Teaching Skills and Concepts for Embedded Systems Design

Peter Bertels
peter.bertels@ugent.be

Michiel D'Haene
michiel.dhaene@ugent.be

Tom Degryse
tom.degryse@ugent.be

Dirk Stroobandt
dirk.stroobandt@ugent.be

Department of Electronics and Information Systems
Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

ABSTRACT

Smart devices are omnipresent today and the design of these embedded systems requires a multidisciplinary approach. It is important that students in electrical engineering and computer science learn these different aspects of embedded systems design. Our course on *Complex Systems Design Methodology* presents an overview of embedded systems design with a strong focus on the main concepts, preparing the students for more detailed follow-up courses on specific topics.

Imparting the theoretical concepts to the students is not sufficient, however. Hands-on sessions are indispensable for the students to acquire the necessary skills. In this article we present our approach for these hands-on sessions, which is to pose relatively small problems in separate sessions, each focusing on a single design aspect. Five years after the introduction of this new course at Ghent University, we can conclude that students not only like this course, but that their design skills have also improved by our new, aspect-focused, approach.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education, information systems education*

General Terms

Design, Experimentation, Human Factors, Languages

Keywords

Embedded Systems Education, Specification, Transaction-level modelling, Architecture Exploration, Optimisation

1. INTRODUCTION

Our world is constantly changing. To a large extent, this is a consequence of the technological revolution that is taking place at an ever increasing pace. It has a significant impact on our social lives, as exemplified by the exponential increase in the number of cell phone users over the last decade.

The beginning of last century marked the industrial revolution. By the end of the same century, we were already amidst the technological revolution, bringing us the era of *smart devices*. The tremendous increase in computational power available in appliances was brought to us mainly by the advances in digital design technology. Surfing on Moore's Law, billions of chips could be produced at low cost and with ever increasing functional capabilities. Yet, the exponential scaling also made the design of these chips much more difficult, requiring a step up into the hierarchy of abstraction levels in order for designers to be able to cope with the increased complexity. At the same time, integration of software and hardware aspects has introduced embedded systems.

Having become very relevant in an industrial context, embedded systems design cannot be left without a proper education of electronics and/or computer science engineers in this new domain. As acknowledged by the ARTIST Guidelines for a Graduate Curriculum on Embedded Software and Systems [1], embedded systems education should be multidisciplinary and contain aspects of control and signal processing, computing theory, real-time processing, distributed systems, optimisation and evaluation, and systems architecture and engineering. In our view, this list should be extended with hardware design for embedded systems as often processing elements have to be augmented with specific hardware blocks performing dedicated functions.

As is apparent in multidisciplinary curricula, it is difficult for students to grasp the big picture from the separate pieces that are provided in courses within an embedded systems curriculum. Therefore, we claim it is important to provide students with an overview course that introduces the main concepts of embedded systems design and combines them to a complete picture of what embedded systems design entails. This course on *Complex Systems Design Methodology* should be considered a backpack enabling students to further extend their knowledge by taking more detailed courses on

the various subjects and put them in their backpack. This is the approach we have taken in designing an embedded systems curriculum at Ghent University in Belgium.

Providing the students with the backpack (course) is not sufficient. They also need to know how to wear the backpack and how to use the tools present inside the backpack. Putting the courses into practice is therefore crucial. This has also been acknowledged by the ARTIST Guidelines [1]. However, in this article, we will elaborate on how this can be done in practice. In contrast to the overview focus in the theoretical sessions, we do not advocate one big project where students have to actually design a complete system as this tends to make them get lost into the details. We rather focus on the main concepts needed for a well-designed embedded system and on the skills students have to acquire for this. Therefore, we present relatively small problems in separate sessions, each focusing on one aspect. We also try to state problems from their own daily environment so that at least the problem is not new and they can focus on the concepts and skills that bring them to the right solution. Once the students have finished these exercises, they are ready to learn how to combine these practical experiences into a big project, which is, in our university, possible in a separate hardware design project course.

In the remainder of this article, we will explain the main intricacies of embedded system design in Section 2, showing the importance of educating engineers in this domain. This section also shows how our course fits in the Ghent University curriculum. In Section 3, we motivate our choice for writing our specific course material for this course that combines the best ideas from several books on the topic [5, 9, 11]. We then discuss the main concepts and skills for embedded systems design that we focus on in the course and we show how these are illustrated with practical exercises in the hands-on sessions (Section 4). Section 5 explains our way of interacting with the students, the yearly student evaluation and our continuous efforts to improve the course and to keep up with state-of-the-art research in the domain. Finally, in Section 6, we conclude with some qualitative results which show that the Ghent University approach towards embedded systems education has been very successful.

