Project IST-2001-34820

ARTIST

Advanced Real-Time Systems

Guidelines for a Graduate Curriculum on Embedded Software and Systems

W2.All.Y1

Year 1 Reference Period: April 1st 2002 March 31st 2003

Review: May 6th, 2003

The design of real-time embedded systems requires skills from three specific disciplines: control theory, computer science, and electronic engineering, and their combination. This often involves experts from differing backgrounds, who do not recognize that they address different issues from complementary angles.

The motivation for defining a specific curriculum in embedded systems is mainly to promote the training of engineers with expertise in the above three disciplines.

History

First draft, November 2002: Written by P. Caspi after extensive discussions with F. Maraninchi, Y. Lakhnech and B. Bouyssounouse. This version was preceded by discussions with A. Sangiovanni-Vincentelli, L. Lavagno, P. Veríssimo, J. Engblom, A. Benveniste, M. Torngren, G. Lipari, R. Wilhelm and I. Crnkovic. Contributors to the curriculum repository were also G. Folher, J. de la Puente and W. Yi.

Second draft, December 2002: Written by P. Caspi, F. Maraninchi and G. Lipari with remarks by B. Bouyssounouse and J. Engblom. Some proof reading by F. Dancoisne is also greatly acknowledged.

Third draft, January 2003: Written by P. Caspi and F. Maraninchi, hopefully reflecting the remarks, comments and contributions received during January 2003, mainly proposed by J. Engblom, M. Torngren, G. Lipari, I. Crnkovic, L. Lavagno, R. de Simone, L. Almeida, R. Wilhelm, J. Sifakis and the discussions that took place on January 22nd, at the education meeting (Participants: P. Caspi, F. Maraninchi, F. Laroussinie, A. Sangiovanni-Vincentelli, J. Engblom, T. Willemse, I. Crnkovic, B. Bouyssounouse, M. Garcia-Valls, J. De la Puente.) and at the Artist plenary meeting on January 24th.

Fourth draft, March 2003: Written by F. Maraninchi and P. Caspi, with important contributions from J. Engblom. New contributors of curricula and courses were also Andy Wellings, Hermann Kopetz and Philipp Peti. This is the first version in which almost everything is present, even not in a final form. This version can thus be usefully reviewed.

Fifth draft, March 24, 2003: New contributions by J. Engblom, G. Lipari, G. Buttazzo and I. Crnkovic. Eindhoven curriculum is provided by T. Willemsee and Oldenburg curriculum by W. Dam. N. Suri, R. Gupta, Insup Lee and L. Aceto agreed that their teaching documents be published.

Sixth draft, April 24, 2003: Major rewriting by J. Sifakis and B. Bouyssounouse.

Executive Summary

Although computer based embedded systems have been designed for more than thirty years, they have recently taken an ever-growing economic importance with the generalisation of computers in every day devices such as mobile phones, cars and consumer electronics. This has led Artist to propose a workpackage on education, to investigating whether embedded systems is adequately taught and whether current education meets industrial and research needs.

Three actions were undertaken:

- Elaborate a questionnaire on the industrial relevance of an embedded systems curriculum,
- Survey existing courses and curricula, with a focus on Europe and the USA,
- Propose guidelines for a graduate level curriculum in the area.

This work is characterized by the following difficulties and constraints:

- The diversity of educational systems and approaches in Europe, which complicates the elaboration of concrete and detailed curricula. For this reasons, only guidelines and abstract curricula have been proposed.
- The Artist consortium is geared towards software, and has fewer competencies in hardware and electrical engineering.

Furthermore, the following considerations have shaped the work:

- In current curricula for embedded systems, there is a lack of a unified vision for computer science. This is a surprising fact, which has several explanations:
  1. The multi-disciplinary nature of embedded systems, which have many application domains: telecommunication, automotive, aeronautics and space, mechanics, consumer electronics, etc. Each has tended to organize its own computer studies, and tends to consider computing as a mere technique.
  2. In parallel, computer science curricula have not always taken into account the application domain perspective.
- The lack of maturity of the domain results in a large variety of industrial practices, often due to cultural habits more than to actual technical differences. This situation, unfortunately, has also consequences in teaching. Many courses and curricula that are proposed concentrate on one technique and do not present a sufficiently wide perspective.

As a result, industry has difficulty finding adequately trained engineers, fully aware of design choices.

Parallel to the fragmentation of industrial practices is the fragmentation of research.

The proposal contains elements aiming to remedy this situation. It is based on three ideas:

1. Computer science can play an important role in avoiding this fragmentation. It has a body of knowledge that can serve as a framework, provided it integrates approaches and techniques from the different application domains.

2. Education is one area where this fragmentation can be fought, and the subsequent gaps can be bridged. For this reason, we propose that the different application domains be fully represented in course material and lab work, including experimentation, specific methods and tools.

3. University education is the main point in an engineer's career to learn the foundations. Techniques may evolve drastically over the course of professional career. In this case, in-house training will provide the necessary skills. It is unlikely that the underlying theory and basic concepts be covered in this way.

The proposal considers that the following bodies of knowledge are necessary for a well-rounded education in embedded systems:

- **Control and signal processing,** required for encompassing applications where the embedded device acts as a controller on the physical environment.

- **Computing theory** provides algorithms, methods and tools for formal description and analysis (including verification and validation) of computing devices. This is complementary to control and signal processing, which focuses on the interaction with the physical environment. Interaction between these two bodies in the curricula should give rise to interesting syntheses;

- **Real-time** is the core of the domain. It is composed of several approaches, which should be covered, with a critical and synthesis point of view. For instance, approaches from control, such as synchronous languages, should be more thoroughly taken into account;

- **Distributed systems**, also core knowledge in the field, is not always satisfactorily covered, although it provides answers to relevant and difficult problems. Theory for distributed systems has been developed by the relatively separate ”Distributed and Fault-Tolerant” community;

- **Optimisation and evaluation**, including traditional engineering methods and tools for measuring, evaluating, optimising non-functional properties specific to embedded systems: such as dependability, performance, power consumption, weight, etc.

In addition, a transverse body of knowledge must be considered:

- **System architecture and engineering** providing rigorous design methods and tools for building systems meeting given requirements. These approaches should rely on the above bodies of knowledge. Several systems engineering approaches should be promoted and compared in an embedded systems curriculum.

Finally, **Practice** on real systems and a simulator is essential for training engineers, with the adequate hands-on skills in embedded systems.

From the curricula examples in the appendix, it is appears that none covers the full spectrum of knowledge needed. This can be explained by relative youth of the area as widespread technique in industry. Efforts should be made to meet the emerging industrial needs for engineers.

Implementation of the proposed curricula will require overcoming inertia and difficulties due to cultural habits and fragmentation.

To move forward, a solution could be through student mobility cycling through European centres with complementary expertise as is done on a national level by the Swedish ARTES network. Nevertheless, differences in scale can be a limitation. It may be impractical to exchange students through Europe for short durations. An alternative could be to exchange and train teachers instead.

To gain acceptance for the proposal, this document will be disseminated, debated and refined. This is one main objective for the coming year. We plan to publish this material in conferences and journals and through the respective teaching institutions and national academic authorities of the Artist partners.

Contents

1 Introduction

Although computer based embedded systems have been designed for more than thirty years, they have recently taken an ever-growing economic importance with the generalisation of computers in everyday devices such as mobile phones, cars and consumer electronics.

The design of real-time embedded systems requires skills from three specific disciplines: **control theory, computer science**, and **electronic engineering,** and their combination. This often involves experts from differing backgrounds, who do not recognize that they address different issues from complementary angles.

The motivation for defining a specific curriculum in embedded systems is mainly to promote the training of engineers with expertise in the above three disciplines, even if one of the three is dominant. In particular, this training should include both theoretical and practical aspects in all three disciplines. Specifically, training in computer science should not be reduced to just writing code.

Furthermore, in contrast to classical computer science applications, the development of embedded systems cannot ignore the underlying hardware characteristics. Timing, memory usage, power consumption, and physical failures of components are important aspects. Moreover, embedded systems are generally developed on platforms other than the final target platform which may modify the execution characteristics, and extra-functional properties in particular. Embedded systems programming requires a deep understanding of the underlying hardware.

The current lack of adequately trained engineers and of specific curricula for embedded systems has motivated the Education and Training workpackage, comprised of three main activities:

- Elaborate a questionnaire on the industrial relevance of an embedded systems curriculum,
- Survey existing courses and curricula, with a focus on Europe and the USA. The renewed Berkeley curriculum on mixed Electrical Engineering and Computer Science has been an inspiring example.
- Propose guidelines for a graduate level curriculum in the area.

The body of this document deals with this last issue. Before going into more detail, we discuss the specific characteristics of embedded software and systems. Then we discuss the objectives of this document and its organisation.

1.1 Diversity and Choices

1.1.1 The Diversity of the Embedded System Domain

Embedded systems engineering and science is not a unified field. Historically, it has been split into a large number of sub-disciplines based on applications domains. Examples of such domains are telecommunications, automotive, aeronautics and consumer products. Each of these application domains has its own culture and tradition and, above all, each has developed its own computer engineering education intended to fulfil the (supposed) needs of its specific domain. Thus, we have seen specialised Computer Engineering curricula devoted to a particular speciality, like Railroad CE or Aeronautics CE.

A consequence of this proliferation of sub-disciplines is that the current state of education in embedded systems and software can be rather hard to investigate, since there are so many different actors and departments involved. For this reason, the ARTIST group on Education has made a particular effort to invite and attract people from disciplines that lie outside the core ARTIST network. These include a presentation by Herv Guguen from the Ecole Nationale Suprieure d'Elctricit and inviting Martin Torngren, Professor in Embedded Control Systems, Mechatronics lab at the Department of Machine Design of the Swedish Kungl Tekniska Hgskolan to join the ARTIST Education group.

1.1.2 The Diversity of Practices

Parallel to this diversity is the diversity of practices. Embedded systems offer a very large choice of implementations.

The choice between hardware and software is an important one, which runs through the entire design process. Within software, a large choice of practices can be found, from control-based designs supported by the very popular Matlab/Simulink/StateFlow toolbox - to object-oriented tools. Moreover, development in assembly language and C constitute an important part of the practices.

Another source of diversity is the choice between synchronous and asynchronous designs and implementations, and between language-based approaches and OS-based approaches. The comparison and the analysis of these practices are not always clear nor are they easy to teach.

1.1.3 The Diversity of European Education

The Bologna Declaration (http://europa.eu.int/comm/education/socrates/erasmus/bologna.pdf) has brought some order by proposing a unified system of credits, the ECTS (European Credit Transfer System) allowing student exchanges through Europe.

Furthermore, it proposes a division of higher education into two levels, undergraduate and graduate, to which one can add a postgraduate level.

Another source of diversity is that some countries prefer a deductive style of education, where students go from theory to practice, while others apply a more inductive approach, which adopts the reverse order.

1.1.4 The role of Computer Science in Embedded Software

The Artist consortium is geared towards software, and has fewer competencies in hardware and electrical engineering. We recognise the risk for bias, but have taken every precaution to present a balanced view.

The diversity of practices in embedded systems design is mainly due to two reasons. One is the variety of cultures from application domains (such as using automata or block-diagrams for the description of synchronous systems). Another is due to more fundamental differences, such as hard real-time vs. soft real- time.

In both cases, unification of the industrial practices is may not be necessary (or even possible). However, the curricula for embedded systems should emphasize on the fundamental common parts as well as on the differences between existing practices. It should insist on the fact that, depending on the specific culture of an application domain, different implementation approaches can be used for the same computation model. We believe that computer science has a major role to play, because it can provide a sound and unifying view on the various computation models. Moreover, it has specific bodies of knowledge, such as formal verification, that are essential for the development of correct embedded systems.

But computer science will be able to succeed in this unifying role only if it is able to meet the needs of the embedded system application disciplines. This requires understanding the specific characteristics and problems of these disciplines. For instance, engineers with a strong computer science background may have difficulties when confronted with a Simulink diagram for the first time. Vendor training alone is clearly insufficient to master the underlying theory, principles, and design methodologies required. Tool vendor training, what ever be its quality, is really a concern if it tends to replace academic education. This is why a larger opening toward application domains is necessary if we want computer science to play its fundamental role.

1.1.5 Computer Science and Electrical Engineering

As stated above, the Artist consortium expertise is essentially in computer science. It is difficult to judge to what extent this document opens the way towards a unified view of education for embedded systems will be possible.

Our goal is to provide an education curriculum for computer science. We would consider as an additional benefit if this document were also useful for electrical engineering and contributed to a unified view of the embedded system domain.

1.1.6 Choosing the Level

In the Bologna Declaration, the undergraduate degree serves either as an intermediate step leading towards an engineering degree, or it is the final diploma. In the latter case an embedded system curriculum could make sense.

We have decided to propose a graduate-level curriculum. This avoids the cumbersome process of setting the restrictions and prerequisites needed for an undergraduate curriculum.

An undergraduate curriculum on embedded systems need not actually be a subset of the graduate-level curriculum, but rather a shallower one on all subjects. This would be easier to do in a theoretically oriented (mainly deductive style) educational setting, but can also be done in an inductive setting by choosing simpler, practical examples to drive the learning process.

1.1.7 Levels of Abstraction

Due to European diversity, a curriculum rigidly organised into courses and modules would not be appropriate. For this reason, we propose an abstract curriculum defined in terms of bodies of knowledge: what is important is the knowledge students will finally acquire, not how and when this knowledge is acquired.

