loop. They are bound to try that at home.

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

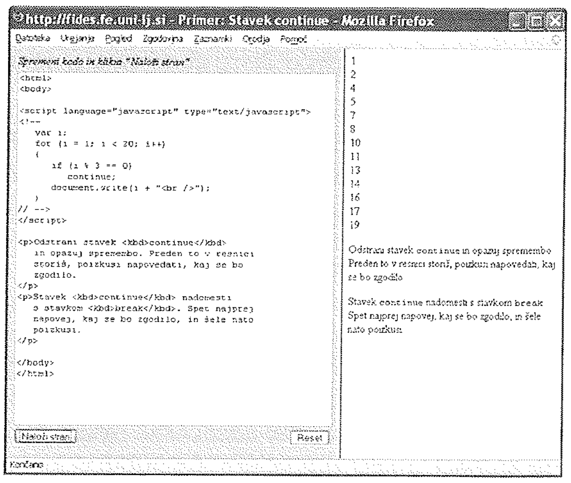
```
var i;
for (i = 0; i < num_thumbs; i++)
{
  document.write("<img src='");
  document.write(thumb[i]);
  document.write("' />");
}
```

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.

Another benefit of using JavaScript is an early need to think modularly and write appropriate functions. Experiences have shown that students tend do avoid decomposing problems in modules and writing functions. Instead they put all code in a main() function. There can be up to several hundreds lines of code and they still don't see why this is wrong.

With JavaScript things are different. If one wants to respond to an event he/she must write a function to do it. It is crystal clear to anyone that trying to put more than a single line of code in a HTML tag verges on a suicide.

In order to support the XHTML/JavaScript part of the course we have written an on-line textbook with embedded live examples. A student is encouraged to experiment with them without a need to copy them elsewhere or even type them. If something gets wrong, there is a Reset button, which restores the original example. One such example is shown in Fig 1. The example demonstrates the use of a statement continue. Apart from the JavaScript code there are some instructions for the student as how to experiment with the code. A student is always encouraged first to try and forecast the effect of the changes to the code and only then to reload the page to see the actual output of the program.


http://fides.fe.uni-lj.si - Primer: Stavek continue - Mozilla Firefox
Datoteka Urejanje Pogled Zgodovina Zaznamki Orodja Pomoč
Spremeni kodo in klikni "Naloži stran"
Naloži stran
Reset
Končano

```
<html>
<body>

<script language="javascript" type="text/javascript">
<!--
   var i;
   for (i = 1; i < 20; i++)
   {
      if (i % 3 == 0)
         continue;
      document.write(i + "<br />");
   }
// -->
</script>

<p>Odstrani stavek <kbd>continue</kbd>
   in opazuj spremembo. Preden to v resnici
   storiš, poizkusi napovedati, kaj se bo
   zgodilo.
</p>
<p>Stavek <kbd>continue</kbd> nadomesti
   s stavkom <kbd>break</kbd>. Spet najprej
   napovej, kaj se bo zgodilo, in šele nato
   poizkusi.
</p>

</body>
</html>
```

```
1
2
4
5
7
8
10
11
13
14
16
17
19
```

Odstrani stavek continue in opazuj spremembo. Preden to v resnici storiš, poizkusi napovedati, kaj se bo zgodilo.

Stavek continue nadomesti s stavkom break. Spet najprej napovej, kaj se bo zgodilo, in šele nato poizkusi.

Fig. 1: An embedded live example in the textbook.

3.2. Transition to embedded C

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 differences between the languages JavaScript and C. Those aren't many since we have "hidden" most of the unnecessary parts of JavaScript that are too different from C language.

For the purpose of teaching embedded C we use a special training hardware platform with very rudimentary input and output, and without an operating system. We describe the platform in detail in section *Hardware Platform*. The most important difference between C and JavaScript 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; once we decided that a variable is of type double, we cannot change it in e.g. string. This is somewhat a relief to a minor population of students that already have some programming experiences. These differences are not very difficult to grasp for the students, and we quickly move on.

Connecting simple external 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 controlling such devices is in fact trivial.

