The Delft MS Curriculum on Embedded Systems

Hans-Gerhard Gross
Delft University of Technology
Mekelweg 4, 2628 CD Delft, NL
+31 15 27 87750
H.G.Gross@tudelft.nl

Arjan van Gemund
Delft University of Technology
Mekelweg 4, 2628 CD Delft, NL
+31 15 27 87750
A.J.C.vanGemund@tudelft.nl

ABSTRACT
Embedded Systems (ES) is the fastest growing sector in ICT technology [1, 15]. Also in the Dutch economy, this sector is believed to become an important generator of added value. In this paper we describe a new MS program on ES that will be offered in Delft as of the academic year 2006. The MS program is a joint offering by the three Universities of Technology within The Netherlands (at Delft, Eindhoven, and Twente). We describe the program, its rationale, and two examples of already existing courses (Embedded Systems and Real-Time Systems), from which the new ES curriculum has emerged.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education; *Curriculum*

Keywords
Embedded systems, Curriculum

1. INTRODUCTION
Amongst the 13 universities present in The Netherlands, three of them have a distinct focus on scientific engineering, namely (i) Delft University of Technology [5], (ii) Eindhoven University of Technology [7], and (iii) University of Twente [23]. For many years each of these three universities offer BS and MS programs in Computer Science (CS) and Electrical Engineering (EE), and in the case of Delft, also an MS program Computer Engineering (CE).

In sync with the international trend, within The Netherlands there is a growing understanding that ES can no longer be designed in terms of two separate threads of hardware and software that are merged at a later stage [1]. A systems approach is required that mixes functional and non-functional requirements from the start. Central to this approach is the need to understand the interaction of the system with its physical and network environments. These changes require engineering teams that possess skills in a wide range of disciplines such as computer science, electrical engineering, real-time computing, systems architecture, control engineering, signal processing, security and privacy, computer networking, mathematics, hardware, sensors and actuators. Engineering teams are currently unable to effectively consider fundamental design issues from all these perspectives simultaneously, because they lack the common background and technical language to interact efficiently. Creating these multidisciplinary skills requires fundamental changes in engineering education, and, since a number of years, many courses and curricula on ES have emerged [6, 16, 17, 20].

Motivated by (1) the need for ES engineers at MS level, that master, or at least are comfortable with the above range of multidisciplinary engineering subjects, and (2) given the ever increasing disparity between the current, discrete-domain driven CS curricula and the predominantly continuous-domain driven EE curricula, the Dutch Ministry of Education has approved the implementation of a new MS program on ES, slated for the academic year 2006. Note, that it is not the intention (nor possible) to 'retrain' CS bachelors to EE masters, nor EE bachelors to CS masters. An important motivation of an MS program on ES is that the bachelor becomes comfortable with the complementary domain. For instance, a graduated ES master with a BS CS will much better understand the language and tools of EE engineers, and has better feeling with the EE problems, and vice versa. In this way, they can more effectively work together on ES.

Unlike traditional programs, for reasons of efficiency[1], the Ministry has stipulated that in order to receive accreditation, the ES program be offered jointly by the three Dutch technical universities, which have recently been federated under the joint brand name 3TU (TU = technical university). While this joint offering implies that all Dutch students learn a national MS degree ES (issued by 3TU), the current implementation still allows some room for a local Delft, Eindhoven, and Twente differentiation.

In this paper we describe the MS program ES as offered by Delft University of Technology. Although some 80 percent of the two-year program2 is identical for all three universities, the Delft program has a particular focus on (1) embedded software, (2) bridging the gap between the discrete and continuous domain, and (3) system-level engineering, rather than component-level engineering which is typical for the Dutch EE/CE programs. Apart from describing the curriculum and its rationale, we describe two courses IN4073 and IN4024 (Embedded Systems and Real-Time Systems, respectively) that have been offered for a couple of years as electives within the CS master program, and which are now part of the mandatory core of the new ES program. In particular, the IN4073 offering can be seen as typical for the Delft approach to ES education. The paper is organized as follows. In Section 2, we present the new ES curriculum as offered by Delft, and motivate the specific focus on embedded software and multi-

[1] No more than some 100 students are expected to register nationwide.

[2] Dutch MS programs are typically two-year programs, comprising 120 EC. 1 EC (European Credit, defined through the European Credit Transfer System, ECTS) stands for 28 hours of nominal study load.

disciplinarity. Section 3 describes the course offering IN4073 Embedded

Systems, while Section 4 describes the Real-Time Systems course. In Section 5, we provide some additional details on the general didactic context in which our education is currently performed. Section 6 concludes the paper.