These bodies of knowledge are larger than individual courses. This makes agreement easier and it allows a larger degree of freedom for practical implementations. Furthermore, according to the emphasis put on one body of knowledge or on another, one can get a quite large variety of curricula.

At this stage of generality, we do not distinguish between practice and research. Here also, this depends on implementation, which can vary depending on the emphasis put on either more theoretical matters or more applied ones.

1.2 Objectives and Organisation

This document tries to identify these large bodies of knowledge, which, we believe, are both useful and difficult to acquire for embedded systems practice and research.

Currently, training takes place continuously during professional life, and it is not easy to distinguish what should be learned during primary education and during continuous training. Yet, it seems that fundamental bases are really difficult to acquire during continuous training if they haven't been initially learned, and we can think we must focus on them.

Coherently organising the bodies of knowledge identified is not an easy task, and any organisation will probably appear arbitrary, and will have unavoidable overlaps between different components. The curriculum's organization is driven more by fundamental issues than by the structure of the application domains. However, this does not mean that this organisation should be rigidly followed.

For each body of knowledge, we describe its pertinence in the curriculum, the required background and its contents. The background is needed because we do not aim at discussing a full curriculum, but only the complements that concern embedded systems. Thus the general-purpose computer science and engineering background is shortly summarised in a first section and then mentioned when needed as background in particular sections.

The curriculum contents should not be overly detailed, to allow flexibility. At the same time, it should contain the hooks to make it useful and understandable for practitioners. Of particular interest are remarks relating several bodies of knowledge, especially those raising questions on consistency. Furthermore, dependencies between bodies of knowledge are suggested which can provide some basis for ordering the courses.

To improve usability, we provide in appendix, a list of existing courses and curricula with pointers to existing material that can help in implementing it.

We believe this document will help teachers identify what may be lacking in their present curricula, and the reasons why it may be needed.

1.3 Related Work

A number of approaches have been proposed for the definition of computer curricula, in various academic, industrial or mixed contexts. Some of them are closely related to our subject, namely "embedded Systems". Others are more general.

Let us cite: the IEEE/ACM guidelines, CareerSpace and the Swedish national network ARTES.

1. The IEEE/ACM guidelines (http://www.acm.org/education/curricula.html) are very general, and

do not seem to address one of the main aspects we focus on in this document: a curriculum on embedded systems has to borrow courses from various teachings contexts, like computer science, control theory, dependable systems, etc.

2. CareerSpace (http://www.career-space.com/) is focused on Information and Communications Technology (ICT). The list of *job profiles* they propose for the domain does not mention embedded systems. The part entitled “Recommendations for Designing New ICT Curricula” is worth reading, for the curriculum structure proposed.

3. The ARTES (http://www.artes.uu.se/) Swedish strategic research initiative in Real-Time Systems is a network of academic and industrial groups, with the ambition to strengthen the Real-Time Systems competence nationwide. The main focus of ARTES is on graduate education and cooperation between industry and academia. Concerning education, ARTES created a number of useful courses in real-time systems. Furthermore, ARTES has promoted the mobility of graduate students from all over Sweden. This helped increase the efficiency of graduate education, and the interchange of people and ideas.

2 Curriculum Description

2.1 General Computer Science and Engineering Background

We first provide a typical list of computer science and computer engineering skills that may constitute the general-purpose background mentioned above.

- Basic algorithmics (at least in an imperative style), practice of an imperative high level programming language (Ada, Java, C++, ...), practice of C.
- Basic notions on logical gates, combinational and sequential synchronous circuits, simple processor architecture, input/output devices, interrupts, buses. Programming in assembly language.
- Elements of language theory (automata, context-free grammars, regular expressions).
- Implementation of programming languages (interpretation, compilation, mixed solutions). Lexical and syntactic analysis, static analysis, code generation and optimisations.
- Basic operating systems, concurrent and distributed programming.
- Software modelling, analysis and design (life-cycle, testing methodologies and tools, ...)

Social impact and ethics of computer science and computer engineering may also be considered as part of the general background. Indeed, there are unlikely to be specific to embedded systems.

2.2 Basic Control and Signal Processing

2.2.1 Motivations

This is a relatively new body of knowledge in a computer science curriculum and its presence is due to the fact that many applications deal with the control of physical processes or the processing of physical signals. The purpose is not to train specialists in such topics, but we believe that computer engineers need a minimal background. One reason is that in most of these control applications, software runs in tight interaction with the physical environment. If the control system developer is not aware of the problems raised by this tight interaction, he may underestimate the importance of some details and this may have grave consequences on correctness. Another reason, already noted in the introduction, is that control-based design tools such as Matlab/Simulink tend to become de facto standards in the domain and computer engineers should be aware of them.

2.2.2 Background

A general scientific education based on general physics and mathematics is required, with special emphasis on differential and integral calculus. Most computer scientists have a weak background in non-discrete mathematics. They may need a general continuous mathematics course.

2.2.3 Content

For basic control and signal processing, training should provide some notion of physical systems and signal modelling, such as linear time-invariant differential equations. Notions of Fourier and Laplace transforms are important because of the role of CAD tools such as Matlab/Simulink and to illustrate notions such as stability, bandwidth and resonance as well.

Notions of feedback and continuous control should be provided, for instance by teaching the basics of pole placement.

Sampling and sampled control should be addressed. This is important, as it builds a strong basis to computer implementations such as time-triggered ones. Thus Shannon sampling theory should be taught, and then sampled-data control systems should be introduced, along with z-transforms. Additionally, the rationales for the choice of sampling periods should also be taught (this is seldom the case). Similarly, notions of (linear) digital filtering and fast Fourier transforms should be provided.

Discrete event control is also an important aspect. It builds on almost the same material as language and automata theory of computer science, and thus provides a bridge between the two disciplines. Discrete event control favours computer implementations that are event-triggered systems. This is a problem when sampled (time-triggered) and event-triggered systems are to be mixed. The lack of sampling theory for event-triggered systems should be noted, and the methods used in practice to compensate this lack should be explained.

Finally, some aspects of the emerging hybrid control theory could be presented.

2.2.4 Documents, teaching material and practice

This body of knowledge is a basic one in undergraduate Control and Signal Processing studies. Many textbooks, documents and exercises can be found. Of particular interest is:

Edward A. Lee and Pravin Varaiya, *Structure and Interpretation of Signals and Systems*, Addison Wesley, 2002. (http://www.aw.com/info/lee/, http://ptolemy.eecs.berkeley.edu/eecs20/)

Practice should include the practice of popular tools such as Matlab/Simulink/Stateflow or some of their free clones such as Scilab/Scicos (http://www-rocq.inria.fr/scilab/) or Octave (http://www.octave.org).

2.3 Computing Theory

2.3.1 Motivations

In a way, this body of knowledge provides the computer science component for embedded systems. It is often neglected in curricula specific to application domains. Of particular interest is the fact that it shows students that computer behaviour can be as formally described as physical behaviours have been in basic control and signal processing. Finally, it allows formal validation, which is an important point when dealing with safety critical systems.

2.3.2 Background

Basic programming and algorithmics are required here.

2.3.3 Content

The first step consists in introducing elements of programming language semantics. Which semantic to choose ?

- Denotational semantics is important because it builds upon functions which draw close links with basic control and signal processing.
- Axiomatic semantics allows reasoning about programs.
- Operational semantics favours a point of view close to automata and machines.

All three types of semantics could be taught, although it is not easy to present a unified view. Presenting these semantics also allows teaching important tools such as logics, induction and fix-point theory.

The semantics of concurrent systems can then be addressed, associated with temporal logics and verification techniques. In parallel, important theoretical aspects of computer science such as computability and complexity are useful to show students the limits of computing.

Finally, more recent developments can be brought in, such as:

- timed automata and their associated decision (analysis, synthesis) methods,
- elements of cryptography and cryptographic protocols, that are becoming important in many embedded systems (e. g. smart cards),

This body of knowledge can be presented either formally based or from an applied perspective. Indeed, it could be sufficient to teach students how to use available tools, and how to write checkable properties in temporal logic or other input languages. Assertion and property languages used in industrial tools would be very valuable to include as well.

In all these aspects, the common points with basic control and signal processing, for instance in all that deals with functions and automata, should be underlined.

2.3.4 Documents, teaching material and practice

No single textbook covers the entire body of knowledge several textbooks are required: **On semantics:**

Hanne Riis Nielson, Flemming Nielson, *Semantics with Applications: A Formal Introduction*, Wiley, 1992.

On computation:

John E. Hopcraft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 2001

On concurrency:

R. Milner. *Communication and Concurrency*, Prentice Hall, 1989

On verification:

Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*, Springer-Verlag, 1995.

B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen, *Systems and Software Verification. Model-Checking Techniques and Tools*, Springer, 2001.

Practice

This should include hands-on experience with available modelling and verification tools. (such as model-checkers, theorem provers).

2.4 Real-Time Computing

2.4.1 Motivations

This body of knowledge encompasses all aspects of computing, that are specific to real-time: design and validation, languages, algorithms, programming, compiling, operating systems, etc.. This is clearly a core body of knowledge for embedded systems and is the most often taught in existing curricula.

2.4.2 Background

This broad subject requires some knowledge of classical computing in all its aspects.

2.4.3 Content

Important distinction are between soft and hard real-time and also between language-based approaches and system-based ones. It is not always easy to make the best design choice in either case, and this is reflected in teaching.

Dealing with time and concurrency in programming languages has led to the distinction between two paradigms:

- Asynchronous paradigm. This is based on ”tasking” (ADA) or ”threading” (Java), in which the control on task executions is provided by means of timers and priorities. Languages supporting this paradigm are associated with their own run-time model.
- Synchronous paradigm. This is based on logical time and where the coordination of parallel activities is handled within the language itself. In principle, implementations may run on bare machines - without using any real-time operating system at all. Nevertheless, real-time operating systems, as for instance OSEK-Time, may support synchronous execution.

It would be important here to draw links with Computing Theory.

Compilation techniques specific to real-time applications should be taught, either geared towards general-purpose computers or towards specific ones such as DSPs. The problem of estimating worst-case execution time may also be addressed here.

Important topics to be taught are real-time operating systems, scheduling theory and schedulability analysis for real-time systems. It is useful to distinguish between time-triggered and event-triggered systems and between pre-emptive and non pre-emptive scheduling. Basic scheduling algorithms should be taught, such as Table-Driven, Rate Monotonic, Priority Inheritance and Priority Ceiling, Earliest Deadline First.

Student should be trained to design real-time systems, from specification to partitioning the application into concurrent real-time concurrent tasks.

The courses should adopt the right level of abstraction. Although there is a need for standardisation, a commonly accepted design methodology does not yet exist. Here the teacher is free to show one or more methodologies, depending course focus. For instance, some methodology are more useful for given application domains, such as aerospace design. Other methodologies are more useful for co-designing embedded systems.

Real-time component-based design should be introduced here, such as timing aspects in UML.

2.4.4 Documents, teaching material and practice

Here also, several books are needed to cover the wide variety of topics:

On synchronous languages:

N. Halbwachs, *Synchronous programming of reactive systems,* Kluwer, 1993. (http://www.wkap.nl/prod/b/0-7923-9311-2)

Stephen A. Edwards, *Languages for Digital Embedded Systems,* Kluwer, 2000. (http://www.wkap.nl/prod/b/0-7923-7925-X)

On asynchronous languages and real-time operating systems:

Alan Burns and Andy Wellings: *Real-Time Systems and Programming Languages (Third Edition) Ada 95, Real-Time Java and Real-Time POSIX,* Addison Wesley, 2001.

On scheduling:

Jane W.S. Liu, *Real-Time Systems*, Prentice-Hall, 2000.

Giorgio Buttazzo, *HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications*, Kluwer, 1997. (http://www.wkap.nl/prod/b/0-7923-9994-3)

In many cases, articles from real-time systems conferences and industry trade shows can provide good reading on how to build real-time systems.

Practice This should include the hands-on experience and comparison of different languages and real-time operating systems. For instance showing the pros and cons of different approaches on the same benchmark examples.

Practice should also provide the students with hands-on experience running real-time systems code on real hardware platforms. A good idea is to implement some kind of real-time control system where failure to compute in real time has visible side-effects in the real world. Good kits for labs here are the LEGO Mindstorms Robotics kits, which provide cheap and reliable hardware for labs.

For teaching real-time operating-systems concepts, a simulated hardware environment can be used, such as the simulators provided with most embedded development environments or stand-alone products like Virtutech Simics. These allow students to run actual real-time operating systems and experiment with scheduling policies, priorities etc. Research RTOS that both run on PCs are:

- Shark (http://shark.sssup.it), developed at Scuola Superiore Sant'Anna is used as a didactic software for experimenting with scheduling algorithms in many courses in Italy and Sweden.
- MARTE OS (http://marte.unican.es), developed at University of Cantabria, also very well suited for teaching RTOS concepts, and natively supporting the Ada language.

- Some open source real-time OS like for example RT-Linux (http://www.realtimelinuxfoundation.org) and its variants, or Ecos.

2.5 Distributed Systems and Embedded Networks

2.5.1 Motivations

This body of knowledge specialises to distributed computing all aspects of computing: design and validation, algorithms, programming, compiling, operating system, etc.. This is clearly core knowledge for embedded systems and it covers all topics from distributed algorithms to data communication protocols. Close links with computing theory can also be found here: semantics, verification and testing.