2. WHY TEACHING EMBEDDED SYSTEMS DESIGN?

2.1 Embedded systems design

The technological revolution is mainly driven by Moore's law, originally stating that the number of transistors per chip is doubling every 24 months. In the '60s, this law was merely an empirical observation made by Intel employee Gordon Moore but it quickly began driving the EDA (Electronic Design Automation) industry, thus turning into a self-fulfilling prophecy. Moore's law has enabled ever more powerful systems on the same die area. The doubling of the number of transistors per unit area every technology generation allowed designers to put more functionality on a single chip, even to the amount that it is no longer manageable. This leads to the infamous *design gap* as the number of designers or the time needed for a large design can no longer keep up with Moore's law. In order to further improve productivity, the only solution is to reuse big existing designs and combine these. This is called IP (Intellectual Property) Reuse. Where this combination of "chips" to a complete system used to be done at the board level, it is now possible at the chip level, leading to a System-on-Chip (SoC) design methodology. In such a SoC, scheduling and arbitration are becoming more important, bringing software issues to the hardware designer's world.

With the increase in compute power, also the applications are getting more demanding. Ubiquitous computing is becoming the name of the game and people are starting to expect access to high quality multimedia data, especially video, everywhere. This puts an enormous pressure on multimedia hardware systems and requires the use of specialised architectures and the exploitation of massive parallelism. At the same time, however, applications demand a high amount of flexibility from the devices as they are rapidly changing and frequent updates need to be possible in the devices that have to run the applications. This is not possible with Application Specific Integrated Circuits (ASICs) and requires a processor architecture. Current systems therefore generally consist of hardware acceleration blocks that co-exist with a software oriented processor environment. Hardware design has thus turned into hardware/software co-design, with a lot of emphasis on the new bottleneck: the communication between the processor and dedicated hardware.

While embedded systems design in Europe has mainly been driven from a software and real-time perspective, as exemplified by the successful European ARTIST Network, the original need for more abstract levels of design and the introduction of system design on a single chip have risen out of the hardware world and are mainly a result of the hardware design gap. Using our background as a hardware design group, we tend to look at embedded systems design from this hardware perspective. This provides us with a rather unique view on embedded systems education. The hardware view indeed enables us to look at embedded systems as what is possible in the design of the actual system instead of the traditional demand-driven approach where a designer has to try and implement something that the client wishes for. In this respect, system designers should, in our opinion, have a much better notion of the actual possibilities in hardware so they can make more educated guesses on possible architecture exploration results upfront, hence reducing the time-to-market.

2.2 Embedded systems education

At Ghent University, we recently adopted a new educational (Bachelor/Master) structure, with a separate option within the Master of Computer Science Engineering dedicated to *Embedded Systems*. This new option is specifically targeted at the hardware/software interface. This was intended to address the need for a more modern education at the forefront of the technical evolutions. A key course in this is the course on *Complex Systems Design Methodology*. This course mainly builds on *Programming* and *Digital Electronics*, both taught in the Bachelor curriculum. It is drafted to be the basic complex (embedded) systems design course, providing a broad overview of the systems design methodology (Figure 1) and the general concepts involved in a complex design that consists of several components working together. Within the overview of this methodology, the

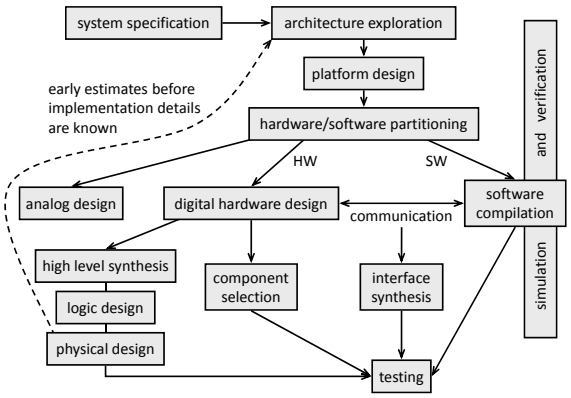

Figure 1: The design flow for complex systems.

course provides many hooks for follow-up courses that go much deeper into the details of specific aspects of system design, such as *Analog Design*, *Sensors and Actuators*, *Hardware/Software Co-Design* (ASICs and FPGAs), *Operating Systems* including scheduling and real-time operating systems, *Advanced Computer Architectures*, etc. Students can choose several of these detailed courses within their curriculum that fit into this complex systems design framework.