2. DELFT ES PROGRAM

The Delft version of the 3TU masters program ES [12] is offered by the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS [9]), which covers the entire embedded systems engineering spectrum from embedded software engineering to sub-micron engineering. The overall, high-level learning goals of the program aim to bring students into the position to develop and apply new research ideas in a multidisciplinary working context, integrate their knowledge to solve complex problems, and make judgments based on limited or incomplete information, reflect on the socio-ethical impact of their judgments, and communicate their decisions, solutions, and reflections to other people even outside of their field. The ES program has the following overall structure:

- Mandatory courses (40 EC). This is the common core of the ES program. Up to 10 EC of the mandatory part consists of so-called homologation courses to equalize the differences in previous educational background.
- Elective courses (40 EC). The electives span a broad range from control theory to sub-micron Si realization. Up to 20 EC of the elective program can be taken in terms of a traineeship, preferably carried out with an international company or research institute.
- Thesis project (40 EC). The thesis project includes an introductory (10 EC) individual project in preparation. This individual project is tailor-made and may contain such elements as literature surveys related to the final project's subject, preparatory research studies, or additional specialist courses, whatever is needed to make the student well-prepared for the project.

Whereas the scope of the electives and thesis project is equal for all three universities, the differentiation between the universities lies in the mandatory part. Consequently, in the following, we will focus on the mandatory part of the curriculum.

2.1 Delft Focus

With regard to the mandatory part of the program, Delft has a particular inclination towards embedded software, dependable systems, and multidisciplinarity.

2.1.1 Embedded Software

As an increasing amount of functionality shifts from hardware to software, embedded software engineering cost is becoming the bottleneck. A contributing factor is the ever decreasing cost of (field) programmable hardware (microcontrollers, FPGAs), which often outweigh the advantages of devising application-specific silicon solutions. This applies in particular to the Dutch context[3] where, on a whole, more valorization is likely to occur in the software systems (specification, integration) domain, than on the processor domain[4].

Another important aspect of the above Delft orientation towards systems and software with respect to the ES program, is the fact that Delft, unlike Twente and Eindhoven, already has a MS program CE in place for a number of years. Although the Delft CE program does include compilers, system software, and operating systems, the main aim of the program is to produce experts in computer architecture and the development and implementation of computer hardware (for example, more than 25 percent of its mandatory core is entirely devoted to computer arithmetic). Although there is overlap between the CE and ES programs with respect to embedded systems, the Delft CE program effectively focuses on hardware issues[5]. In a sense the Delft ES program seeks to complement the existing CE program, by focusing on embedded software, multidisciplinarity, and system-level engineering, rather than embedded hardware, and component-level engineering.

2.1.2 Dependable Systems

Furthermore, the increasing level of connectivity that comes with "ambient intelligence" is leading to increased availability of data and information, anywhere and at any time. This offers a huge potential, but also presents tough challenges in terms of interoperability, efficiency, complexity and vulnerability. Embedded systems must also be safe and reliable: it is important that increased complexity does not compromise their dependability. In line with this emphasis on embedded software and dependability, Delft has established a new (part-time) chair Embedded Software that performs research and education on embedded software and dependability [8].

2.1.3 Homologation

Similar to Twente, a specific feature of the Delft approach is that a part of the mandatory program is devoted to what is called homologation. Homologation is a an equalization process aimed at achieving homogeneity with respect to the student's starting level, in view of the fact that the MS program is typically entered by CS and EE bachelors which have quite a different background. For example, Dutch students with an EE BS will lack knowledge and experience in, e.g., programming, operating systems, and software engineering, whereas Dutch students with a CS background will

[3] In a hardware sense The Netherlands is essentially an import economy.

[4] In this view, hardware synthesis based on HDLs is considered a software approach. Actually, in the CS bachelor program, a sophomore course on HDL programming is envisaged for the academic year 2007.

[5] It is worth mentioning that the Dutch situation with respect to CE slightly differs from that in the US. The CE Body of Knowledge (BOK) in the US [14] has naturally included embedded systems (denoted ES-ESY) from the outset, even though it has recently been acknowledged that an ES BOK should somewhat extend this CE BOK [21]. With only one (Delft) CE program in The Netherlands, which, compared to the CE BOK, is more hardware-oriented, there is even more point to start a new ES program than in the US.

generally lack knowledge and experience with, e.g., logic design, control theory, and signal processing. As indicated earlier, the purpose of the ES program is to essentially integrate the CS and EE disciplines, rather than just providing an ES flavor to CS and EE bachelors, who would then maintain their respective path through the electives and final project without meaningful cross-fertilization. Hence, the first 10 EC of the master program, called the homologation phase, provide specific subjects to remedy the differences in background. Depending on a student's background, some of these subjects will be mandatory. Homologation is seen as a crucial instrument to acquire a common background to achieve early integration between the various disciplines. It provides the students with the broadening that is required to effectively absorb the multidisciplinary program that lies ahead.