For distributed algorithms, a strong link should be established with fault-tolerance, for two reasons:

- All fault-tolerant implementations require some type of redundancy, and this should often be distributed,
- Distribution raises problems of imperfect communication and synchronisation, that require some kind of fault tolerance. For example, protocols need to cope with imperfect communication.

An important body of theory on ”Distributed Fault-tolerant Systems” has been developed over the past twenty years. It has brought important, difficult and fundamental results on distributed algorithms. These results deserve better recognition, and should be included in the curricula.

2.5.2 Background

This is broad body of knowledge requires some awareness of classical computing in all aspects mentioned above, combined with real-time computing.

2.5.3 Content

The design of Distributed algorithms and systems should take into account the following:

- Computing and communication capabilities: Are the communication delays known? Are they bounded? Are clocks available? How precise are they? etc.
- Fault which can impair computations and communications. These strongly depend on the criticality of the application.

The various models for computing, communication and faults should be presented. It is worth noticing here that the terminology in this domain differs slightly from the one used in Computing Theory and in Real- Time Computing. For instance, the terms ”asynchronous”, and ”timed” may have different meanings and these differences should be clarified.

The various distributed algorithms (point-to-point protocols, network protocols, broadcast protocols) and associated problems (atomicity, commitment, elections, consensus, logical/physical clock synchronisation, membership) with their main solutions and limitations should be presented.

The close relationship between distribution and fault-tolerance should be illustrated. The dependability required will determine the complexity and intricacy of the fault model. For safety critical systems, experience shows that high dependability cannot be achieved through trial and errors techniques, because the design process may not converge. Principles for distributed fault-tolerant architectures, successfully used in various application areas, should be discussed and classified according to their real-time and fault-tolerance features. For instance, time-triggered architectures can be included, as well as globally asynchronous, locally synchronous (GALS) architectures. These are the most widely used for fault-tolerant distributed critical control systems. Other popular architectures for industrial applications, such as those based on CAN and Ethernet should also be included. Finally, examples of soft real-time, high-availability systems such as telecommunication switches (Ericsson Erlang OTP and the Linux/TelORB system) can be provided.

2.5.4 Documents, teaching material and practice

Books on the topic include:

Sape J. Mullender (editor), *Distributed Systems*, Addison-Wesley, 1993.

Paulo Verissimo, Luis Rodrigues, *Distributed Systems for System Architects*, Kluwer Academic Publishers, 2001. (http://www.navigators.di.fc.ul.pt/dssa/index.html)

Hermann Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications* Kluwer Academic Publishers, 2001. (http://www.wkap.nl/prod/b/0-7923-9894-7)

Practice Practice should be based not only on simulation but also include experimentation on real platforms.

Experimenting only with workstations linked through a local-area network is usually not sufficient. In the absence of real platforms, a simulated network and computers can be used in labs to provide sense of reality.

Another important goal of practice in this domain is to show students that distributed computing is several orders of magnitude more difficult than ordinary programming and this is why experimentation is important.

2.6 Extra-functional properties: Optimisation and Evaluation

2.6.1 Motivations

This body of knowledge deals with the evaluation and optimisation of the extra-functional properties of embedded systems, related to power usage, throughput, quality of service and dependability.

Beyond the superficial differences between these properties, they share common techniques for assessment and optimisation. These techniques should be taught to engineers and computer scientists regardless of the areas their future careers will bring them to: optimisation and evaluation are base pillars of all engineering activity.

2.6.2 Background

This is an advanced theme: the techniques that have to be used to ensure a guaranteed level of quality are likely to depend heavily on the other technical themes, like ”distributed systems”, ”real-time computing”, etc., and may be taught in these themes. It is also necessary to provide a background in optimisation (operation research), probability, statistics and queueing theory. This background can be part of the theme, or a pre-condition.

2.6.3 Content

Performance The student must be introduced to the basic models for performance evaluation, usually based on queuing theory. Then, soft real-time systems should be introduced as a generalisation of hard real-time systems when uncertainty in the temporal characteristics of the system is present.

An introduction to dynamic real-time systems must be provided. The theme should contain an overview of basic techniques for scheduling soft real-time applications in a dynamic environment. The concept of resource reservation should be introduced, both in the network (presenting algorithms like WFQ, WF2Q+, SFQ, etc.), and in operating systems (Resource Kernel, CBS, Fair Scheduling, etc.).

It is then possible to show the connection between the performance models and the scheduling algorithms, by analysing the performance of example applications (like a multimedia streaming program) using the above mentioned techniques.

Finally, the topic of QoS management and QoS negotiation should be explained. In particular, it is important to introduce the concept of adaptive application and adaptive system. Basic on-line and off-line optimisation techniques can be used to maximise the overall quality of the system. A survey of basic techniques like imprecise computation, elastic scheduling, value based scheduling, adaptive reservation should be given. Also, an introduction on feedback scheduling techniques. In this last topic there are interesting connections with control system theory.

Dependability Techniques for dependability improvement and assessment evaluation should be introduced : first combinational techniques (fault trees, reliability networks) and then dynamic ones based on Markov chains and derived techniques (stochastic Petri nets for instance). This allows to introduce important aspects of fault tolerance such as selective and massive redundancy, detection, replacement and masking, etc. It is important to note here the analogy with the techniques used for performance evaluation. Furthermore, links with Computing Theory and Control can be stressed by viewing Markov chains as stochastic automata and linear time-invariant differential systems. For instance the analogy between state lumping in Markov chains and automata minimisation is interesting, as well as the use of Laplace transform for solving Markov chains.

Power consumption There is a growing body of techniques being proposed to reduce the power consumption of computer systems, not only in hardware, but also addressing software issues: power management and evaluation, techniques like algorithmic tweaks to lower computational needs, parallelisation to save power, turning off unused parts of a system, operating system techniques like measuring behaviour, architectures for low- power systems, including both hardware and software.

Memory consumption and stack usage Due to the hard limits on the resources available in an embedded system, it is necessary that engineers learn how to track and determine the amount of memory used by and needed by a system. Of particular interest is the size of stacks for software, since process stacks can occupy a significant amount of the available RAM in an embedded product. Techniques that need to be taught include static estimation of memory usage, and the use of dynamic techniques like high-water-marking and profilers to determine memory usage. The demonstration of stack measurement techniques that are present in many real-time operating systems is a good idea

Execution time Determining the execution time of process, tasks, or other units of computation is of utmost important to the construction of embedded real-time systems. Students should be taught techniques applicable to both soft and hard real-time systems, like using hardware clocks to time executions, watching bus transactions with logic analysers or oscilloscopes, or using static analysis techniques. The support in current programming tools for time measurement (RTOS tool suites and stand-alone tools) should be included in the curriculum.

2.6.4 Documents, teaching material and practice

R. Ramakumar, *Reliability Engineering: Fundamentals and Applications*, Prentice Hall,

Giorgio Buttazzo, Luca Abeni, Marco Caccamo, Giuseppe Lipari, *Soft Real-Time Systems: Predictability vs. Efficiency*, Kluwer Academic Publishers, 2003 (to appear).

Robin A. Sahner, Kishor S. Trivedi and Antonio Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package (The Red Book)*, Kluwer Academic Publishers, 1996. (http://www.ee.duke.edu/ kst/book2.html)

Practice To practice on the development of soft real-time systems, the Shark kernel (http://shark.sssup.it) provides several soft algorithms and libraries that can be used by the students to make their multimedia applications.

For the topic of performance evaluation, the student should play with reservation-based systems. It is possible to use Linux/RK (http://www-2.cs.cmu.edu/ rajkumar/linux-rk.html), a modification of the Linux kernel that implements a resource reservation scheduler. The student should be able to run soft real-time and legacy applications on such system. Possible lab projects are:

- implementation of a QoS-aware MPEG player
- implementation of a QoS manager system that dynamically accepts new applications with a certain guaranteed quality of service

Concerning dependability, an interesting point is to make students understand the difficulty of modelling and computing very small probabilities, as required in very critical systems, and the need for a very careful design method in this case.

2.7 System Architecture and Engineering

2.7.1 Motivations

This theme is a transverse one, motivated by the fact that all the aspects that have been studied separately in previous sections are related to each other; building a system that is, for instance, distributed, fault-tolerant and real-time, is a problem in itself, and the solution is a multi-aspects development method.

Furthermore, these development methods need to cover the whole life-cycle of systems: project management, problem understanding and requirements specification, analysis and design, implementation, verification and validation methods.

Component-based development (CBD) has achieved significant results in software in certain engineering domains. In embedded systems CBD is considerably less utilised, mainly because inability of component-based technologies to manage extra-functional and real-time properties. However in some particular domains such as telecommunication and consumer electronics this approach has proved to be successful. New CBD technologies that combine real-time and component-based aspects, such as RT CORBA or RT Java, are relatively new for RT community and usually not a part of curricula. There is a need for education of CBD basic principles, the current CBD and middleware technologies

2.7.2 Background

All the previous sections.

2.7.3 Contents

The theme can be organised according to application domains. In each of them, one or several development method taking into account several aspects and covering the whole life cycle should be addressed.

A (probably non exhaustive) list of application domains and examples:

- Distributed safety-critical control system.
- Soft real-time QoS oriented distributed application.
- Component-based designed applications.

This theme should stress the need for system architecture, considered as a way of fulfilling the multi-aspect requirements of an application by coherently organising these requirements and the solutions used to meet them. The notion of platforms as backbones for architectural design can also be introduced.

Architectural styles, UML for RT, and similar, are examples of techniques used for design. Different design approaches should be included: a top-down approach and structural programming, object-oriented approach, component-based approach. Implementation part should include principles for writing good programs (general, efficient, modifiable, modular, etc.). Test parts should include verification part (testing the functionalities) and validation part (validation of the requirements). Finally maintenance procedures, software and process measurements, and system disposal should be included.

This theme should stress the idea of model-based design which allows to anticipate problems, shortens the path toward correct design decisions and provides earlier validation. This clearly includes multi-paradigm modelling and *simulation* techniques. Simulation is not only a technique used for early validation, but can be considered as a problem in itself, in the embedded software domain. Indeed, the software cannot be developed without some model of the environment, be it a physical process or a piece of hardware which is not already developed. Both cases require reliable and semantically well-defined simulation techniques. Simulation environments like Metropolis[1] or Ptolemy [2] may be used.

Further, basic concepts of component-based development should be included: Component specification, software architecture and component-based approach, designing component-based system, designing components as reusable entities, component specification, component evaluation and certification, component testing. In addition, the component-based development process such as software product lines should be included. More advanced topics are component compositions, system emerging properties and component properties. The RT and embedded specific part includes the following topics: properties and interfaces of real-time components, composition and predictability of RT and other extra-functional properties of component-based systems. Further, an insight in different component technologies and middleware should be included as a practical part: CORBA, RT CORBA, RT Java and some of component models implemented in different domains of embedded systems.

Another aspect is project management. Different types of project and different phases of a project including a team-work, specific roles in the project, supporting methods and tools should be explained.

2.7.4 Documents, teaching material and practice

Ivica Crnkovic and Magnus Larsson, *Building Reliable Component-Based Software Systems*, Artech House Publishers, 2002. (http://www.idt.mdh.se/ icc/)

Jim Cooling, *Software engineering for Real-Time Systems*, Addison Wesley

[1]Metropolis: http://www-cad.eecs.berkeley.edu/ polis/
[2]Ptolemy: http://ptolemy.eecs.berkeley.edu/

2.8 Practice

Practice is an essential component for a well-rounded education in embedded software and systems. A curriculum on embedded software and systems cannot rely only on theoretical courses. Laboratory work should be used to illustrate each course.

Beyond illustrating each course individually, it is important to tie them together through common, multi-course projects. In a curriculum such as the one described here, students are taught many different theories and methods. Practice is important to demonstrate their joint applicability for real systems. This will provide students an understanding of their practical interest. Practice, especially in the form of larger projects, gives students the opportunity to integrate the different pieces of acquired knowledge.

Practical labs also offers students a chance to get familiar with embedded systems programming tools, such as cross-compilers, various debugging tools such as emulators, simulators, JTAG-probes, bus analysers, logic analysers, etc.

This helps the student understand problems specific to resource-constrained environments, the importance of choosing the right platform and the ways in which the system architecture affects the runtime characteristics.

3 Conclusion

This work presents Guidelines for a Graduate Embedded Software and System European Curriculum. It is characterized by the following difficulties and constraints:

- The diversity of educational systems and approaches in Europe, which complicates the elaboration of concrete and detailed curricula. For this reasons, only guidelines and abstract curricula have been proposed.
- The Artist consortium is geared towards software, and has fewer competencies in hardware and electrical engineering.

We propose contents that, we believe, should be found in a curriculum in Embedded Software and Systems. In the Appendix, examples of courses can be found as well as examples of actual curriculum implementations. The choice of contents was driven fundamental issues, points that are important to the subject and difficult, and which are not likely to be easily acquired through on the job training. This has led us to identify five pillars of fundamental knowledge on which to build a curriculum:

- **Control and signal processing,** required for encompassing applications where the embedded device acts as a controller on the physical environment.
- **Computing theory** provides algorithms, methods and tools for formal description and analysis (including verification and validation) of computing devices. This is complementary to control and signal processing, which focuses on the interaction with the physical environment. Interaction between these two bodies in the curricula should give rise to interesting syntheses;
- **Real-time**, is the core of the domain. It is composed of several approaches, which should be covered, with a critical and synthesis point of view. For instance, approaches from control, such as synchronous languages, should be more thoroughly taken into account;
- **Distributed systems**, also core knowledge in the field, is not always satisfactorily covered, although it provides answers to relevant and difficult problems. Theory for distributed systems has been developed by the relatively separate ”Distributed and Fault-Tolerant” community;
- **Optimisation and evaluation**, including traditional engineering methods and tools for measuring, evaluating, optimising non-functional properties specific to embedded systems: such as dependability, performance, power consumption, weight, etc.