Our course on *Complex Systems Design Methodology* follows our own version of the embedded systems design flow described in Figure 1. It focuses on the front end design steps of specification, architecture exploration and system partitioning. Fully in line with the backpack approach, the remaining design steps are only briefly touched upon and presented as hooks for the mentioned follow-up courses. As a concrete example of these hooks, we would like to mention a cluster of four blocks in Figure 1: digital hardware design, high-level synthesis, logic design and physical design. These aspects of the methodology are covered in detail at Ghent University in the *Hardware/Software Co-Design* course.

In the second part of the course, we focus on some specific optimisation aspects for embedded systems design. We first emphasize optimisations of data locality to optimise memory structures and data transfers. A chapter is also devoted to early performance estimates (of speed, area, power, cost, etc.) that are needed for a good architecture exploration. In line with the current fact that interconnect dominates delay and area figures in ASICs as well as FPGAs, we address performance estimates based on wire length prediction to enhance the architecture exploration step. Because interfaces between subcomponents in a larger system are not part of any other course, this important aspect of embedded systems design is also addressed in our course.

As stated in the introduction, the focus of the course is on the basic concepts and skills needed for embedded systems design. This already starts with the *system specification*, where the importance of a formal functional specification is addressed, together with the need to also describe non-functional design aspects to clearly formalise the requested performance. Formal ways of describing concurrency, state machines, communicating processes lay the basis for the following design steps.

One of the important things we feel students should learn is that there is no single step-by-step approach that works for all embedded systems. Designing is always a matter of making *trade-offs*. Depending on the problem at hand, the desired performance features and the available options in the design space, a completely different solution might be suggested for two designs that are intrinsically very similar. This may seem *natural* to people designing systems, it makes teaching system design very difficult. The course therefore spends ample time on the most important performance measures: currently power, cost, time-to-market, latency, bandwidth, and area are the major ones. Not only should students know why these are important, also the impact of current technologies on these performance measures is important.

One of the critical issues is the design level at which trade-offs are made. In the first design stages, the design description is very abstract. Yet, one immediately is confronted with very important design decisions on the architecture of the system. In this *architecture exploration* step, not a lot is known about the details of the final implementation as the idea is exactly to abstract most of this away. This leaves a lot of implementation choices for later but it also means it is hard to say something about the relative performance of the solutions to choose from. Therefore, very early high-level performance estimations are paramount. The estimates at this level will not be very accurate but they should give a good relative appreciation of the solutions in order to retain the right architectures.

After (and sometimes during) architecture exploration, one also has to decide which parts of the problem will be handled by software and which parts should be taken care of by a digital or an analog hardware component. System partitioning again induces a very important trade-off, that of flexibility versus computing strength (time needed to perform a certain task). The course also addresses the *interfaces* needed for connecting the different system components.

3. COURSE MATERIAL

There are several good books on embedded systems design in general and on the aforementioned specific concepts and skills in particular [5, 9, 11]. However, these books all advocate a slightly different vision on design. Therefore, we have opted for writing our own course material starting from capita selecta of these books. This way, the course material for *Complex Systems Design Methodology* perfectly fits in the Ghent University curriculum with its specific stress on concepts and skills, while meeting the highest international standards.

The second chapter of our course textbook, *System specification*, illustrates how we advocate a different emphasis than similar courses on the subject. Many books introduce the concepts of a model of computation that lies at the basis of embedded systems specification and modelling, but most of these books do not really make a clear distinction between the models of computation and the languages that implement and use these models. We believe that in teaching embedded systems design it is of paramount importance to

focus on the concepts first, i.e., the models of computation, and only then show how different languages use (combinations of) these models of computation to describe systems. In fact, only a few models of computation exist or are relevant to embedded systems design.

Apart from differential equations, suitable for analog implementations, and the sequential programming model students are mostly familiar with, we clearly describe discrete event based models, finite state machines, and data flow models. These three models form the basis of embedded systems implementations as discrete event based models capture the concept of synchronicity in hardware, finite state machines are especially suited for control-based implementations, and data flow models represent the basic data oriented approach.