2.2 Mandatory Courses

As mentioned earlier, the 40 EC mandatory part includes a 10 EC homologation phase and a 30 EC common core. For a student with a Delft CS bachelor degree, the mandatory homologation subjects are Systems and Signals, Control Theory, Logic Design, and Digital Signal Processing. For a student with a Delft EE bachelor degree, the mandatory subjects are Operating Systems, Systems Programming (using C), and Software Engineering (using Java). While the homologation phase establishes a common background, the following 30 EC offerings essentially constitutes the common core of the curriculum:

- IN4087 System Validation (formal methods, model checking)
- IN4088 Software Testing and Quality Engineering (testing large embedded software systems)
- IN4024 Real-Time Systems (detailed in Section 4)
- ET4282 Performance Analysis (performance modeling of computation and communication)
- IN4073 Embedded Systems (detailed in Section 3)
- ET4165 Embedded Computer Architecture (contemporary embedded processors, microcontrollers)

Compared to the two sister universities in Eindhoven and Twente, the above program has a distinct focus on systems, software, and dependability. It should be kept in mind, however, that next to the 40 EC thesis work, the remaining 40 EC electives offer a very broad scope, ranging from control engineering to sub-micron Si realization, parts of which can be taken from any of the three universities.

Describing all elective courses (more than 50) would go well beyond the scope of this paper. We can therefore only list the range of courses offered. Electives comprise courses on system modeling, signal processing, software engineering, artificial intelligence, high performance computing, robotics, and microprocessor design. The full list of courses can be obtained from the study information system of TU Delft [18]. Elective courses are chosen by students based on consultation with the MS ES coordinator of the faculty and the supervising professor of the thesis project.

3. COURSE: EMBEDDED SYSTEMS

The course IN4073 Embedded Systems [10], now part of the mandatory ES curriculum, originates from an elective offering as of 2004 within the Delft CS master's curriculum (and is still being offered at CS). Based on the new guidelines for quality teaching at Delft described in more detail in Section 5, the course is primarily aimed at providing the students with hands-on experience with designing embedded systems that perform non-trivial tasks. The main ingredient of the 6 EC course is a lab project carried out by 12 competing teams of around 5 students per team, where each team must design an embedded system to control a model helicopter. Due to financial restrictions, each team has (supervised) access to the lab for only 7 slots of 4 hours per slot. This implies that most of the work must be prepared by the teams outside lab hours, while the actual testing is to be done during the 7 slots.

The course has been taken by some 55 MS students each year, of which some 50 pass the course. The students' background is mostly CS (60 percent) and CE (30 percent). Future editions will, of course, include the new ES students. As the lab work is essentially multidisciplinary, each team is composed of a mix of CS and CE students, which usually presents a first-time opportunity to get acquainted with different engineering backgrounds. Apart from the lab work, the course includes a number of supporting lectures, as to provide the necessary background knowledge to successfully perform the lab work. Given the large focus on lab work, a support web site is provided [11] that contains various resources, rather than a text book. In any case, embedded systems text books are usually very much oriented towards a particular hardware architecture, instruction set, and/or programming model. As indicated earlier, our approach towards embedded hardware synthesis is through software. Consequently, in the design lab, hardware is programmed in terms of VHDL and C virtual machines only, where hardware particulars are reduced to a few trivialities such as memory mapping registers of the (VHDL) devices in the C programming model, and setting interrupt routine function pointers. The goal of the 6 EC credits course is not to have the student master all multidisciplinary skills of embedded systems engineering, but rather to have the student understand the basic principles and problems, develop a systems view, and to become reasonably comfortable with the various disciplines involved in embedded systems design.

3.1 Lab Project

In order to describe the body of knowledge covered by the course, we will briefly detail the lab project. The project chosen for this course is to design embedded software to control and stabilize an electrical model helicopter, such that it can be flown by inexperienced users using a single joystick. This application has been chosen for a number of reasons:

- The application is typical for many embedded systems, i.e., it integrates aspects from many different disciplines (mechanics, control theory, sensor and actuator physics/electronics, signal processing, and last but not least, computing hardware and software).
- The application is contemporary. Today's low-cost RC model-helicopters are only rudimentary controlled (simple yaw control), which implies that only skilled hobby pilots are capable of flying these machines (i.e., simultaneously controlling motor speed, pitch, roll, and

yaw[6]) without crashing within a few seconds after lift-off. Although perceived as the true sporting challenge, many recreational users (the authors included) would prefer an aerial vehicle that is much easier to handle.