In addition, a transverse body of knowledge must be considered:

- **System architecture and engineering** providing rigorous design methods and tools for building systems meeting given requirements. These approaches should rely on the above bodies of knowledge. Several systems engineering approaches should be promoted and compared in an embedded systems curriculum.

Finally, **Practice** on real systems and simulators is essential for training engineers, with the adequate hands- on skills in embedded systems.

It can be noted in the appendix curriculum examples, that none of them fully implements our proposal. The one that seems to obey most closely these guidelines is at the Vienna Technical University, though it is too focused on a given network technology. This may be due to:

- The concept of a curriculum is not well-defined. It depends on the local education system and on the teaching institution. In many cases, students are free to choose the courses they want, provided they follow prerequisites and their program is accepted by their supervisor. Thus, identifying a precise curriculum is difficult.

- The educational system has significant inertia and takes time to adapt to evolving industrial needs. In particular, it takes time to organise curricula and gather specialists from the required disciplines.

The lack of adequately trained engineers has impacts on current industrial practices. The lack of specialists fully aware of the various design techniques, induces fragmentation of practice, difficulties to communicate and share experience within a company.

For instance, aeronautics and space, which are closely related and address similar problems, use very different methodologies, design flows, tools, etc. for reasons that do not seem at all obvious.

Another consequence is fragmentation of research. Some examples have been cited in this document. For instance, very basic terminology related to timing and communication, words like ”timed”, ”untimed”, ”synchronous” and ”asynchronous”, do not have the same meaning for the Language Theory communities as they have for the Distributed and Fault Tolerant Systems community.

Here also, this results in communication difficulties and fragmentation of research. Moreover, this is a circular phenomenon: since teachers don’t understand each other, neither do students. Papers are poorly understood, conferences are separated, and finally, communities do not interact.

Education is clearly a place where these issues can be addressed. For instance, some differences in practices between industries can be justified, for cultural reasons. Training students to experiment and compare several approaches can be a good way to bridge gaps.

This is why we believe that proposed abstract curriculum provides a basis for avoiding these drawbacks and is worth being implemented. This will require overcoming inertia and difficulties due to cultural habits and fragmentation.

To move forward, a solution could be through student mobility cycling through European centres with complementary expertise as is done on a national level by the Swedish ARTES network. Nevertheless, differences in scale can be a limitation. It may be impractical to exchange students through Europe for short durations. An alternative could be to exchange and train teachers instead.

To gain acceptance for the proposal, this document will be disseminated, debated and refined. This is one main objective for the coming year. We plan to publish this material in conferences and journals and through the respective teaching institutions and national academic authorities of the Artist partners.

It was also suggested that the document serve as a basis for some kind of European ”certification”, assessing the compliance of curricula to this document. This is clearly an interesting possibility. Yet, besides this point, just reaching agreement on its content, among colleagues and Academic authorities is in itself a worthy goal.

A Structured Set of Courses

This appendix presents examples of courses that either come from the ARTIST repository, or have been gathered by simply browsing the Internet.[3]

The courses are quite naturally organised according to the theme structure of the document.

A.1 Basic Control and Signal Processing

A.1.1 Berkeley's EECS20

URL :

http://ptolemy.eecs.berkeley.edu/eecs20/

Short description :

- Prerequisite :

  There is only one formal prerequisite: Math 1b, Calculus

  In the Spring of 2000, we found that having one of Math 53, 54, or 55 prior to or concurrent with this course had a significant impact on performance. Math 54 had a slightly greater impact than the others.

- Main Theme and objectives :

  This course is an introduction to mathematical modeling techniques used in the design of electronic systems. Signals are defined as functions on a set. Examples include continuous time signals (audio, radio, voltages), discrete time signals (digital audio, synchronous circuits), images (discrete and continuous), discrete event signals, and sequences. Systems are defined as mappings on signals. The notion of state is discussed in a general way. Feedback systems and automata illustrate alternative approaches to modeling state in systems. Automata theory is studied using Mealy machines with input and output. Notions of equivalence of automata and concurrent composition are introduced. Hybrid systems combine time-based signals with event sequences. Difference and differential equations are considered as models for linear, time-invariant state machines. Frequency domain models for signals and frequency response for systems are investigated. Sampling of continuous signals is discussed to relate continuous time and discrete time signals. Applications include communications systems, audio, video, and image processing systems, and control systems. A Matlab-based laboratory is an integral part of the course. Although the course may be taken after Math 1b, certain topics from Math 54 (matrices and vectors) and Math 55 (sets and functions) may be helpful.

[3]This browsing also shows that American students are often provided with richer on-line information on courses, than European ones.

Bibliography :

Two papers describing this course and its lab are available:

Edward A. Lee and Pravin Varaiya, ”Introducing Signals and Systems – The Berkeley Approach,” Proc. of the First Signal Processing Education Workshop, Hunt, Texas, October 15 - 18, 2000.

Edward A. Lee, ” Designing a Relevant Lab for Introductory Signals and Systems,” Proc. of the First Signal Processing Education Workshop, Hunt, Texas, October 15 - 18, 2000.

A.2 Theory of Computing

- A course proposed by Florence Maraninchi . (an adapted copy of : http://www-ensimag.imag.fr/formations/enseignements/1e-annee/math.html#thla1)
- A course by Luca Aceto, Aalborg University
  (http://www.cs.auc.dk/ luca/SS/ss.html).
- Verification by Kim Larsen, Aalborg University
  (http://www.cs.auc.dk/ kgl/VERIFICATION98/coursepage.html).

A.2.1 Formal Language Theory (Florence Maraninchi, ENSIMAG)

URL: http://www-ensimag.imag.fr/formations/enseignements/1e-annee/math.html#thla1

Contact: Florence.Maraninchi@imag.fr

Short description :

- Course exists already ? Y
- Course level : undergraduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 5
  (*1 year = 60 credits*)
- Prerequisite diplomas :
- Prerequisite knowledge and professional experience :
- Main Theme and objectives : The course develops topics from mathematics that have proven to be relevant to computer science: formal language theory, computabilty, recursion and induction.

Description :

- Regular languages : finite automata, regular expressions, determinisation and minimalisation of finite automata
- Context-free languages : context-free grammars, normal forms ; pumping lemma, closure properties ; push-down automata
- Computability : Turing machines, computable functions, decidable problems and reduction.
- Decision problems on languages.
- Partial orderings - Well founded orders - Fixed points theorems.
- Definitions of languages using systems of equations

Bibliography : J.E. Hopcroft, J.D. Ullman, Introduction to automata theory, languages, and computation, Addisson-Wesley, 1979

A.2.2 Syntax and Semantics (Luca Aceto, Aalborg)

URL :

http://www.cs.auc.dk/ luca/SS/ss.html

Contact : Luca Aceto

Short description :

- Course exists already ? Y
- Course level :
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits :
  (*1 year = 60 credits*)
- Prerequisite diplomas :

  The course assumes knowledge of basic discrete mathematics, such as the one that you have been using in the course Dat 1/D5D. Those of you who would like to refresh their memory, or who are not confident about their understanding of the basic mathematics used in this course, are strongly encouraged to read, e.g., Chapter 0 of Sipser's book and/or Chapter 1 of the book Elements of the Theory of Computation (2nd edition) by Lewis and Papadimitriou. As practice makes perfect, I also recommend that you work out some of the exercises to be found in those references. Try, for example, exercises 0.2-0.5, 0.7, 0.10-0.11 in Sipser's book and/or exercises 1.1.1-1.1.4, 1.5.1-1.5.2 in the book by Lewis and Papadimitriou.

  Note: The above exercises are not really part of this course. How many you attempt to solve depends only upon your level of confidence with the mathematics that will be used as the course progresses.

- Main Theme and objectives : The goal of this course is to introduce the main computational models and techniques that underlie the syntax and semantics of programming languages. As you will discover during this semester and in the remainder of your studies, the theory that we shall cover in this course has important applications in, e.g., compiler design (as covered in Josva Kleist's Sprog og Oversttere course), text processing, and many facets of program analysis and implementation. At the end of the course, the students will be familiar with the basic computational models of Finite State and Pushdown Automata, with the classes of grammars that generate the languages recognized by these abstract computational devices, and with basic operational and axiomatic semantics of programming languages. We shall try to emphasize the role played by semantics in compiler and language design, program analysis, and good programming practice. (The notions of pre- and post-conditions, loop invariants and the like have their root in a formalism for reasoning about programs called Hoare logic, which is the foundation of axiomatic semantics.)

  – Syntax

Programming languages, like all other natural or formal languages, have a syntax and a semantics. Their syntax describes the collection of legal programs by means of a set of rules that allow one to construct new syntactically correct programs from smaller ones. This set of program formation rules gives a formal definition of the syntax of a programming language. Such descriptions are now quite common, and are often given in BNF notation. The development of high level programming languages — like FORTRAN, COBOL and LISP — was one of the major advances in Computer Science in the 1950s. These languages allowed programmers to specify commands in mnemonic code and with high level constructs such as loops, arrays and procedures. With the development of these languages, and of the more sophisticated ones that followed, it became important to understand their expressiveness (that is, what programs can be written using them) as well as the characteristics of the simplest computing machines that can translate them into machine language. This brought about the study of formal languages and the automata that recognize them. The goal of the first part of the course is to introduce some of these classes of formal languages, their grammars, and the simplest machines that recognize them.

– Semantics

  To quote from the introduction of Matthew Hennessy's book The Semantics of Programming Languages: an Elementary Introduction using Structural Operational Semantics (John Wiley and Sons, New York, N.Y., 1990):

  It is not possible to have a true understanding of a programming language without a mental model of its semantics, i.e., ”how the language works”.

  Programmers usually develop this mental picture by day-to-day use of the language via a compiler and an interpreter. This empirically obtained knowledge of ”what programs do” is imprecise, haphazard and hard, if not impossible, to work with. In the second part of this course, we shall see that the mental picture of ”how programs should work” can be developed and understood in a machine independent way by using the tools of the ormal semantics of programming languages. More specifically, we shall focus on the techniques of Structural Operational Semantics (SOS), and will have a brief look at Axiomatic Semantics.

Description :

- Syntax
  - Introduction to the course; Mathematical Preliminaries: Strings and languages; A taste of finite automata.
  - Nondeterministic automata and their relationships to deterministic ones.
  - Regular expressions and finite automata.
  - Languages that are not regular.
  - Context-free grammars.
  - Pushdown automata and their relationships to context-free grammars.
  - Languages that are not context-free.

- Semantics
  - Introduction to the formal semantics of programming languages.
  - Natural and structural operational semantics for the language While.
  - Relationships between the natural and structural operational semantics for the language While.
  - Operational semantics for extensions of the language While.
  - Natural semantics for blocks and procedures.
  - Natural semantics for blocks and procedures (continued).
  - A taste of program verification - partial and total correctness of programs.
  - A formal system for partial correctness assertions.

Bibliography :

The main textbook for the syntax part of the course is Introduction to the Theory of Computation by Michael Sipser. The author maintains a list of errata for the current edition of this book. This book is quite expensive, but you will also use it as your main textbook in the DAT3/FS course next year.

For the semantics part of the course, I plan to use two textbooks that are available on line. These are:

Hanne Riis Nielson, Flemming Nielson: Semantics with Applications: A Formal Introduction. Wiley Professional Computing, (240 pages, ISBN 0 471 92980 8), Wiley, 1992.

In July 1999, a revised edition has been made available for download, in gzip'ed postscript, postscript (recommended), or pdf formats.

Hans Httel. Pilen ved trets rod .

Whenever necessary I shall supply notes and chapters from other references to supplement the material in these books

Available material: *(course notes, exercices, tool support, student projects...)*

Syntax:

The Java Computability Toolkit. The Java Computability Toolkit (JCT) contains a pair of graphical environments for creating and simulating NFAs, DFAs, and high level Turing Machines. For an in-house simulator for Finite Automata, look at the DAT2 project developed last year by some of your colleagues!

Semantics for the language While:

Miranda implementations of evaluation of expressions, and of the natural and structural operational semantics. (Courtesy of Hanne Riis Nielson and Flemming Nielson.)

LETOS – A Lightweight Execution Tool for Operational Semantics. This is a lightweight tool proposed by Pieter Hartel to aid in the development of operational semantics. To use letos, an operational semantics must be expressed in its meta-language, which itself is a superset of Miranda. The letos compiler is smaller than comparable tools, yet letos is powerful enough to support publication quality rendering using LaTeX, fast enough to provide competitive execution and tracing using Miranda or Haskell, and versatile enough to support derivation tree browsing using Netscape. Letos has been applied to a Java Secure Processor, which is a version of the Java Virtual Machine intended to run on smart cards. Letos has also been used with subsets of various programming languages. The latest version of letos is described in detail in a paper, which is included with the program and some sample inputs in a gzipped tar file.