Once the students grasp the essential ingredients of these models, they can better understand the basic ideas behind the languages that are built around these models of computation (such as Statecharts, Kahn process networks, etc.) Of course, the basic models of computation can only be represented in a certain language but we endorse the students to abstract away the syntax of these languages and focus on the concepts. We do this by providing a single running example of an elevator control throughout the second chapter.

The use of our own course textbook also enables us to incorporate current research in our Embedded Systems group in *Complex Systems Design Methodology*. This approach is taken in chapters 4 and 7 which cover memory optimisation and early estimations in the design process.

4. HANDS-ON SESSSIONS

When teaching a course on complex systems design, hands-on sessions are very important for students to learn the intricacies of systems design. Hence, we specifically paid attention to setting up relevant practical exercise sessions for the students. These practical sessions first elaborate on the basic parts of the embedded systems design flow presented in Figure 1: specification and architecture exploration. The system partitioning step has a major impact on the design of the communication infrastructure. These aspects are covered in a session on transaction-level modelling. Finally, as the second part of the course details important performance aspects in embedded systems design, a last set of practical sessions introduces the students to optimisations of data locality.

Although the course focuses on complex system design, we deliberately keep the practical examples small and simple enough to be able to explain the basic concepts without losing ourselves in the complexity of the problem. At the same time, we explain to the students how the different concepts help them in designing more complex systems so that they understand why the concepts are important.

4.1 System specification

The first step in the design methodology of embedded systems is the specification of the system. This specification describes the desired behaviour of the system as a black box with inputs and outputs. At this stage in the design flow the implementation details of the proposed system are not important yet. The specification merely defines exactly how the system should respond to its inputs. It needs to be clear, unambiguous and readable for humans as well as for computers. Specification in this context is purely functional. Although non-functional properties are equally important, commonly used system specification languages have no means to describe them. Therefore, we deal with non-functional properties during architecture exploration in Section 4.3.

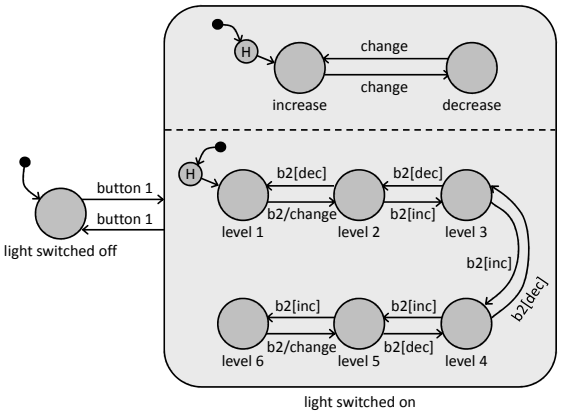

Figure 2: First version of the intelligent switch.

In this course, several specification languages are discussed. For the hands-on exercises we have selected only two of them: StateCharts [7] and Petri nets [10]. StateCharts are chosen because they are an extension of the basic concept of finite automata that was already introduced in a preceding course on *Digital Electronics*. Petri nets are handled because they express important and often critical properties of dynamic systems: mutual exclusivity and concurrency. UML diagrams are also discussed in the theory sessions but not included in the practical sessions because they are already extensively treated in software courses.

We want to focus on several interesting aspects of StateCharts and Petri nets separately. To take all these aspects into consideration without losing the overview or without overwhelming the students, we have prepared several small pencil and paper exercises. The students learn to make an exact, unambiguous and standardised specification out of a vague initial textual system description. Attention is also given to interpreting the behaviour of systems by means of a formal specification.

4.1.1 StateCharts

As indicated above StateCharts are an extension of finite automata which are introduced in a preceding course. One of the most important new aspects is hierarchy which enables large complex systems to be expressed comprehensibly. In StateCharts, hierarchy is expressed in AND and OR super states. AND super states express that the system has several concurrently active states, one for each AND part. OR super states indicate that only one of several states is active in the system.