- The application is typical for many air, land, and naval vehicles that require extensive embedded control software to achieve stability where humans are no longer able to perform this complicated, real-time task.
- Last, but certainly not least, helicopters are great fun.

Although designing embedded software that would enable a heli to, e.g., autonomously hover at a given location, or move to a specified location (a so-called autopilot) would by far exceed the scope of a 6 EC course, the course project is indeed inspired by this very ambition. In fact, the project is presented to the students as a prototype study aimed to ultimately design such a control system, where, e.g., a pilot would remotely control the helicopter through a single joystick, up to the point where the helicopter is entirely flown by software.

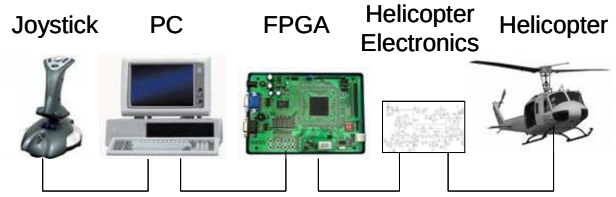

Fig. 1: Hardware Setup

3.1.1 Experimental Setup

While the ultimate embedded system would ideally be implemented in terms of one chip (SoC) which would easily fit within the electrical model helicopter's limited payload budget, the student's prototype embedded system comprises an external FPGA board, that is connected to a sensor/actuator electronics board that interfaces the FPGA board with the helicopter's actuators (rotors, servos) and sensors (gyros and accelerometers). The system (which is perceived as part of the helicopter) receives its commands from the so-called ground station, which consists of a Linux PC and a joystick. In our prototype approach, we refrain from implementing radio communication between the helicopter and the ground station. Rather, we conduct tethered flight, where the helicopter is connected to the ground system through wires. This, incidentally, also allows for prolonged testing, since low-cost helicopter batteries are usually depleted within 20 minutes. The system setup is shown in Figure 1. As shown, an additional, analog electronics board is required to interface the heli rotors, servos, and sensors to the FPGA board. The (low-cost) board, specifically designed for the used sensors, is realized by the authors, and features power FETs for PWM motor control, and DC-to-PWM converters for the 3 gyros. Although, the prototype embedded system therefore comprises two boards (FPGA +heli interface), it is perfectly possible to map all electronics within one chip onboard the helicopter[7].

3.1.2 Resources

Rather than resorting to expensive helicopters, sensors (e.g., entire IMUs), electronics, FPGAs, etc., we explicitly choose a low-level, low-cost approach, which opens up a real possibility for students inspired by this course to continue working with helicopters and/or ES on a relatively low budget. The helicopter[8] is a low-cost (O(100) Euros) version that essentially comprises a fuselage with only the main + tail motors, and the pitch + roll servos (i.e., 4 actuators). The gyro and accelerometer sensors required to derive the attitude (which is the minimal information for a simple autopilot application) are also low-cost (O(50) Euros) per sensor; at least 5 are required for proper attitude computation.

The ES hardware approach is also low-cost. Rather than using some high-performance microcontroller (and costly development tool suite) we use a low-cost FPGA board (O(100) Euros) featuring a 400K gates Xilinx Spartan-3 (XC3S400), that comes with free development tools. In order to allow less time-critical parts of the ES to be conveniently developed in C rather than VHDL, an experimental 32 bit soft core (VHDL processor component) is provided, that interfaces with high-speed VHDL components through shared memory. The soft core, called X32, has been developed at our Embedded Software Lab [24], and comprises the VHDL core and associated C programming tool chain (ANSI compiler based on lcc, assembler, linker, simulator, up-loader, debugger). A Delft development, the X32, is license-free, allowing the entire software to be used by students at home without cost.

The PC is a standard Linux PC, that acts as development and upload platform for the FPGA (VHDL) and the X32 soft core (C). The PC is also used as run-time user interface, in which capacity it reads joystick and keyboard commands, transmits setpoint commands to the embedded system, logs telemetry data from the embedded system, while visualizing and/or storing the data on file for off-line inspection. The PC has a parallel port (LP), used to upload FPGA designs to the FPGA board, and a 115,200 baud serial link (COM) to communicate to the FPGA board at run-time.