A.2.3 VERIFICATION (Kim G. Larsen, Aalborg)

URL :

http://www.cs.auc.dk/ kgl/VERIFICATION98/coursepage.html

Contact: Kim G. Larsen (Aalborg University)

Short description:

- Course exists already ?
- Course level :
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits :
  (*1 year = 60 credits*)
- Prerequisite :
- Main Theme and objectives :

  Software is becoming increasingly complex and there is a growing awareness within software engineering practice that formal verification techniques are helpful in dealing with this growing complexity. In particular, formal verification is finding its way into areas such as embedded systems as used in consumer electronics, time dependent systems occurring in safety critical systems and communication protocols from the telecommunication industry. In the course we shall concentrate on fully automatic formal verification (coined model checking by Ed Clarke). The course will cover the area through anumber of prototypical automatic verification tools illustrating the models, specification formalisms, and underlying algorithmic techniques within the area. Opportunities of getting hands-on experience with these tools will be given.

  CWB: Assuming you are somewhat familiar with the tool CWB (the Concurrency Workbench), its underlying model (CCS) and specification language (the modal mu-calculus), we will illustrate various algorithmic techniques for equivalence and model checking.

  SPIN: The tool SPIN, developed by researchers at ATT, is one of the most commonly used tools in the verification of real life communication protocols. One of the distinguishing features of this tool is the use it makes of the techniques based on the so-called 'partial-order reduction method' to alleviate the well known problem of combinatorial state-explosion.

  visualSTATE: visualSTATE is developed by the danish company BEOLOGIC A/S and is a design tool used by many companies for developing embedded software in a variety of embedded applications ranging from simple controllers to very large advanced interactive simulators. Together with BRICS at Aalborg and Computer Systems Section at Denmarks Technical University, the BEOLOGIC is now developing a new generation of the visualSTATE tool that will move the limit on the complexity of the embedded software that can be verified by several orders of magnitude. The underlying technique is exploiting the art of symbolic model checking using Binary Decision Diagram (BDDs), a notion which will be covered in the course. The developed technique itself - on which a patent is currently being filed - will also be illustrated.

UPPAAL: Since the emergence of the notion of timed automata introduced by Alur and Dill in 1990 a number of real-time verification tools has emerged, including UPPAAL. UPPAAL has been developed in collaboration between Uppsala University and BRICS at Aalborg. In addition to classical techniques for dealing with discrete state-spaces, real-time verification tools such as UPPAAL deals with the continuous part of a state-space through a variety of data structures and techniques for representing and manipulating subsets of the Euclidean space.

Description:

- Introduction. Modal mu-calculus. Model checking and Equivlance Checking, CWB.
- SPIN, LTL and Partial Order Model Checking.
- Reduced Ordered Binary Decision Diagrams
- visualSTATE, Compositional Verification.
- Modelling Real Time Systems. Timed Automata and Timed Logics. UPPAAL
- Algorithms for Verifying Real Time Systems.
- Formal Methods Workshop.

Bibliography:

Available material: *(course notes, exercices, tool support, student projects...)*

A.3 Real-Time

There is a large sample of courses associated with this theme. Many of them covers several aspects and could also be considered as curricula.

- Software methodologies for Real-Time Systems (G. Buttazzo, universita di Pavia)
  http://www.sssup.it/ giorgio
- Real Time Systems (in Ada)(J. de la Puente, universidad de Madrid)
- Hard Real-Time Computing Systems (G. Buttazzo, universita di Pavia)
  http://www.sssup.it/ giorgio
- Advanced Topics in Real-Time Systems (Gerhard Fohler, Malardalen University)
  http://www.idt.mdh.se/gfr
- Real-Time Systems and Programming Languages (Andy Wellings, University of York)
  http://www.cs.york.ac.uk/rts/RTSBookThirdEdition.html
- Real-Time Systems (Josep M. Fuertes, Universitat Polytèchnica de Catalunya)
  http://webesaii.upc.es

A.3.1 Software methodologies for Real-Time Systems (Giorgio Buttazzo, Pavia)

URL :

http://www.sssup.it/ giorgio

Contact: buttazzo@unipv.it

Short description :

- Course level : undergraduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 12
  (*1 year = 60 credits*)
- Prerequisite knowledge and professional experience :

  Undergraduate courses in math and physics.

  Fundamentals of programming languages.

  Fundamentals of computer architectures.
- Main Theme and objectives :

Description : To provide software methodologies for developing real-time control systems. More specifically, the aim of the course is to introduces basic techniques for interacting with phisical devices, perform data acquisition and sensory processing, achieve digital filtering, drive dc motors, exchange data among computers, and interprete results through statistical analysis. Neural networks are also introduced as a technique for performing sensory data processing, prediction and adaptive control.

Available material: *(course notes, exercices, tool support, student projects...)* Material provided by the teacher.

Lab experience supervised by a Teaching Assistant.

A number of educational robotic devices are available for the lab experiments or can be build during the course.

The course includes a laboratory part, in which students learn how to control a physical system, such as a dc motor, an inverted pendulum, or a mobile video camera.

A.3.2 Real Time Systems (in Ada) (Juan A. de la Puente, Madrid)

URL :

http://www.dit.upm.es/jpuente/strl/

Contact : Juan A. de la Puente

Short description:

- Course exists already ? Y
- Course level : undergraduate - graduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 5
  (*1 year = 60 credits*)
- Prerequisite :

  Computer programming, computer organization, operating systems
- Main Theme and objectives :

  - To understand the specific problems of real-time systems, and the special characteristics that make them different from other kinds of computer systems.

  - To learn about software development methods for building reliable real-time systems, especially those related to time measurements, resource scheduling, and software organization, and to understand their operational principles.

  - To learn about some specific tools (especially programming languages and operating systems) for developing real-time systems.

Description:

- Introduction to real-time systems.
- Programming languages and operating systems - Overview of Ada.
- Cyclic executives - discussion.
- Review of concurrent programming - Threads and Ada tasks, task synchronization.
- Time management, clocks, timeouts. Periodic & sporadic tasks.
- Task scheduling. Fixed-priority preemptive scheduling. Response time analysis. Introduction to dynamic scheduling.
- Distributed systems. Clock synchronization. Response time analysis in distributed systems.

- Interrupt and device handlers. Response time analysis with device handlers.
- Applications: timing analysis applied to design, case studies.

Bibliography: Alan Burns & Andy Wellings. Real-Time Systems and Programming Languages. 3rd ed.

Other books: Kopetz, Liu

Available material: *(course notes, exercices, tool support, student projects...)*

Presentation slides, previous years' examination tests http://www.dit.upm.es/jpuente/strl/

GNAT/ORK Ada compiler & RT kernel

A.3.3 Hard Real-Time Computing Systems (Giorgio Buttazzo, Pavia)

URL :

http://www.sssup.it/ giorgio

Contact : buttazzo@unipv.it

Short description :

- Course exists already ?
- Course level : graduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 12
  (*1 year = 60 credits*)
- Prerequisite :

  Undergraduate courses in mathematics
  Fundamentals of programming languages
  Operating system

- Main Theme and objectives :

  Hard Real-Time

  To provide the main software methodologies for supporting predictable real-time systems at the operating system level, in the presence of hard and soft timing constraints.

Description :

- 1. Introduction

  Basic concepts
  Motivations
  Typical real-time applications
  Terminology and notation
  Where timing constraints come from?
  Counterintuitive behaviors
  Speed vs. predictability

- 2. Task scheduling

  General formulation

Notes on complexity

Classification of scheduling algorithms

Performance metrics

Classical best effort algorithms

Heuristic vs. optimal algorithms

-

- 3. Periodic task management

  Timeline scheduling

  Utilization factor

  The Rate Monotonic algorithm

  Earliest Deadline First

  Response Time Analysis

  Processor Demand Criterion

- 4. Aperiodic event handling

  The general problem

  Background vs. server approaches

  Polling Server

  Deferrable Server

  Total Bandwidth Server

  Handling execution overruns

  Constant Bandwidth Server

- 5. Accessing shared resources

  The mutual exclusion problem

  The priority inversion phenomenon

  Possible solution

Bibliography :

Giorgio Buttazzo, ”HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications”, Kluwer Academic Publishers, Boston, 1997.

A.3.4 Advanced Topics in Real-Time Systems (G. Fohler, Malardalen)

URL: http://www.idt.mdh.se/gfr

Contact: gerhard.fohler@Mdh.se

Short description:

- Course exists already ?
- Course level : graduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 7
  (*1 year = 60 credits*)
- Prerequisite : Operating Systems, basic Real-time Systems

Description :

Hard Real-Time

deepen knowledge in advanced topics of real-time systems including aperiodic task scheduling, RT operating systems, RT network, RT case studies;

a lab exercise with theoretical and practical development of overload algorithm

A.3.5 Real-Time Systems and Programming Languages (Andy Wellings, University of York)

URL :

http://www.cs.york.ac.uk/ andy

Contact : Andy Wellings

Short description:

- Course level : graduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 8
  (*1 year = 60 credits*)
- Prerequisite : A basic understanding of sequential programming, computer architecture and operating systems.
- Main Theme and objectives :

  This course considers the requirements and techniques for implementing real-time systems and how they have influenced the design of real-time programming languages and operating systems

  On completion of the module, the students should:

  understand the techniques that can be used to construct reliable and timely real-time systems,

  understand different models of concurrency and how they can be used to facilitate the programming of real-time systems,

  understand different approaches to programming fault tolerance in real-time embedded systems,

  have an appreciation of the facilities provided by Real-Time POSIX and Real-Time Java,

  understand how to undertake scheduling analysis of real-time systems,

  have practical skills in developing real-time software in Ada 95.

Description :

Requirements for programming real-time systems. Concurrent programming. Real-time facilities and deadline scheduling. Facilities for interacting with special purpose hardware. Error handling. Reliability of real-time systems; fault-tolerance, atomic actions. Languages considered include: C (with Real-Time POSIX), Real-Time Java, Ada (in detail, for practicals), and Modula.

Bibliography :

Burns A and Wellings A J, Real-Time systems and programming languages (3rd ed.), Addison Wesley, 2001

Available material :*(course notes, exercices, tool support, student projects...)*

Teaching support material: http://www.cs.york.ac.uk/rts/RTSBookThirdEdition.html

A.3.6 Real-Time Systems (Josep M. Fuertes, Universitat Polytèchnica de Catalunya)

URL: http://webesaii.upc.es

Contact: Josep M. Fuertes

Short description:

- Course exists already ? Yes, in two industrial engineering schools (to be implemented in a third school)
- Course level : graduate
  (undergraduate [0,3], graduate]3,5], doctoral]5,8])
- *ECTS credits : 4*
  (1 year = 60 credits)
- *Prerequisite knowledge and professional experience :*

  Basics on mathematics, control theory and programming
- *Main Theme and objectives :*

  To describe the main characteristics of computer based control and monitoring applications. To provide basic knowledge on real-time systems theory and technology in order to construct reliable and timely real-time control systems.

Description:

1. Introduction:

   Requirements of control and monitoring applications

   Architectures for real-time control systems (from centralized to distributed systems, I/O interfaces - data acquisition, communication interfaces, man-machine interfaces, etc -)
2. Basic control systems programming:

   Cyclic executives

   Micro-controller implementation
3. Multitasking real-time control systems:

   Basic concepts on multitasking systems (concurrent programming, mutex, synchronization, scheduling)

   Implementation/technology issues (rtos, languages)
4. Co-design of scheduling and controllers:

   Interactions between control and scheduling

   Integrated approaches: analysis, design and implementation

A.4 Distributed Systems

- Dependable Distributed and Embedded Systems, Neeraj Suri, Darmstadt Technical University
  http://www.deeds.informatik.tu-darmstadt.de/suri/classes/classes.htm
- Networks in Distributed Embedded Systems, Luis Almeida, Universidade de Aveiro

A.4.1 Networks in Distributed Embedded Systems (Luis Almeida, Universidade de Aveiro)

Motivation Technological advances in hardware made possible the embedding of both processing and communication functions in highly integrated, low-cost components, fostering the use of a distributed approach in the particular field of embedded systems. These systems, called Distributed Embedded Systems (DES), are typically part of intelligent automatic equipment with a high degree of autonomy or operator support. In most cases, they have a strong impact on human lives, either because they are used within important economic processes, e.g. complex machinery in factories, or because they control equipment that directly interacts with people, e.g. transportation systems.

Mainly along the 90s, DESs have evolved towards highly distributed systems, following the concept of encapsulating different functionality in separate intelligent nodes (e.g. intelligent sensors and actuators). This resulted in a growing number of nodes, leading to higher connectivity and scalability requirements. However, it also resulted in higher system complexity, even with simpler individual nodes, and led to a stronger impact of the network on the global system properties.

Therefore, the network within a DES plays now a vital role since it supports all the interactions among the set of interconnected nodes and, generally, supports all global system services. The network, or generally the communication system, determines, to a great extent, the support for properties such as composability, timeliness, flexibility and dependability as well as determines the efficiency in the use of system resources. Hence, it is essential for those that will be engaged in the design or utilisation of DES to be aware of the consequences of using specific network protocols.

Background In order to understand the issues covered in this course it is helpful to have at least an introductory-level knowledge of computer networks, digital transmission, real-time scheduling, response time analysis. These however are not mandatory and the course will be relatively self-contained.