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

```
int i;
for (i = curpos; i < stop; 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.

Even more control over hardware is possible when we learn binary coding principles, bitwise logical operations, bit masking, and direct addressing of the hardware registers.

3.3. Early detection of multitasking and real-time problems.

A perceptive student will soon realise that the above example works good only as long as there are no other concurrent demands on the system apart from rotating a motor. Consider for example that the motor is driving a ramp. A closing ramp should stop immediately after a sensor has sensed an obstacle e.g. a car or person under it. It wouldn't be healthy if the system detected the obstacle only after the motor rotating has been completed.

Students learn that a computer can perform multiple tasks concurrently even if the microcontroller operation is serial in nature. The concurrency is achieved by simply distribute a processor time among different tasks that need to be done. The concept is not very difficult to understand, more interesting are the consequences. Without a proper guidance students quickly tangle into dead loops waiting for a key to be pressed, while a system has frozen.

In most cases concurrency alone is not enough to get the system working. Some tasks have to be executed in certain prescribed time span. Writing programs that meet certain performance deadline is quite different from ordinary, non-real-time programming that most students are used to. This concept again is easy to understand but to engineer a real time application requires a lot of system knowledge that is beyond the scope of a first-year student. So in first year we introduce somehow intuitive polling and assume that all partial tasks execute in a time span that is much smaller than the time available. To meet the timeliness of execution we constantly read the system clock and when the time is due, we simply execute what has to be executed. How really to engineer a real time program to meet performance demands, how the real time concepts affect the overall system performance, and how it complicates debugging is left for a later time, although students already get some idea that things are quite challenging and not so trivial.

4. Advanced Embedded Systems 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. The latter 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 /7/.

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 are able to purchase their own development boards at affordable price, 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.

5. 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 driving 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.

Above all we wanted 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.