Of course, the above low-cost approach is not without consequences. A very small helicopter implies that it is highly unstable and therefore extremely difficult to control in comparison to real helicopters. Moreover, instead of controlling thrust using collective rotor pitch, thrust is currently controlled by motor speed, which also adds to the stability problem. Using low-cost sensors introduces larger measurement errors (drift) which degrade computed attitude accuracy and control performance. Using a low-cost FPGA (compared to a high-end FPGA with multiple hard processor cores) severely reduces the size and performance of the designs. Although the X32 occupies less than 50% FPGA space, this leaves limited space for the additional VHDL devices (timers, PWM converters, UART) that make up the total microcontroller architecture. Moreover, the FPGA designs typically run at 50 MHz, which implies limited soft core processing performance,

[6] pitch, roll, and yaw are the three angles that constitute the helicopter's attitude in 3D.

[7] In a future version, the interface electronics will be integrated within the heli.

[8] Currently a Piccolo V2, to be succeeded by a more sturdy TREX 450 as of next year.

compared to a high-end hard core. Nevertheless, practice has shown that the low-cost setup is more than capable to perform adequate helicopter stabilization, and provides ample opportunity for students to get fully acquainted with embedded systems programming in a real and challenging application context that includes resource limitations.

3.2 Design Challenges

The FPGA - PC design involves components such as signal filters, yaw, roll, and pitch controllers, serial-parallel communication transceivers, pulse-width modulators/demodulators (motors, servos), joystick/keyboard handling, and a UI (OpenGL), including a main FSM controlling the various mode of heli operation (safe mode, calibration mode, yaw stabilizer-only, full roll/pitch/yaw stabilization). The design involves a number of challenges, including:

- concurrent real-time programming and debugging at both FPGA and PC (e.g., scheduling/interaction of FPGA - PC communication tasks, the PID controllers, and signal filters),
- hardware/software co-design (determining which components must be implemented in VHDL or C in view of performance and space limitations),
- coping without floating point support: as many low-cost microcontrollers, the X32 has no floating point support. Hence, controlling and filtering must be done using fixed-point arithmetic, which introduces stability issues,
- teamwork: producing an overall system design, and partitioning the overall design in components, allowing parallelization of student effort, considering the various differences in educational, and cultural background,
- understanding the various disciplines (mechanics, electronics, control theory, software engineering), and understanding and coping with the various interfaces (sensors, motors, joystick, graphics, X32 memory and programming model).

The supporting lectures include lectures on helicopter mechanics, modeling, and simulation, control theory, digital filter theory, VHDL programming, X32 architecture and programming, and basic electronics.

3.3 Evaluation

Currently, the course has seen two editions (academic year 2005 and 2006). As to be expected with a new course, a number of issues surfaced. In the 2005 edition, the moderate goal of the lab was to design helicopter yaw control only (i.e., no roll and pitch control). Even then, students found themselves struggling with many side issues, mostly relating to insufficient prior knowledge of VHDL. Consequently, PC—FPGA communication, which involved designing UARTs at the FPGA side, as well as designing PWM generators to control the heli actuators, and the PW readers to read back sensor data (also in PWM format), generally took more than half of the lab time. As a result, only 50 percent of the teams was able to deliver a demonstrator that performed helicopter yaw control.

In the 2006 edition more functionality was added to the soft core, such as a UART, which somewhat took the load of the VHDL part, allowing more focus on the actual control problem (which is programmed in C on the X32). As this year's goal was to also deliver pitch and roll control (next to the easier yaw control), signal filtering became a more demanding issue. At this point, another deficiency became apparent, namely the fact that most students had no prior hands-on experience with designing fixed-point IIR filters, even though, at least half of the students had prior exposure to a DSP course. Although the course provides a supporting DSP lecture (a 4-hour crash course), immediately applying the fresh knowledge proved to be time-consuming. While, on average, the 2006 results were much better than 2005, only some 25 percent of the teams were able to demonstrate 3D attitude control.

Despite the fact that most teams didn't fully comply with the project goals, student satisfaction and retention was very high. In both editions of the course, less than 10 percent failed the course. Students consistently indicated a steep learning curve, especially with regard to the subjects outside of their curricular domain. Due to the low-cost setup, some 10 percent of the students actually purchased their own (FPGA) hardware to continue on private ES projects. The time spent on the course typically exceeded the nominal 168 study hours that stands for a 6 EC course.

One major complaint was the lack of lab time and resources. Although each team had access to an FPGA board outside lab hours (and access to two FPGA boards during lab hours plus a PC for each team member), lab time was usually devoted to testing and debugging (parts of) designs, which, of course, took much longer than anticipated. A complication was the fact that there is only one model helicopter available, leading to restrictions on helicopter test time. Although, students understood that this adds to engineering reality, we are inclined to investigate the (financial) possibility of increasing the level of lab support.