In the first exercise we specify an intelligent light switch with

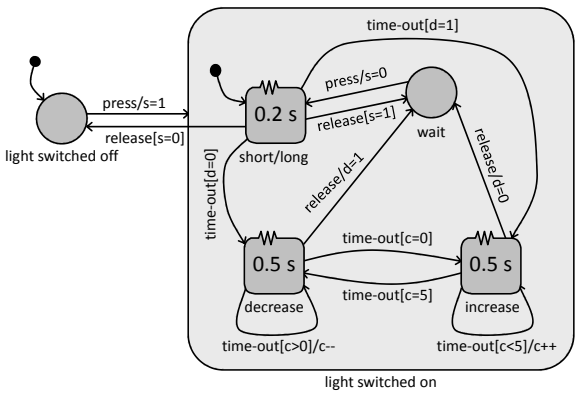

Figure 3: Second version of the intelligent switch.

two buttons: an on/off button and a dimmer button. Our system, depicted in Figure 2, has 6 brightness levels which can be accessed by pushing the dimmer button several times. The intelligent switch remembers the brightness level of the last time the light was on. The dimmer button increases the level from 1 to 6. When the maximum is reached the level starts decreasing again. Therefore the direction (increasing/decreasing) has to be modelled as an explicit state. The system intrinsically has 24 states in total (6 x 2 x 2). Nevertheless, StateCharts can express this in a smaller and comprehensible diagram using hierarchical states.

The second assignment builds on the first intelligent switch to make an even more intelligent switch with just one button. Holding the button for more than half a second increases or decreases the brightness level. Pushing the button for a short time (less than 0.2 seconds) toggles the light on or off. This additional time behaviour can be represented in StateCharts by using timeout blocks as shown in Figure 3.

To further practise the newly learned concepts we present a few more problems such as an underground car park with sensors to count the incoming and outgoing cars and a traffic light system with a pedestrian button.

4.1.2 Petri nets

Petri nets originate from the Ph.D. thesis of Carl Adam Petri [10] and have been augmented to several new types of Petri nets. We discuss three of these types in the session: basic condition event nets, place transition nets and coloured Petri nets. In condition event nets the conditions are either true or false, represented by the presence or absence of a token. Place transition nets take this concept to a higher level by allowing multiple tokens in one *place*. For these nets also transitions can require or produce multiple tokens. In place transition nets, all tokens are equivalent, which is not the case in coloured Petri nets where each individual token has its own identity.

Petri nets can express fundamental relations between operations in the system such as concurrency or exclusivity, illustrated in our examples. A detailed description of all examples is beyond the scope of this article. In short we have an exercise on a call centre where each operator has a specialisation to handle certain calls. This example illustrates queuing where callers have to wait until a suitable operator becomes available. Another exercise is about an airport where multiple runways cross each other. A Petri net is used to express the mutually exclusive use of these runways.

In addition to exercises on the transformation of vague descriptions into a formal specification, Petri nets are particularly suited for exercises on formal analysis and even proof of certain system properties. Examples of these exercises include a satellite communication system where the students prove that no message can get lost and a memory model guaranteeing exclusive memory access by a CPU and a DMA unit.

4.2 Transaction-level modelling

The growing complexity of embedded systems (Section 2.1) has created the need for high level system modelling and simulation. Transaction-level modelling was proposed to raise the abstraction level and keep the design complexity manageable. A transaction-level model separates the details of communication among modules from the details of the implementation of functional units and of the communication architecture. During the design flow of Figure 1 transaction-level modelling is used to abstract specific implementations of the communication in the system in generic communication channels. This enables a general formulation and modelling of the system. Several implementations of specific communication channels can be plugged in for simulation of their impact on the final system.

An important model for embedded systems is Kahn Process Networks [8]. A Kahn process network is a network of processing units connected by FIFO communication channels. Together these processes incrementally transform an infinite flow of data. Processors can work sequentially or in parallel. This model of computation is commonly used for streaming applications and for embedded systems. Kahn process networks fit nicely in the more general concept of transaction-level modelling with the restriction that FIFO channels are used.

SystemC [6] is a set of libraries on top of C++ developed for describing and simulating complex systems. It provides flexible templates to specify functional units (modules) whose ports can be connected through communication channels. On the other hand, SystemC also has a built-in simulation kernel. These features make SystemC very convenient to put transaction-level modelling in practice.

In this session we will therefore use SystemC to build and simulate a reasonably complex Kahn process network. Although we acknowledge that an illustrative example embedded system would be good for introducing transaction-level modelling, our experiences over the last five years have shown that students can acquire the relevant skills better when the example is closer to their everyday lives. Then they can focus more on the concepts instead of on the problem. For this reason, we use a model for the administrative procedures in our Faculty of Engineering in the lab session. In this example, forms enter the system and are processed by several persons. The processing of forms often leads to

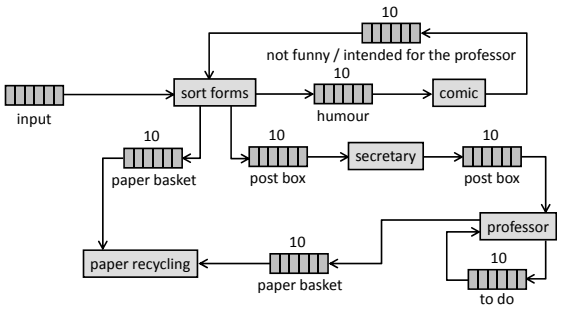

Figure 4: Transaction-level modelling of the administrative procedures in our Faculty of Engineering

the creation of new forms. A feedback loop allows additional processing. Eventually all forms disappear in the archive or in the paper basket.