5.1. 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 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 negotiation on both sides a strong consortium of six companies was ready to develop and finance our new ARM7 development board.

According to the demands identified in previous sections we designed the basic development module as depicted in Fig. 2.

The highlight is the integrated on-board debugging hardware linking the ARM7 CPU to the well known professional development environment winIDEA™ by iSystem /8/ 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 professional use of the system. This is an original concept protecting the copyright of winIDEA™ and giving the students full development power at the same time.

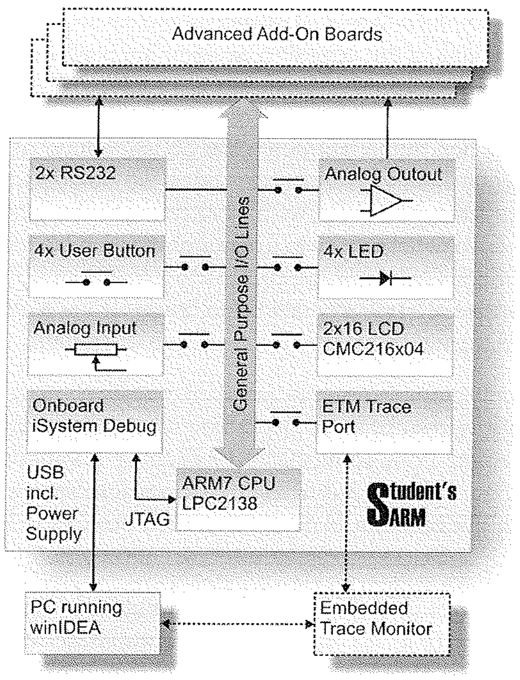


Fig. 2: ARM7 Development system overview.

As shown in Fig. 2, the development board 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 supported with libraries of (semi)-standard C functions enabling students a quick start. There are four keys, four small LEDs, a potentiometer connected to one of the A/D inputs, a general purpose operational amplifier connected to one of the D/A outputs, a pair of RS232 serial ports, and the facilities to mount a standard LCD piggyback. That is more than enough for a beginner course. Advanced level course on the other hand requires more specific devices.

In order to accommodate these specific needs we have provided respective connectors to 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 development board is manufactured in SMD technology, based on a four layer 10 by 10 cm PCB as seen in Fig. 3. 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.

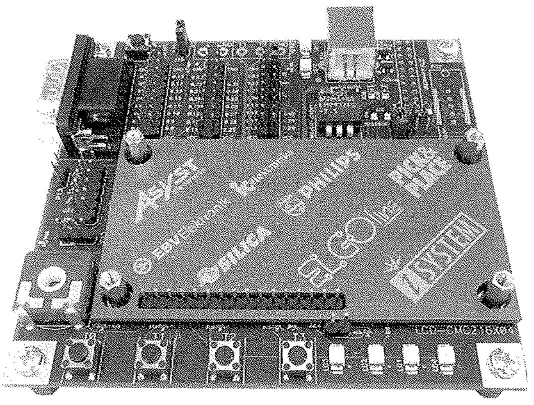


Fig. 3: Student's ARM7 Development board.

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 /8/, demonstrating the capabilities of winIDEA^{TM}.

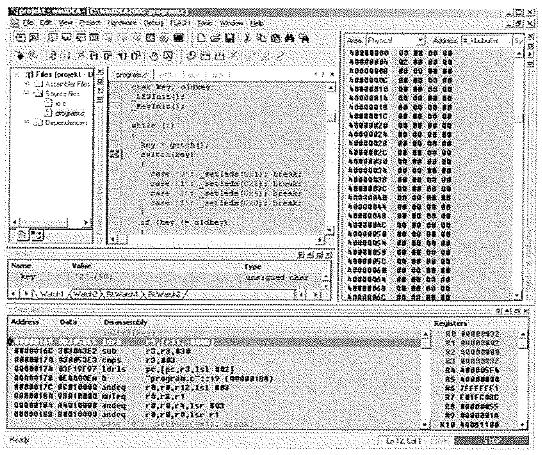
5.2. Integrated Development Environment

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

Fig. 4 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.

6. Observations

It has been almost three years since we introduced the approach described in the paper. Nevertheless it is extremely difficult to give any objective measure of the approach efficiency. We are aware that many parameters change each year upon which we have little or no control and are very difficult to measure. These are the quality of the students enrolled in the program, dates of examinations of other subjects, changes of teaching stuff, to mention just a few. Some may argue that any conclusions made can be of a post-hoc-ergo-propter-hoc type. There are however many indicators, mostly empirical, that make us believe that the approach presented has been successful in achieving the intended goal.