4. COURSE: REAL-TIME SYSTEMS

The course IN4024 Real-Time Systems [13] is also 6 EC. The course originates from the CS curriculum in which it is still taught as an elective module. It is obligatory in the new ES curriculum. Both courses, IN4073 and IN4024, are complementary. The real-time system course is intended to equip students with the basic concepts of real-time systems in the context of a standard PC, whereas the embedded system course goes beyond the boundaries of the PC, connecting it to an outside physical world, and thus adding another dimension. This is why, in the schedule of the curriculum, the real-time system course is placed before the embedded systems course.

IN4024, similar to IN4073, is split into 14 two-hour lectures and 7 four-hour lab sessions, with a strong emphasis on hands-on real-time system development experience. The lab sessions are the driving factor of the course, and this is in line with the new didactic requirements of the university (see Section 5). Assessment is carried out based on an 8 page research paper that the students have to submit at the end of the module. This focus on research is due to the fact that an MS degree requires scientific skills, in contrast to a BS which concentrates more on predefined implementation of tasks. Additional advantages of such an assessment are the students' improved research and writing skills when they enter their final year MS projects. The lectures are intended to equip them with the right terminology, and in the labs, they gain experience with doing their own research on a selection of subjects in

the real-time system domain. Typical number of students at registration is 100, going down to 60—70 in the first two lectures, dropping to some 35 in the last lecture with just about 40 paper submissions at the end of the module, all of which pass the course. The quality of the submitted papers is usually high[9].

4.1 Laboratories

The equipment of the labs comprises standard Linux PCs with the RTAI real-time extension [19] installed. We do not permit the students to use the full functionality of the real-time modules provided, which, in fact, would require the students to have administrative permission for loading and unloading their own real-time modules. Instead, the system provides the LXRT module which implements a user-level interface to RTAI's real-time services. This requires only normal user privileges with minor overall performance losses. RTAI is a freely available, easy to use real-time environment that most students install on their own PCs in order to be able to spend more time on the subject beyond the scope of the lab sessions, and perform more thorough experiments. We find this very positive. The real-time system lab is not one single larger project, but rather based on a number of different assignments that the students have to solve. They are supposed to implement parts of the assignments in C, do some experiments, e.g., timing measurements and devising schedules, reflect on the problems encountered, and the solutions found during the assignments. The assignments are designed in the form of scenarios that would be typical in a normal working environment in industry, such as "we need the worst-case execution time for this algorithm", "choose the best implementation of existing algorithms for the constraints of this system", or "which scheduling strategy would be optimal under such circumstances", etc. The assignments represent the framework in which the students carry out research. The following types of assignments are available:

- Comparison of standard Linux timing operations and the RTAI timing operations, accuracy of time measurements,
- Development of a high-resolution timer based on the processor clock and comparison of that timer with the standard Linux and RTAI timers,
- Development of a code framework under RTAI for worst-case execution time (WCET) analysis experiments and scheduling experiments
- Dynamic worst-case execution time analysis for standard algorithms, e.g., sorting, searching, computer graphics, etc., optimization of algorithms toward higher analyzability,
- Evaluation of dynamic timing analysis through code coverage analysis,
- Design of constant execution time algorithms (WCET-oriented programming),
- Search-based execution time analysis; comparison between manual testing, random testing, and application of a genetic algorithm as test case generator,
- Development of an instruction tracer for combined static and dynamic timing analysis.

The assignments are solved in groups of 3-4 students, and they provide a broad enough basis for many research topics. Each student chooses from the subjects of the module a research topic for the research paper, according to own preferences. Example submissions are:

- Priority-driven Algorithms for Safety-Critical Systems.
- Timing Analysis for Real-Time Systems.
- Using Advanced Genetic Algorithms to Improve Dynamic Testing.
- Implementation of a Constant Execution Time Sorting Algorithm.

The students are very eager to come up with original ideas, which makes reading the papers a pleasure.

4.2 Lectures

The lectures are intended to provide all the background knowledge, terminology and concepts for carrying out the assignments and writing the research paper. There is traditional style face-to-face presentation, but a large amount of lecturing time is devoted to discussions, questions, summaries, ad-hoc exercises and small assessments, so-called mini-tests. The primary topics covered in the lectures are about

- How to perform system domain analysis and derive high-level system timing requirements,
- How to decompose the system timing requirements and distribute them over the individual components,
- How to realize and implement software components with timing requirements,
- How to perform static and dynamic timing analysis in order to derive an execution schedule,
- How to implement a schedule with the means of the platform used,
- How to deploy a system or parts thereof on the platform used,
- How to test components and a system with timing requirements,
- How to evaluate the afore-mentioned steps by taking a development process view, and
- How to communicate this evaluation to others in terms of paper writing (scientific writing).