The session consists of two parts. In the first part the students start from a given SystemC model describing the connections between modules in the system. The actual communication details of these modules are left to the students. The emphasis is on reading and writing to the FIFO channels in a blocking or a non-blocking way.

During the second part of the exercise the students gain insight in the system. They learn which components and which interactions are critical for the behaviour of the system as a whole. The students make the conversion from the ideal world of conceptual Kahn process networks with infinitely large buffers to the real world where true infinity does not exist. All internal buffers are now downsized to their optimal size. Handling the restrictions of real systems is an important skill which we wanted to teach in this session.

The learning climax is reached when after adding a few extra forms the system suddenly blocks: a deadlock has occurred. The concept of a deadlock was already mentioned in preceding courses but this concrete and sudden appearance opens the eyes of most students. After this exercise they will never forget this important concept.

The students have to solve the deadlock by changing the internal implementation of the system components. After studying the properties of the system and verifying the conditions for a deadlock, most students independently find the right solution: assigning priorities to the incoming FIFO queues of one of the components in the system. This way the lab session forms an excellent illustration of the concepts mentioned in the theory sessions.

4.3 Architecture exploration

The design space for the realisation of a complex system is often very large. Different implementations are possible and a lot of these possibilities meet the functional requirements set out in the specification step. The designer then uses the non-functional properties to make a well-founded design choice. This exercise discusses the most important non-functional properties: cost, energy consumption and execution time. Other non-functional properties are handled in the theory sessions.

The designer has to find a suitable implementation for the application at hand in the sea of possibilities that is the design space. This is the task of the architecture exploration. Eventually, this exploration leads to a Pareto curve with all Pareto optimal solutions.

The goal of this session is threefold. First, we want to illustrate that the design space is very large, even for a rather simple application. Second, the concept of Pareto optimality and Pareto curves has to be made explicit in this session. Finally we want to teach the students how to find their way in the vast design space without trying every possible implementation, which is of course impossible. We do this by means of early back of the envelope estimations of the non-functional properties of several architectures.

Several tools have been developed for architecture exploration, but in our experience no tool exists which can perform the entire exploration on any example. Therefore we have chosen to do the estimations by hand with a pencil and paper approach and the exploration in Microsoft Excel. This way we can impart the important insights without losing ourselves in the nasty details and singularities of specific tools. Another advantage of not using a tool is that we do not end up with a lot of figures that came *automagically* out of a tool. The lack of accuracy of our manual estimations is not deemed as a problem as gaining insight is more important.

Because the complexity is a crucial aspect in this exercise we use a real world example: a digital camera with a JPEG encoder in its core. This encoder consists of four sequential blocks: RGB to YUV conversion, a discrete cosine transformation (DCT), a quantisation step and a Huffman encoder.

After a short introduction, we will make estimates for software implementations on an inherently sequential microprocessor and parallel hardware implementations of the different blocks of the JPEG encoder. Several architectures are possible for each of these blocks: a fully sequential implementation, massive parallelism, a pipelined architecture ... Various implementations of the RGB to YUV conversion are extensively covered in a class discussion. For a software estimation, the number of processor instructions is counted and weighted by the execution time on an ARM embedded processor. We also make a cycle accurate estimation of the execution time and hardware area for three different hardware implementations: a sequential implementation, an implementation with maximal parallelism and a pipelined approach. This example prepares the students for estimating the DCT by themselves. This leads to a hardware schedule that fits the discrete cosine transform of one row of a block of 8 by 8 pixels in 7 clock cycles (Figure 5).

In the second part of this session the estimates for separate subsystems are combined to form an estimate for the digital camera as a whole. A new design space arises at a higher level: all parts could be implemented in specialised hardware, everything can be done in software on a microprocessor, ..., several combinations are possible.