Contents The course starts with an introductory part about computer communications in order to unify concepts and nomenclature, to review basic concepts about networks and to present the traffic requirements of the DES applications. The course will then address the use of networks in real-time systems, discussing issues such as message scheduling, event/time-triggered communication. The impact of the physical media, network topology and medium access control in these issues and, consequently, in the systems properties must be thoroughly analysed.

At a higher level, the cooperation models such as client-server, producer-consumer and publisher-subscriber must also be reviewed and case studies of typical protocols used today in DESs (CAN, TTP, Profibus, WordFIP, LonTalk, Ethernet) must be carried on, with emphasis on timeliness properties e.g., the network access delay.

Other important topics to include in the course contents are the synchronisation of the nodes and of the clocks, error detection, and techniques to achieve fault-tolerance. Recent topics that generate considerable interest such as wireless networks, sensor webs and real-time over Internet should also be discussed at least at an introductory level.

A.4.2 Dependable Distributed and Embedded Systemsv (Neeraj Suri, Darmstadt Technical University)

URL: http://www.deeds.informatik.tu-darmstadt.de/suri/classes/classes.htm

Short description:

- Prerequisite : Basic background in OS, SW Engg., Discrete Math and basics of Distributed Systems.
- Main Theme and objectives :

  Embedded systems are pervasively appearing in all facets of life - mobile systems & e-services, cars, planes, banking, www etc etc. As we increasing become dependant on the services provided by these systems, consequently it becomes more and more unacceptable to have these services fail. Thus the value of these systems lies in their being able to provide sustained delivery of desired services in spite of perturbations encountered by them, i.e., dependable delivery of services.

  Focusing on distributed and networked systems and SW/OS's, the course builds up basic principles of dependability. One part of the course addresses issues in design for dependability. The second part covers aspects of assessing & verifying the actual provision of dependability. This covers aspects such as testing and verification and validation of SW and OS's.

  Structurally, I will be building up background in both dependability concepts and dependability-relevant distributed system/SW/OS operations. The connotation of embedded systems & services will be related throughout. Later on (time permitting) I also plan to touch on some related security issues.

Bibliography: There is a dearth of good books covering the extremely broad area of dependable systems, distributed systems and esp. SW Engg. The following texts are cited as suggestions - they do not represent the full scope of the course. I do plan to use to use the first mentioned book for some parts over the course, though I will be pointing to research papers along with the book material.

(1) Distributed Systems for System Architects; P. Verissimo, L. Rodriques: Kluwer Press 2001. ISBN 0-7923-7266-2.

(2) Fault Tolerant Computer System Design; D. K. Pradhan; Prentice Hall 1996 (This book seems to be sometimes in print, sometimes not I will try to use this as a supplement only given its limited availability. The plus of this book is wide coverage of topics and extensive reference list though it s a bit dated at present)

(3) Fault Tolerance in Distributed Systems; P. Jalote, Prentice Hall, 1994 (old but good book with a FT heavy slant as indicated by the title)

(4) Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating System; M. Singhal and N. Shivaratri; McGraw-Hill Publishing Company, New York. 1994, 525 pages (oriented more towards OS though with excellent coverage of some distributed protocols)

A.5 System Architecture and Engineering

- Design of Embedded Systems: Models, Validation and Synthesis (A. Sangiovanni-Vincentelli, UC Berkeley)

  http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/class/index.html

- Advance Topics in Software Systems Software Reliability Methods and Embedded Systems (Insup Lee, University of Pennsylvania)

  http://www.cis.upenn.edu/ lee/02cis640/

- Real-time computer control systems (Martin Torngren, KTH)

  http://www.md.kth.se/RTC/RTCC/

- Component-Based Software Engineering (I. Crnkovic, Malardalen University)

  http://www.idt.mdh.se/kurser/cd5490/

A.5.1 Design of Embedded Systems: Models, Validation and Synthesis (Alberto San Giovanni-Vicentelli, Berkeley)

URL :

http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/class/index.html

Contact: Alberto Sangiovanni-Vincentelli

Description :

This course is about the design of embedded real-time systems. Embedded real-time systems are pervasive in today's world. The methodology used for the design of these devices is still based on principles and tools that are not adequate for the complexity of the applications being developed today. The most important characteristic of these systems is the massive use of programmable components to achieve the design goals. Hence, their design requires the use and optimization of both hardware and software.

In this course, we will present the principles of a methodology that favors design re-use, formal verification and software implementations. The basic tenet of the methodology is orthogonalization of concerns, and, in particular, separation of function and architecture, computation and communication. Platform-based design will be presented as a paradigm that incorporates these principles and spans the entire design process, from system-level specification to detailed circuit implementation.

We will place particular emphasis on the analysis and optimization of the highest levels of abstraction of the design where all the important algorithmic and architectural decisions are taken. The notion of behavior will be analyzed and the role of non-determinism in specification will be explained. We will present the basic models of computations that are needed to represent the behavior of most of the designs: Finite-State Machines, Synchronous Languages, Data Flow Networks, Petri Nets, Discrete Event Systems. We will outline the use of a unified framework developed in collaboration with Prof. Lee to compare the different models of computation and a unifying theory to allow the composition of different models of computation to describe a complete system. We will introduce the Ptolemy tool for analysis and simulation of heterogeneous specifications.

We will then introduce the notion of ”architecture” as an interconnection of building blocks that are capable of carrying out computation. ”Optimal” architecture choice will be presented as the selection of a particular set of computational blocks in a ”library” of available components, and of their interconnection. The evaluation of the quality of a selected architecture is based on a ”mapping” process of behavior to the computation blocks, including programmable components. The mapping process should be accompanied by an estimate of the performance of the design once mapped onto the architecture.

Performance simulation will be introduced and a variety of techniques with different accuracy-speed trade-off will be presented.

Interface synthesis and formal verification will be introduced as enabling technologies for the design of complex systems by composition. Hardware-software co-simulation will be presented and analyzed in details illustrating a number of approaches and commercial offerings. Among the hardware-software systems, we will focus on the latest offerings by the FPGA community that include micro-processor cores in a ”sea” of FPGA blocks.

Then we will explore the issue of embedded software design including optimization and the selection of scheduling algorithms to maximize resource utilization. The issue of communication among different parts of the design including the interface between hardware and software will be discussed.

The framework for architecture selection, software optimization and real-time operating system design will be POLIS, VCC and their successor: Metropolis. We will use them to illustrate some of the key points of the design methodology.

Moreover, the student will gain experience on actual system designs through graded laboratories dealing with realistic design examples and case studies (like automotive seat belt design, vehicle controller design and elevator controller design). During lab sessions, students will use and analyze various design tools including: Ptolemy, POLIS, Metropolis, VCC, Co-ware, Seamless, Proceler, Xilinx Virtex FPGA and Anadigm FPAA.

Pre-requisite :

There are no pre-requisite for this course but it is preferable to have had some exposure to logic synthesis and simulation. At the end of the course the student will have a basic understanding of the system-level design issues in the areas of specification, validation and HW/SW implementation.

A.5.2 Advance Topics in Software Systems, Software Reliability Methods and Embedded Systems (Insup Lee, University of Pensylvania

URL :

http://www.cis.upenn.edu/ lee/02cis640/

Contact : Insup Lee

Short description:

- Prerequisite :

  General understanding of opearating systems and programming Languages, and being able to ”think”, ”implement” and ”communicate.”

- Main Theme and objectives : This course is to study various methods that have been developed for improving software reliability, including specification formalisms, analysis and verification techniques, automatic code generation, testing, run-time monitoring, etc. The study will be based on embedded systems application whenever appropriate. The purpose of this course is to provide background to students so that they can understand and work on the currect research issues in these areas. Students will be encouraged to identify potential research topics and carry them as class projects. Students are expected to participate in class discussion, to attend CIS colloquiums, to present the summary of their independent readings to the class, and to carry our programming and/or significant term projects.

Bibliography: Software Reliability Methods, Doron A. Peled, Springer, 2001.

Real-Time Systems, Jane W.S. Liu, Prentice Hall, 2000.

A.5.3 Real-time computer control systems (Martin Torngren, KTH)

URL :

http://www.md.kth.se/RTC/RTCC/

Contact :

Martin Trngren

Short description:

- ECTS credits : 5
  (*1 year = 60 credits*)

Description:

- Course overview, Area introduction
- Timing Problems in Control Applications
- Real-time implementation and operating systems
  Lectures: Real-time systems design and implementation
  Invited lecture: Microntroller architectures and execution time analysis
- Communication and Scheduling
  (including TTP, CAN and comparisons)
- Systems Architecture - general concepts and examples
  The Architectural trade off analysis method
  System level design: orthogonalization of concerns and platform based design

A.5.4 Component-Based Software Engineering (Ivica Crnkovic, Malardalen)

URL :

http://www.idt.mdh.se/kurser/cd5490/

Contact: Ivica Crnkovic

Short description :

- Course exists already ? Y
- Course level : graduate-doctoral
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits : 7
  (*1 year = 60 credits*)
- Prerequisite : At least 150 ECTS credits or corresponding, from which it should be at least 90 from computer science or computer engineering or corresponding subjects.
- Main Theme and objectives :

  Components

  * Give the student an overview of component-based software engineering

  * Get students an overview of requirements of component-based dependable systems

  * Get student familiar with reading, analysing and presenting research papers and writing stat-of-the-art reports

Description:

- CONCEPTS OF COMPONENT-BASED SOFTWARE ENGINEERING

  On the Definition of Concepts in Component-Based Software Engineering

  On the Specification of Components
- SOFTWARE COMPONENTS

  Component-Based Software Development Life-Cycles

  Semantic Integrity in Component Based Development

  Role-Based Component Engineering
- SOFTWARE ARCHITECTURE

  Software Architecture and Component Integration

  Component Models and Software Architecture

- DEVELOPING USING SOFTWARE COMPONENTS
  Component Evaluation
  Component Composition and Integration Testing
  Component-based systems
- REAL-TIME SOFTWARE COMPONENTS
  Requirements for Real-Time Components
  Building Real-time Systems from COTS Component
  Contracts for Safety-Critical Real-Time Systems

Bibliography :
Building Reliable Component-Based Software Systems, Ivica Crnkovic & Magnus Larsson

Available material: *(course notes, exercices, tool support, student projects...)*
The course is based on the book Building Reliable Component-Based Software Systems
The book web page: http://www.idt.mdh.se/cbse-book
The course web page: http://www.idt.mdh.se/kurser/cd5490/

B Implementation examples

This section presents examples of existing curricula.

- Real-Time Systems (H. Kopetz, Vienna Technical University)
- Introduction to Embedded Computer Systems(R. Gupta, University of California, Irvine)
  http://www.cecs.uci.edu/ rgupta/ics212.html
- Embedded Systems (R. Wilhelm, Universität des Saarlandes)
- Hard Real Time (Gerhard Fohler, Malardalen University)
  http://www.idt.mdh.se/magister
- Real-Time Systems(W. Yi, Uppsala University)
- Component-Based Software Engineering (I. Crnkovic, Malardalen University)
  http://www.idt.mdh.se/ icc/
- Master of Science in Embedded Systems at the TU/e (J.F. Groote, R.H.J.M. Otten Eindhoven Technical University)
- Undergraduate and Graduate Curriculum on Embedded Systems and Micro-Robotics at the University of Oldenburg (Werner Dam, University of Oldenburg)

B.1 Real-Time Systems at Vienna Technical University

Students, that decide to specialize in technical computer science, have the possibility to attend the following lectures to gain deeper knowledge about real-time systems:

Real-Time System: (Lecture)

This lecture aims at teaching the basic concepts necessary to design, develop and implement distributed real-time systems. This includes:

- Basic concepts and terminology
- Reliability
- Modeling of real-time systems
- Clock synchronisation
- Communication protocols
- Scheduling
- Operating systems
- Validation and verification
- ...

Fault-Tolerant Systems: (Lecture + Lab)

This lecture aims at teaching the basic concepts of fault-tolerance computing. The following basic concepts are addressed,

- Basic terminology
- Concepts of Dependability
- Error models
- Maintenance
- Error Detection
- Fault tolerant computer systems
- Replica determinism
- Voting

- N-redundancy
- Case studies
- ...

Embedded Systems Programming (Lab + some introduction lectures):

This lecture teaches programming of embedded systems, especially real-time systems, including:

- Programming of sensors and actuators
- Real-time communication protocols (TTP/A and TTP/C)
- Simulation and automatic code generation (Matlab/Simulink)

Distributed Real-Time Systems Engineering: (Lecture + Lab)

This lecture teaches the development of distributed real-time systems. This lecture is accompanied by a laboratory. Based on the knowledge of the ”Embedded System Programming” lecture, different aspects of real-time systems programming are addressed.

These lectures are all offered by the ”Real-Time System Group”. The head of this institute is Prof. Hermann Kopetz.

A lecture not offered by this group but also relevant in the context of real-time systems is: ”Computer Aided Verification” and the corresponding ”Formal Verification” laboratory This lecture deals with the theory and practice of computer aided verification of hard and software (e.g. real-time model checking with Uppaal, PVS).