The students are supposed to discuss most of the topics among themselves and provide definitions and explanations. Their opinions are collected to form a general picture of the terms and concepts (supported through the lecturer). Some of the concepts are suitable for exercises, e.g., worst-case path analysis for static

[9] It seems that either students make the effort and perform well, or they drop out.

timing analysis, so in each lecture there is either an exercise of 20-30 minutes or a mini-test. In mini-tests, which is 5 questions to be answered on paper, students can assess for themselves whether they have understood the most important topics of previous lectures, or whether they have grasped the essentials of an article that they were supposed to read as homework. Each lecture closes with a summary provided by the students.

4.3 Evaluation

The real-time systems course is now also in its third edition, and it has been improved considerably compared to the first time, in particular, with respect to the diversity of the lab assignments and the amount of student/student interaction in the lectures. The difficulty of the lab assignments has been increased considerably, but also more support is being provided. Initially, students had to do a lot more own programming work which meant that there was not enough time for trying out different solutions and performing more measurements. Although now the assignments are more complex and more difficult, the students can use more existing code that they can incorporate into their own developments which gives them much more leeway for experimenting.

The lectures are now much more interactive, and much time is spent on exercises and discussions. According to course evaluation sheets, the students appreciate the high level of interaction and the fact that they are asked to "perform their own lectures". This, of course, costs a lot of time, so that coverage of the subject had to be sacrificed for in-depth analysis of individual topics. In our opinion, this is not necessarily a disadvantage, since the students are now a lot more confident on the fewer topics covered.

Demanding a research paper as the single assessment of the module is a bit ambivalent for two reasons. First, students like to be awarded for whatever they have done, so completing assignments and not getting course credits for them seems odd. Their assignments could be collected and marked, but it would impose an extremely high working load upon the lecturer. Not marking them is simply a matter of budget. Second, in particular the first participants feared that their provided papers would not be up to standards, and they would fail the course because of their low expected paper quality (another reason for having the assignments marked). However, the submitted papers were very good; only 2 out of 45 submissions had to be resubmitted in the first year. All subsequent students have now access to the best papers submitted (4-5 papers every year) which gives them a much better idea of what is required to pass the module, so that in the second round all 42 submissions were of high quality. Despite the fact that many are struggling, almost all students like the idea of submitting a research paper. Most of them have never written a scientific report or a paper, and they appreciate this opportunity to exercise for their coming research assignment and master thesis project. Every paper is going through a peer review by the students themselves, based on quality criteria provided, they can improve their work, and then make the final submission[10]. A final review by the lecturer provides them with concrete comments for improvement. Such final assessment is feasible only because of the low number of submissions (around 40). For larger courses, this would definitely not work.

[10] This is the version to be marked.

5. DIDACTICS

New rules in the didactics and tuition development in Delft require from academic staff to participate in a training program called the "Basiskwalifikatie Onderwijs" (BKO – basic qualification tuition), a quality scheme carried out or set up in most universities in The Netherlands. It has been implemented in order to impose a measurable base-line for teaching activities throughout Delft University, but also among the other universities. The main modules of the BKO program communicate skills for activating students to engage in learning processes, course design for interactive learning, project oriented and problem-based learning, and assessment, but also social skills such as teaching in English, and inter-cultural communication. In addition, the program offers further education in developing digital learning environments, and self-assessment and personal development.

The University's didactic vision, and, hence, the focus of the BKO is on communicating and training skills for active and collaborative learning (ACL) [3, 4]. ACL promotes instructional techniques that help students engage in higher-order thinking and problem-solving activities during class or seminar, and collaborative and communicative activities involving other fellow students. Traditional university teaching focuses on making students "understand" a subject through transmitting information via lecturing the students. "Pouring knowledge into the students" depends on what the teacher does and says. It is heavily teacher-centered. In contrast, ACL focuses on what the student does, and how the student engages in learning activities. It sees the teacher more in the role of the moderator and supporter [22]. Here, the primary task of the teacher is to help create a positive learning context for the students and moderate activities that encourage students to engage in deep, or higher-order learning approaches [3]. Higher-order (deep) learning approaches help students to apply knowledge, hypothesize and reflect upon a subject, in contrast to lower-order (shallow) learning approaches such as merely memorizing, identifying, or describing subjects [2].