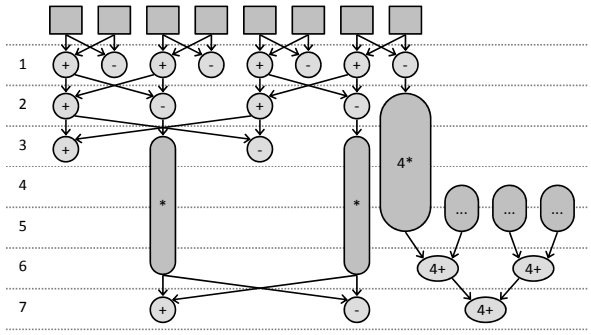

Figure 5: Cycle-accurate estimation for the hardware implementation of the DCT for an array of 8 pixels

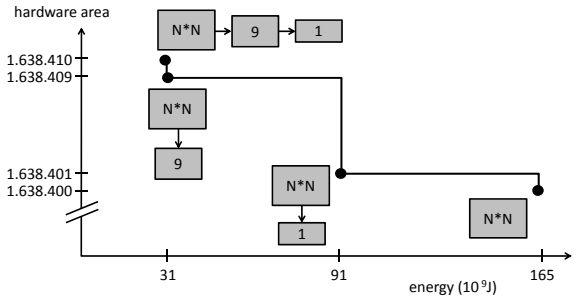

Figure 6: Evaluation of several custom memory hierarchies on a Pareto curve

For the exploration the students get a framework in Microsoft Excel with some basic figures for the cost of several microprocessors and the cost and energy consumption of several hardware implementations. In this framework the students then add estimates for the execution time. For the combined estimates for the camera as a whole, the students have to also consider the communication cost between different software or hardware components and the memory. Based on this the students draw Pareto curves, neglecting non-optimal solutions.

In the end the students must make a motivated choice for the final architecture. The students are divided in groups with a different set of non-functional properties for each group: a professional camera, a low cost camera, several resolution levels ... This way they also learn that the trade-offs to be made really differ significantly from one specification to another.

4.4 Optimisation in an embedded context

Most embedded systems include small computationally intensive code parts. Such hot code plays an important role in the optimisation of the system: small improvements in this code can make a huge difference for the overall performance. Today optimisation skills have become indispensable for engineering students in computer science or electronics.

Although software optimisation in general is already present in the curriculum of our students, we feel the need for specific optimisations aimed at embedded systems. Specific properties of embedded systems that should be taken into account are: limited resources (of memory, e.g.), stringent timing constraints (soft real-time or even hard real-time), power limits, etc.

Especially optimisations that improve the data locality in programs – such as the well known loop transformations – proved to be lacking in other courses. By improving the data locality, the program is prepared for the introduction of a custom memory hierarchy based on the use of scratch pad memory. This way also the power consumption of the system can significantly be reduced [4].

Typical in embedded systems is also the parallel nature of the underlying hardware platform. Therefore we train our students in exploring parallelism in the program: coarse grain parallelism across functions as well as fine grain parallelism in a single function or even loop body.

All optimisation techniques and concepts discussed in these sessions – from loop transformations over defining a custom memory hierarchy up to parallelism detection – are introduced with a short academic example. Afterwards the students can try for themselves on a real world example. For the optimisation labs we have chosen a medical image processing application as a running example. The cavity detector [3] is a relatively simple application but it nevertheless offers lots of optimisation possibilities which make it ideal for our hands-on sessions.

Figure 6 shows four different custom memory hierarchies on a Pareto curve. It helps our students not only to be creative in finding possible custom memory hierarchies for the cavity detector application, but also to compare them and to find optimal solutions. It should be noted that our own Hardware and Embedded Systems research group is very experienced in research on such memory optimisations. Hence, in these sessions of *Complex Systems Design Methodology* we can link research and education. The students use, among other tools, our SLO tool for suggesting locality optimisations [2] which enables them to find memory hierarchies which reduce the power consumption even more.

5. DIDACTICAL APPROACH

5.1 Key success factors

The course on *Complex Systems Design Methodology* was first taught in 2004-2005. Since then students evaluate this course as very good. We believe that this is due to our personal attention on top of the intrinsic qualities of the course. Key factors are enthusiasm and motivation, interactivity, evaluation and real world examples. In this section we will elaborate on these items.

Interactivity Interaction between teachers and students improves their commitment. In the first sessions on specification languages we strongly encourage this interaction. Students are invited to bring their own solutions on the blackboard for a class discussion. The computer sessions are obviously more individual but even in these sessions we try to encourage interaction and cooperation.