B.2 Introduction to Embedded Computer Systems (Rajesh Gupta, University of California, Irvine)

URL :

http://www.cecs.uci.edu/ rgupta/ics212.html

Contact : Rajesh K. Gupta

Short description :

- Course exists already ? Y
- Course level : graduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- Prerequisite :

  An undergraduate degree in ICS is required for this graduate course. Basic courses in digital hardware, algorithms and data structures, elementary calculus, and probability.
- Main Theme and objectives : Increasing integration of communications, multimedia and processing and relentless digitization of data (including even RF data) continues to expand the scope and complexity of embedded systems. To appreciate these advances, and to productively contribute to future advances of these systems, a critical appreciation of the underlying technology underpinning is a must. The goal of this course is to develop a comprehensive understanding of the technologies behind the embedded systems, particularly, those using computing elements (processor, DSP, or ASSPs). The students develop an appreciation of the technology capabilities and limitations of the hardware, software components for building embedded systems, and methods to evaluate design tradeoffs between different technology choices.

  Course Rationale and Relationship to ICS Curriculum:

  Continuing advances in system software and hardware components now present exciting opportunities in building embedded systems for applications ranging from embedded control, multimedia, networking and information and biomedical appliances. Building these systems, particularly for highly integrated micro-electronic technologies and mobile applications, presents a challenge at every of level abstraction from gate-level designs to complex runtime systems. Even with a detailed technical knowledge in a specific technology area that make up an embedded system, a good system design would require understanding of the design tradeoffs across choice in technologies that make up the system. For instance, is it better to a particular interface as a gate-level logic or build the functionality into device driver software. This course fills this gap by presenting basic characteristics and usage model of the technologies that make up an embedded system and describing their relations

Description :

- Introduction to embedded systems: classification, characteristics and requirements.
- Timing and Clocks in Embedded Systems
- Task modeling and management. Real-time operating system issues.
- Signals: frequency spectrum, and sampling, digitization (ADC, DAC), signal conditioning
- Modeling and characterization of embedded computing systems.
- Embedded Control and Control Hierarchy.
- Communication strategies for embedded systems: encoding, and flow control.
- Fault Tolerance
- Formal Verification

Bibliography : H. Kopetz, ”Real-time Systems,” Kluwer, 1997

Available material: *(course notes, exercices, tool support, student projects...)*

B.3 Embedded Systems (R. Wilhelm, Universität des Saarlandes)

Contact : Reinhard Wilhelm

Description:

- Modelling and Programming
  - Classifications of systems (transformational, reactive, embedded, real-time)
  - Formal basics: finite automata, Mealy and Moore machines. Standard algorithms (NFA-¿DFA, minimization)
  - Formal basics: Omega automata, Buechi and Muller automata, timed automata. Timed languages and timed regular languages.
  - Statecharts. Syntax and Statemate semantics. The Statemate system.
  - Synchronous programming. Overview; Esterel in detail: syntax, correctness, and semantics (intuitive, constructive behavioral).
- Hardware and Applications
  - Overview of compiler structure with focus on backend techniques. Standard algorithms for code selection, register allocation and instruction scheduling.
  - Specific code generation techniques: data routing, address optimization, phase coupling techniques, SIMD code generation, software pipelining.
- Reliability
  - Task scheduling and schedulability analysis.
  - Short overview of model checking.
  - Static analysis and abstract interpretation; timing validation based on static analysis.
- Bus systems and Operating Systems
  - Network topolgies and bus protocols: CAN, MAP, Flexray, TTTA.
  - Overview of real-time operating systems with typical requirements and characteristics.

B.4 Hard Real Time Systems (Gerhard Fohler, Malardalen)

URL :

http://www.idt.mdh.se/magister

Contact: Gerhard Fohler

Short description:

- ECTS credits : 60
  (*1 year = 60 credits*)
- Prerequisite diplomas :

  Bsc

  basic computer science; real-time systems
- Main Theme and objectives : set of courses and project to provide comprehensive education in real-time systems as preparation for real-time research.

Description:

- Advanced Real-time System
- Scientific Methodology
- Parallel Systems
- Safety-Critical Real-Time Systems
- Project in real-time systems

Bibliography:

Available material: *(course notes, exercices, tool support, student projects...)*

B.5 Real Time Systems (W. Yi, Uppsala University)

URL: http://user.it.uu.se/ yi/courses/rts/dvp-rts-02/02.html

Contact: Wang Yi

Short description:

- Main Theme and objectives :

  The aim of the course is to introduce a special class of time-sensitive computer systems known as real-time systems whose behaviour must satisfy timing-constraints (i.e. deadlines). They are often embedded in safety-critical applications such as modern vehicles, process control, and traffic control etc, and therefore they are also known as embedded systems. The course covers topics: scheduling theory (especially rate monotone scheduling, schedulability analysis, response time analysis), real-time operating systems, real time programming languages, execution time analysis, resource control protocols, methods and software tools for modelling, simulation and verification of real-time systems.

Description:

- Introduction
- RTOS
- WCET
- Scheduling non periodic tasks
- Scheduling periodic tasks
- Soft RT Tasks
- Resource Access Protocols
- RT Communication
- Modeling and Verification
- Timed Automata

Bibliography :

Real Time Systems, Jane W.S. Liu, Prentice Hall, ISBN 0-13-099651-3, published 2000

UPPAAL in a Nutshell, av Kim Larsen, Paul Pettersson och Wang Yi. In International Journal of Software Tools for Technology Transfer, 1(1-2), December 1997, pages 134-152.

HARD REAL-TIME COMPUTING SYSTEMS - Predictable Scheduling Algorithms and Applications, Giorgio Buttazzo ISBN: 0-7923-9994-3

Real-Time Systems and Programming Languages, Alan Burns and Andy Wellings, Addison Wesley.

Available material: *(course notes, exercices, tool support, student projects...)*

Software packages for course assignments:

LegOS (RT Operating System)

FpsCalc (Schedulability analysis)

UPPAAL (Modelling, Simulation and Verification)

B.6 Component-Based Software Engineering (Ivica Crnkovic, Malardalen)

URL: http://www.idt.mdh.se/magister

Contact: Ivica Crnkovic

Short description:

- Level : graduate
  (*undergraduate [0,3], graduate]3,5], doctoral]5,8]*)
- ECTS credits :
  (*1 year = 60 credits*)
- Prerequisite :

  At least 120 university course points or corresponding, from which it should be at lease 60 from computer science or computer engineering or corresponding subjects.
- Main Theme and objectives :

  Component-Based Software Engineering - Building Reliable Component-Based Systems

Description:

- CONCEPTS OF COMPONENT-BASED SOFTWARE ENGINEERING On the Definition of Concepts in Component-Based Software Engineering On the Specification of Components
- SOFTWARE ARCHITECTURE Software Architecture and Component Integration Component Models and Software Architecture DEVELOPING
- SOFTWARE COMPONENTS Component-Based Software Development Life-Cycles Semantic Integrity in Component Based Development Role-Based Component Engineering
- USING SOFTWARE COMPONENTS Component Evaluation Component Composition and Integration Testing Component-based systems
- SOFTWARE PRODUCT-LINES Components in product line architectures
- REAL-TIME SOFTWARE COMPONENTS Requirements for Real-Time Components Building Real-time Systems from COTS Component Contracts for Safety-Critical Real-Time Systems

Bibliography :

CBSE - putting pieces togehter, Heineman and Councill

Building large-scale component-based applications, Brown

Building systems from Commercial Components, K. Wallanu et. al

Available material: *(course notes, exercices, tool support, student projects...)*

Software packages for course assignments:

B.7 Master of Science in Embedded Systems at the TU/e

B.7.1 Preceding bachelors

The TUE bachelors *Elektro- and Informatietechniek* and *Technische Informatica* are the primary preceding bachelors for the master of science in Embedded Systems. It is conceivable that graduates from the bachelors *Werktuigbouwkunde, Technische Wiskunde* and *Industrial Design* enroll in this master program, although they may have to satisfy certain prerequisites.

B.7.2 Area of expertise

The goal of this master program is primarily to educate students in the design of embedded systems at an academic level with the most suitable means. A graduate has become a professional engineer who has in-depth knowledge of the whole software/hardware hierarchy concerning embedded systems and who can oversee the design of such systems. He is up-to-date concerning realization techniques and has the skill to quickly assess and acquire new techniques that become available. More specifically, he is well-educated in the following skills and concepts:

Requirements engineering. To determine a priori what the properties of a system under construction are, both from a technical viewpoint, as from the viewpoint of use.

Architecture. Knowledge about and experience in the use of high level architectures of embedded systems. Understanding the influence of architecture on the usability, maintainability, cost and life span of products and product families.

Behaviour. The capability to specify and analyze the behaviour of embedded systems prior to use or production. The capability to specify protocols that link embedded systems.

Realization. The capability to realize an embedded system by available means now and in the future. Knowledge of compilers, platforms and trends in silicon technology.

Testing. Knowledge of and experience in testing of embedded systems.

Relation to society. The place of embedded systems in society, in particular understanding the relation between embedded system design and its impact on daily life. The skill to present and communicate designs and ideas. Being able to work in a multidisciplinary setting.

B.7.3 Participating institutes and faculties

- Division of computer science
  1. Area of expertise: Design and analysis of systems (Groote)
  2. Area of expertise: System architecture and networks (Lukkien)
  3. Area of expertise: Formal methods (Baeten)
  4. Area of expertise: Algorithms (dBerg)

- Department of Electrical Engineering
    1. Area of expertise: Design technology for electronic systems (Otten)
    2. Area of expertise: Signal processing (Bergmans)
    3. Area of expertise: Multimedia systems (Corporaal, vMeerbergen, dHaan)
- Embedded Systems Institute, ESI (Rem, Beenker, dWith)

B.7.4 Job market

Currently, there are many job openings for electrical and computer engineers. The region around Eindhoven is one of Europe's leading areas in embedded systems design. Students graduated from the designer's program of Software Technology are also in high demand by the industries in the vicinity. Therefore, we expect that embedded systems engineers will be in very high demand. As an example, Philips has set up an in-house part-time course on *Embedded Systems Architecting* that takes four to six years to complete in order to fill its own vacancies. The first year of this course is offered by ESI in a slightly adapted form, to which employees of a variety of companies are sent.

B.7.5 Curriculum

The curriculum consists of three major blocks of 40 ects-points each:

1. The mandatory courses.
2. The elective courses.
3. The final *master thesis* project.

The curriculum is organized such that it is in principle possible to start with any trimester in the first year. However, it is preferable to start in the first trimester. Starting in the second and especially the third trimester is more difficult, due to dependencies between some of the courses.

B.7.6 Mandatory courses

The mandatory courses form the central part of this program. They reflect the broad spectrum of embedded systems from software architecture and product requirements to the design of systems on a chip. Special courses deal with automated reasoning, which has become an important tool to establish correctness of embedded systems, performance modelling and the position of embedded systems in society.

The courses are:

Software architecting. 4 *ects*.
This course provides insight in the design and design process of architectures for distributed and/or embedded software systems. The main topics of this course are:

- Introduction to architecture (objectives, definitions, architectural views).
- Process of architectural design (stakeholders, requirements).
- Description techniques for architectures (BCK, Darwin, Koala, Unicon, UML).
- Architectural styles and distributed design patterns
  (Pipe-and-filter, Blackboard, Publish-subscribe, Client-Server, Layering).
- Fundamentals of component technology
  (taxonomy, composability, types of dependencies, black/grey/white box).
- Distributed component architectures (Corba).
- Understanding Legacy Architecture.

By means of a case study, the design process and the design techniques are exercised.

Analysis of embedded systems. 4 *ects*.

This course teaches how to capture requirements concerning the behaviour of a system in natural language. It subsequently provides the means to describe the behaviour of embedded systems in a concise mathematically precise way. For this, process algebraic techniques are used (μCRL). Third, several ways to model the requirements formally are offered (assertions, behavioural equivalence and modal logics). Finally, using various verification techniques and tools, it is shown that the behavioural design meets the requirements. The whole assignment is carried out on an embedded system currently being designed.

Automated reasoning. 4 *ects*.

Insight in the translation of a diverse range of problems to formulas such that they can be solved by automatic manipulations using computer programs.

Many problems, among others in the area of the verification of computer systems, can be described as a large formula that must be shown to be always true. In this course several techniques are presented that can be used for this problem. Not only questions as soundness or completeness of these techniques are important, but especially efficiency and usability are subjects that are considered. Specifically, the following topics are dealt with:

- resolution as a proof rule for propositions and several algorithms based on resolution.
- binary decision diagrams as an efficient representation of boolean expressions.
- unification; resolution on predicates.
- reasoning modulo equations, term rewriting.

Software testing. 4 *ects*.

Different views on testing and testability (various test equivalences). Testing techniques for finite state machines (homing sequences, state identification, conformance testing, UIO sequences). Statistical testing (models of Jelinski Moranda, Littlewood). Test coverage (statement, branch and path coverage). Measurement of adequacy by program mutation. Hardware testing (stuck-at faults). Test derivation from formal and informal specifications. Advanced testing techniques (e.g. using bit hashing).

Performance modelling for embedded systems. 4 *ects*.

Introduction to stochastic processes (Poisson process, Markov chain, Markov process); elementary queueing models; queueing models with priorities; networks of queues; applications in distributed embedded systems.

Multiprocessor systems. 4 *ects*.

Methodology for obtaining an implementation on a (given) structure of processors starting from an executable specification of the application. Topics:

- multiprocessor structures (hardware, and system software)
- real-time operating systems
- concurrency (task- and instruction level, data parallelism)
- executable, analysable specifications
- mapping of tasks to (co-)processors
- assignment of data to memory elements
- optimization of concurrency, timing, and power aspects
  - analyses
  - code transformations
  - compilation and synthesis techniques
  - (dynamic) reconfiguration

Platforms (for multimedia design technology). 4 *ects*.