Both courses outlined here implement ACL principles according to the new university guidelines. Both apply student centered learning principles. Both the embedded systems course and the real-time systems course do that by forming small groups, cooperating in a large project. Additionally, the real-time systems lectures have many self-directed activities such as exercises, group discussions, and tests. Both courses put their main emphasis on the performance of the lab projects/assignments and use the lectures more for supporting the actual practical work. Overall, there is a great emphasis put on students carrying out their own research work in an interactive and stimulating environment.

6. CONCLUSIONS

In this paper we have presented the new Delft MS program on ES that is slated for the academic year 2006. With respect to the 40 EC mandatory part of the 120 EC program, the program focuses on embedded software, ranging from UML to VHDL, which is a departure from the more hardware-oriented approach, such as offered by the Delft CE program. Another feature of the mandatory program is its equalization of the students' prior educational backgrounds during the first part of the program in order to optimize the multidisciplinary education that underlies the ES curriculum. Launched for the coming academic year, no results are yet available. Yet, the authors are convinced this is a step towards

a curriculum that will meet the industrial and academic requirements of the embedded systems field in The Netherlands.

7. ACKNOWLEDGMENTS

The national 3TU Embedded Systems MS curriculum, is a joint effort by the three Dutch technical universities. The joint curriculum has been authored by Jan-Friso Groote and Arlène Louiza (Eindhoven University of Technology), Gerard Smit and Gerrit van der Hoeve (University of Twente), and Arjan van Gemund and Hans Tonino (Delft University of Technology).

8. REFERENCES

[1] ARTEMIS. European Platform for Advanced research and Technology for Embedded Intelligence and Systems. www.cordis.lu/artemis.

[2] J. Biggs and P. Moore. The Process of Learning. Prentice Hall, 1993.

[3] J. Biggs and P. Moore. Teaching for Quality Learning at University. Prentice Hall, 2003.

[4] C. Bonwell and J. Eison. Active learning: Creating excitement in the classroom. ASHE-ERIC Higher Education Report No. 1, George Washington University, Washington, 1991.

[5] Delft University of Technology. www.tudelft.nl.

[6] S. Edwards. Experiences teaching an fpga-based embedded systems class. ACM SIGBED Review, 2(4):56–62, 2005. www.cs.virginia.edu/sigbed/vol2_num4.html.

[7] Eindhoven University of Technology. www.tue.nl.

[8] Embedded Software Lab. www.rtess.ewi.tudelft.nl.

[9] Faculty of Electrical Engineering, Mathematics, and Computer Science. www.ewi.tudelft.nl.

[10] A. v. Gemund. In4073 Course Web Site. www.st.ewi.tudelft.nl/~gemund/Courses/In4073/index.html.

[11] A. v. Gemund. In4073 Resource Web Site. www.st.ewi.tudelft.nl/~gemund/Courses/In4073/Resources/index.html.

[12] A. v. Gemund and J. Tonino. Master of Embedded Systems, Information Dossier in Support of the Application for the New Graduate Studies Test. Technical report, Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Delft, The Netherlands, Sept. 2005.

[13] H.-G. Gross. In4024 course web site. www.st.ewi.tudelft.nl/gross/index files/Page348.html.

[14] IEEE-CS/ACM. Computing curricula 2005: The overview report. www.computer.org/education/cc2001.

[15] ITEA. Technology Roadmap for Software Intensive Systems 2nd edition May 2004. www.itea-office.org/itea_roadmap_2.

[16] D. Jackson and P. Caspi. Embedded systems education: Future directions, initiatives, and cooperation. ACM SIGBED Review, 2(4):1–4, 2005. www.cs.virginia.edu/sigbed/vol2_num4.html.

[17] J. Muppala. Experience with an embedded systems software course. ACM SIGBED Review, 2(4):29–33, 2005. www.cs.virginia.edu/sigbed/vol2_num4.html.

[18] TU Delft. Ms embedded systems elective courses. www.sis.tudelft.nl.

[19] Politecnico di Milano. Rtai - the realtime application interface for linux. www.rtai.org.

[20] A. Sangiovanni-Vincentelli and A. Pinto. Embedded systems education: A new paradigm for engineering schools? ACM SIGBED Review, 2(4):5–14, 2005. www.cs.virginia.edu/sigbed/vol2_num4.html.

[21] R. Seviora. A curriculum for embedded system engineering. ACM Trans. Emb. Comp. Syst., 4(3):569–586, August 2005.

[22] T. Shuell. Cognitive conceptions of learning. Review of Educational Research, 56:411–436, 1986.

[23] University of Twente. www.utwente.nl.

[24] S. Woutersen and A. v. Gemund. X32 Web Site. Via www.st.ewi.tudelft.nl/~gemund/C6/index.html.