```
char key, oldkey;
_LEDInit();
_KeyInit();

while (1)
{
  key = getch();
  switch(key)
  {
    case '0': _setleds(0x1); break;
    case '1': _setleds(0x2); break;
    case '2': _setleds(0x4); break;
    case '3': _setleds(0x8); break;
  }
  if (key != oldkey)
```

```
Name    Value       Type
key     '2' (50)    unsigned char
```

```
Address   Data     Disassembly
0000016C  E2433030 sub     r3,r3,#30
00000170  E3530003 cmps    r3,#03
00000174  979FF103 ldrls   pc,[pc,r3,lsl #02]
00000178  EA00000A b       "program.c"::19 (000001A8)
0000017C  00010000 andeq   r0,r1,r12,lsl #00
00000180  00010000 mulleq  r0,r0,r1
00000184  00010000 andeq   r0,r0,r4,lsr #03
00000188  00010000 andeq   r0,r0,r0,lsr r1
```

```
Registers
R0  00000032
R1  00000002
R2  40000000
R3  00000032
R4  400005F4
R5  40000000
R6  7FFFFFF1
R7  E01FC000
R8  00000055
R9  0000001A
R10 40001100
```

Fig. 4: The winIDEA^{TM} integrated development environment.

The first thing we notice is drastic increase in students' interest in the subject already during the lectures. 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, and further through e-mail and after-hours. An increased number of students having problems with software and hardware is a sure indicator, that they actually try things out at home. Last month a group of students approached us with a wish that we organize additional summer classes covering the subject more in depth.

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. Fig. 5 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 seven years. When we introduced the new concept, a significant raise in success rate was observed (year 2005). The percentage is calculated against the population that took the exam, and not the total student population.

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

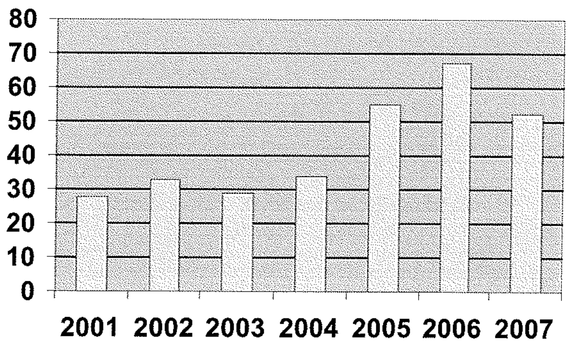


Fig. 5: Percentage of students passing the exam during the first examination period.

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.

7. Conclusion

In the paper we described a recent redesign of embedded systems curriculum at the University of Ljubljana. The old curriculum became dangerously outdated due to enormous technical advances accompanied by not neglect able changes in social and cultural environment through the last decade or two.

We carried out the reorganisation on a three point action plan. Firstly, we strove for a unified software/hardware platform, which would serve as much microcontroller and programming courses as possible. Secondly, we wanted each student to posses his/her own microcontroller development board right from the first year in order to get more involved and to be able to work more efficiently. Thirdly, we had to attract 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, which closes the loop. 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 three years 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.

8. 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.

9. References

/1/ W. Wolf, J. Madsen, *Embedded Systems Education for the future*, Proceedings of the IEEE vol. 88, no. 1, January 2000, pp. 23-30.

/2/ R. J. Wang, *From elitism to mass higher education in Taiwan: The problems faced*, Higher Education vol. 46, no. 3, 2003, pp. 261-287.

/3/ John Sharpham, *Managing the transition to mass higher education in Australia* Long Range Planning, vol. 26, no. 2, April 1993, pp. 51-58.

/4/ Martin A. Trow, From Mass Higher Education to Universal Access: The American Advantage (March 1, 2000). Center for Studies in Higher Education. Paper CSHE1-00. http://repositories.cdlib.org/cshe/CSHE1-00

/5/ OECD (2006): Education at a Glance OECD Indicators.

/6/ P. D. Stephenson, J. Peckham, Seeing is Believing: Using Computer Graphics to Enthuse Students, IEEE *Computer Graphics and Applications*, vol. 26, *no. 6*, Nov.-Dec. 2006, pp. 87-91.

/7/ *ēSmartX, the free real time operating system for the ARM7TDMI platform*, usmartx.sourceforge.net, 2006.

/8/ *iSystem AG* www.isystem.com, 2008.

/9/ T. Tuma, I. Fajfar, *A new curriculum for teaching embedded systems at the University of Ljubljana*, 7th International Conference on Information Technology Based Higher Education and Training : July, 2006, Sydney, Australia.

prof.dr. Iztok Fajfar, univ.dipl.ing.el.
Fakulteta za elektrotehniko Univerze v Ljubljani
Tržaška cesta 25, 1000 Ljubljana
tel.: (01) 4768 722, fax: (01) 4264 630
e-pošta: iztok.fajfar@fe.uni-lj.si

prof.dr. Tadej Tuma, univ.dipl.ing.el.
Fakulteta za elektrotehniko Univerze v Ljubljani
Tržaška cesta 25, 1000 Ljubljana
tel.: (01) 4768 329, fax: (01) 4264 630
e-pošta: tadej.tuma@fe.uni-lj.si

doc.dr. Arpad Burmen, univ.dipl.ing.el.
Fakulteta za elektrotehniko Univerze v Ljubljani
Tržaška cesta 25, 1000 Ljubljana
tel.: (01) 4768 322, fax: (01) 4264 630
e-pošta: arpad.buermen@fe.uni-lj.si

doc.dr. Janez Puhan, univ.dipl.ing.el.
Fakulteta za elektrotehniko Univerze v Ljubljani
Tržaška cesta 25, 1000 Ljubljana
tel.: (01) 4768 322, fax: (01) 4264 630
e-pošta: janez.puhan@fe.uni-lj.si

Prispelo (Arrived): 23.07.2008 Sprejeto (Accepted): 19.03.2009