The design of multimedia systems and their mapping to programmable platforms. Emphasis is on efficient data control for high performance and low power consumption, beside the utilization of parallelism at the instruction and task level. The methodology is based on stepwise code transformation, starting from an initial specification and leading to an efficient application oriented implementation.

Systems and networks on silicon. 4 *ects*.

Efficient realization of embedded systems (mostly with multimedia applications) with real-time constraints in silicon, with due consideration for size and power. The lecture will move from existing embedded processors (with some weakly programmable processors) via the hardware and software architecture description at system level (with emphasis on communication aspects with combined control and data processing) to large scale implementation with shared use of communication resources (networks on chip).

Design of systems-on-chip. 4 *ects* (lab).

Students have to realize a system on a silicon carrier during this course. A number of cad-tools, from specification to mask layout, have to be used in this course.

Societal impact of the emerging embedded system technology. 4 *ects*.

This course concerns the influence of embedded technology on society. Starting from a historical perspective, the effects of industrialization and computerization are illustrated in economical and social terms. Using a sketch of the expected development in embedded system technology (miniaturization, increased processing power, reduced power consumption) several potential scenarios for the development of society are discussed.

The development of embedded systems is also discussed from the product viewpoint. What are success factors for products and for which reasons did certain products find their demise. How does the design of a product family influence the cost and earning capability during the lifespan of a product. What is the effect of open development (standardization, code sharing) and how effective are patents. How is the embedded industry organized.

What are the responsibilities of a professional engineer. Which means are available to recognize and exercise these responsibilities (ethics, law).

I.1	software architecting	automated reasoning	multiprocessor systems	elective	elective
I.2	performance modelling	analysis of embedded systems	platforms	elective	elective
I.3	software testing	design of systems on a chip	systems and networks on silicon	societal impact	elective
II.1	elective	elective	elective	elective	elective
II.2 II.3	Master thesis				

Figure 1: Curriculum master *embedded systems*

B.7.7 The elective courses

The elective courses can be chosen from all courses in the master program offered by the department Electrical Engineering and the division Computer Science, provided the students have sufficient knowledge to follow these. All electives need to be approved by the examination committee. At most 14 *ects* credits can be used for courses of a more introductory nature (homologation). A practical period (stage) of 20 *ects* credits can be part of the program and is advised for those students that do not need credits for homologation.

The following list contains existing recommended courses from which students can choose:

- Introduction to VLSI programming
- Advanced switching theory
- Provable safety of security protocols
- Applied system analysis
- Introduction into formal methods
- The language and the structure of mathematics
- Time-discrete signal processing
- Heuristic search techniques
- Telecommunication
- Seminar modern information systems
- Programming by calculation
- Signal processing systems
- Stochastic processes
- Stochastic signals
- Design of programs and proofs
- The Linux operating system
- Software project management
- Control theory
- Computer graphics (two courses)
- System identification
- Parallel calculations with applications
- Visualization
- Robotics
- Object oriented databases (doublecourse)
- Robust control
- Neural networks
- Speech recognition and synthesis
- Image processing for multimedia
- Modern signaltransformations
- Sensors and actuators
- Mechatronics
- Design of multiprogrammes
- Advanced system design
- Information system architecture
- Physiological measurements
- Software tools
- Neuromonitoring
- System architecture
- Optics and optical systems
- Software construction
- Knowledge systems
- Information theory
- Technology of integrated circuits
- History of computer science
- Hypermedia structures and systems
- Semiconductors
- Electro-/optical communication systems
- Telecommunication networks
- Electromagnetic compatibility
- Radio sysytems

B.7.8 The master thesis

For the master thesis a design, or modification of a design of an embedded system must be made. The design must be novel and preferably is situated at the edge of what is technologically possible. The master thesis is intended to provide the experience of carrying out a substantial project. The outcome is not always predetermined. Although this practice is not to become prevalent, the master thesis may also consist of research in embedded system design.

The master thesis can be carried out under supervision of any member of the division of Computer Science or the department of Electrical Engineering. It is possible to do the research either within the university or in a company in the Netherlands or elsewhere.

B.7.9 Expected number of first year students

Based on polls we expect that yearly at least 20 students with a B.Sc. in Electrical Engineering and Information Technology and 30 students with a B.Sc. in Computer Science and Engineering will start with this master program. Depending on the level of external advertisement, we expect at least 10 other students enrolling each year.

B.7.10 Relation to research schools and other forms of education

The master program in embedded systems will cooperate closely with existing research schools (e.g. IPA, ASCI) and will actively adapt to the developments in organizational, scientific or industrial contexts.

The master program in Embedded Systems is primarily intended to train designers, and will not be integrated into a PhD-trajectory. The master thesis can be the basis for a subsequent PhD phase where it can be re-used as part of the PhD thesis (preferably a ‘promotie op proefontwerp’), provided it meets the quality standards for PhD theses.

B.7.11 Expected starting date

This master program starts at September 1, 2003 as variant of the M.Sc. program in Electrical Engineering and Information Technology, and at the same time as a variant of the M.Sc. program Computer Science and Engineering. It is intended to start it as an independent master program as soon as possible, preferably in September 2004.

B.8 Undergraduate and Graduate Curriculum on Embedded Systems and Micro-Robotics at the University of Oldenburg

The specialisation offers a professional education for engineers, who are to be entrusted with the development of embedded systems as well as different micro-systems and micro-robots. It enables them to conceive and develop the mentioned systems in a professional and engineer-like way and to impart prevailing basics and the principles of the methods, the tools and the design. In addition, the Diploma studies shall prepare the students for scientific work. The department of computer science structures its training in two study phases. A study of six semesters finishes with a profession-qualifying degree, the Bachelor of Computer Science. Alternatively, the diploma course of studies leads to the degree of a Diplom Informatiker/in. Continuously all courses are offered in form of modules with 4 semester periods per week. The following tables describe the provided courses concerning the *Embedded Systems and Micro-Robotics* specialisation.

First Year Semester 1	Algorithms and Data Structures I and II	Programming Course	Technical Computer Science I	Discrete Structures	Mathematics for Computer Scientist I and II
First Year Semester 2		Software Engineering	Technical Computer Science II	Theoretical Computer Science I	
Second Year Semester 3	Practical Computer Science	Software Project including Pro-seminar	Embedded Systems I	Theoretical Computer Science II	Differential Equations
Second Year Semester 4	Embedded Systems II		Practical Course Technical Computer Science	Soft Skills	Basics of Electrical Engineering
Third Year Semester 5	Computer Science and Society	Software System Engineering	Control Engineering	Choice from Practical or Applied Computer Science	Micro-System Engineering and Micro Robotics
Third Year Semester 6	Individual Project including Presentation and Graduation work usually within the area of Embedded Systems and Micro-Robotics			Real Time Operating Systems or Micro Robotics II	Digital Signal Processing or Micro Robotics II

Table 1: Curriculum: BSc in Computer Science with Embedded Systems and Micro-Robotics Specialisation

First Year Semester 1	Algorithms and Data Structures I and II	Programming Course	Technical Computer Science I	Discrete Structures	Mathematics for Computer Scientist I and II
First Year Semester 2		Software Engineering	Technical Computer Science II	Theoretical Computer Science I	
Second Year Semester 3	Practical Computer Science	Software Project including Pro-seminar	Embedded Systems I	Theoretical Computer Science II	Differential Equations
Second Year Semester 4	Embedded Systems II		Practical Course Technical Computer Science	Soft Skills	Basics of Electrical Engineering
Third Year Semester 5	Computer Science and Society	Software System Engineering	Control Engineering	Choice 6 from Practical or Applied Computer Science	Micro-System Engineering and Micro Robotics
Third Year Semester 6	Individual Project including Presentation and Graduation work			Choice 8	Digital Signal Processing or Micro Robotics II
Fourth Year Semester 7	Project Group including Seminar Lecture and Final Report usually within the area of Embedded Systems and Micro-Robotics		Subject Choice 1	Subject Choice 2	Choice 9
Fourth Year Semester 8			Subject Choice 3	Subject Choice 4	Choice 10
Fifth Year Semester 9	Thesis (Diploma) usually within the area of Embedded Systems and Micro-Robotics				

Table 2: Curriculum: Diploma in Computer Science with Embedded Systems and Micro-Robotics Specialisation

In order to achieve the learning targets, basic knowledge in electrical engineering, control engineering, micro-system engineering and physics is increasingly imparted within the choice area of the basic study period. During the choice modules 1-7, the variety of courses comprehends for the BSc-studies and the Diploma studies, firstly an introduction to Architecture and Design Methodology of Embedded Systems as well as the two basis modules in Electrical Engineering, Micro-System Engineering and Micro-Robotic lasting for two trimesters. The module Embedded Control Engineering introduces the application area embedded controllers, whereas the module Digital Signal Processing gives the basics for embedded communication systems. The module Software System Engineering prepares for the control over and development of complex software systems. The module Micro-Robotics II builds on the basic module Micro-System-Engineering and Micro-Robotics and offers to specialise in various subjects within Micro-Robotic such as Micro-Actuator Engineering, Micro-Sensor Engineering, Controlling of Robots etc. Within the module Fuzzy Regulation and Neural Networks, behaviour- and knowledge-based methods of controlling and data-processing are being discussed, which are important for embedded systems as well as for micro-robotics.

In the BSc course of studies the students usually have to choose their topic for the individual projects in the sixth semester from within the area of Embedded Systems and Micro-Robotics, in order to gain pertinent practical skills in this area. The choice modules 8 and 9 are assigned to the courses of Real Time Operating Systems or Micro-Robotics II as well as Signal Processing, in order to gain a knowledge as broad as possible with the degree of a bachelor.

In the Diploma course of studies, the students can also choose a subject for the individual project to broaden the knowledge and skills outside the Embedded Systems and Micro-Robotic specialisation. The project group and the subject of the thesis are usually to choose from the area Embedded Systems and Micro-Robotic. The choice modules 8 to 10 are to be selected from the modules listed in table 3.

Choice 8 one module from	- Formal Methods of Embedded Systems - Theory of the Real Time Systems - Continued Courses in Mathematics
Choice 9 and 10 two modules from	- Fuzzy Regulation and Neural Nets - Digital Signal Processing - Micro-Robotics II - Measurement Engineering - Control Engineering II - Sensor Technologies - Complex Systems - Mechanics and Thermodynamics - Robotics - Pilot-Assistance -Systems- Real Time Operation Systems - Distributed Systems - Verification and Test of Embedded Control Systems - Hybrids Systems - Low Power System Design
Area choice 4 Technical Computer Science one module from	- Design of Integrated Circuits - Multiprocessor Systems - Fuzzy Regulation and Neural Nets

Table 3: Choice modules from the Embedded Systems and Micro-Robotics Specialisation in the diploma curriculum

Within project groups and intern-ships, students should develop concrete applications involving their tutors, who work in user companies.

The modules listed in table 3 do not show a completed list of options – the current module announcements provide information about which additional modules will be offered in the current semester.

Technical Computer Science I and II

- Introduction
- Design Area
- Design of Instruction-set Architectures
- System-Level Design
- Basic Concepts at the Level Transistors
- Design of Combinatorial Circuits
- Design of Sequential Circuits
- CPU Design I: Sequential Program Execution
- CPU Design II: Pipelining
- Caches
- DSP Design
- HW-SW Co-Design
- Application I: Safety Critical Systems
- Application II: Digital Signal Processing

Intern-ship Technical Computer Science

- Design of Digital Circuits
- CPU Design
- Design of a Control System

Embedded Systems I and II

- Characteristics of Typical Application Domains
- Process Model
- Introduction to Modelling
- Modelling Languages and Tools
- Target Platforms
- Architecture of Safety Critical Systems
- RT-Operating Systems
- Safety Analysis
- Validation Techniques
- Design Techniques
- Implementation Techniques
- Integration & Testing

Micro System Engineering and Micro-Robotics

- Micro System Engineering and Micro-Robotics: Ideas, Problems, Activities
- Applications of Micro-Systems
- Techniques of the MST: Micro Techniques, System Techniques, Materials and Effects

- Micro Mechanics: Silicon-Process, the LIGA-Process
- Micro Actuators: Principles, Implementation
- Micro Sensors: Principles, Implementation
- Design and Simulation in the MST
- Test and Diagnosis of Micro-Systems
- Data Processing in Micro-Systems
- Introduction to Micro-Robotics
- Micro and Nano-Positioning Systems
- Micro and Nano-Manipulators
- Problems with the handling Micro and Nano-Systems

Fuzzy Regulation and Neural Nets

- Introduction to Robotics and Automation
- Introduction to Fuzzy and Neuro-Systems
- Basics of the Fuzzy Logic
- Fuzzy Logic of Rule-Based Systems
- Simple Models of Neural Nets
- Learning Algorithms for Neural Nets
- Multilevel Nets and Back-propagation
- Associative Memories and Stochastic Nets
- Self-Organising and Hybrid Nets
- Design of Fuzzy Regulation Systems
- Transmission Characteristic of Fuzzy Automatic Regulators
- Practical Use of the Fuzzy Logics
- Design of Neuro Control Systems
- Practical Use of Neural Nets
- Fuzzy + Neuro: Basics and Applications
- Hard and Software Tools for Realization of Fuzzy and Neuro Systems