Evaluation The Faculty of Engineering at Ghent University has set up a biyearly student evaluation for each course, based on a standard questionnaire. For *Complex Systems Design Methodology* we extended this evaluation process with our own yearly survey with more specific questions about each chapter of the course material and each hands-on session. The student evaluation has been very important for the evolution this course has gone through since 2004-2005. Based on the results of the survey we could improve the course to its current quality: in last year's evaluation the course was given an average overall score of 91%.

Real world examples For the hands-on sessions we have chosen to use small examples each focusing on a single concept. In contrast to a project based approach we cannot afford the time for a large introduction of each example. Using practical examples close to the daily environment the students are used to, a lengthy introduction is redundant. Furthermore, real world examples keep the students focused and motivated.

Enthusiasm and motivation The last key factors are enthusiasm and motivation of the teachers. It is clear that students automatically appreciate a learning experience more if it is brought with enthusiasm. It is important for a student to feel that the teacher really believes in what he or she is teaching and in the way he or she is teaching it.

5.2 State-of-the-art technology

Obviously, no embedded systems design course is complete as the extremely fast pace of technological evolution requires the course to remain a work in progress. Each year, new technical evolutions and new insights into what the design of embedded systems means should be added to the course so that it keeps pace with the state-of-the-art.

Reading relevant papers is the best way to follow recent trends in embedded systems design. Every year we select a few papers which are read by groups of four students. Every group makes a short presentation about the contents of the paper such that all students can learn from each other's papers. This way we can keep the students at the forefront of technology and novel concepts, within the limits imposed by their study times and capabilities. Moreover they acquire the necessary skills for reading scientific papers, which is new to most of our students. In our experience, this didactic approach appeals to most of the students who are really fascinated by the research concepts in the papers.

6. CONCLUSIONS

Given the omnipresence of embedded systems and the continuous lack in industry of a sufficient number of good embedded systems designers, it is clear there is a strong need for well-thought-of embedded systems curricula, especially for computer science and electrical engineering students.

In this article, we have described the approach we took at Ghent University in this matter. It is based on a few golden rules: (i) a good systems designer has to know and understand the basic concepts and skills underneath systems design, (ii) our basic embedded systems design course gives a bird's eyes overview of embedded systems design and provides hooks for many other courses that elaborate on some aspects in a much greater detail, (iii) theory sessions have to be augmented by relevant practical hands-on sessions. We believe it is important that these practical sessions provide a step-by-step learning environment for the students, rather than sending them through the woods on a big project assignment, and (iv) the course material is updated yearly to include new technological evolutions, apply new didactic insights and address student remarks from the course evaluations.

Our practical exercises, described in this article, illustrate this step-by-step approach. However, the quality of the course itself cannot be separated from the way in which the material is presented by the teaching staff. The very positive student evaluations indicate we are on the right track.

The results of our students in follow-up courses is also very good. Our students make their first complete system design in the *Hardware/Software Co-Design* course or in their master's thesis. They prove that they can bring the important skills learned on small and simple examples in *Complex Systems Design Methodology* into practice in one real world design. This convinces us that the course described in this article, which was added to the curriculum in 2004-2005, was definitely an improvement of Ghent University's embedded systems education.

7. REFERENCES

[1] ARTIST network of excellence. Guidelines for a graduate curriculum on embedded software and systems. 2003.

[2] K. Beyls and E. H. D'Hollander. Intermediately executed code is the key to find refactorings that improve temporal data locality. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 373–382, New York, NY, USA, 2006. ACM.

[3] M. Bister, Y. Taeymans, and J. Cornelis. Automatic segmentation of cardiac MR images. *IEEE Journal on Computers in Cardiology*, 1989.

[4] F. Catthoor, E. de Greef, and S. Wuytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic, 1998.

[5] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd. *Surviving the SoC Revolution*. Kluwer Academic, 1999.

[6] T. Grötker, S. Liao, G. Martin, and S. Swan. *System design with SystemC*. Kluwer Academic, 2002.

[7] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[8] G. Kahn. The semantics of a simple language for parallel programming. In *Proceedings of the IFIP Congress 1974*, North-Holland, Amsterdam, 1974.

[9] P. Marwedel. *Embedded System Design*. Springer-Verlag New York, Inc., 2006.

[10] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[11] F. Vahid and T. Givargis. *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, 2002.