Gathering Skills 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 all aspects of the design of such systems. 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 paper 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.

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. Too 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 decades.

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 mainly by the boom in digital design. 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 special 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 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 paper, 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 paper, we will explain the main intricacies of embedded system design in Section 2, showing the importance of educating engineers in this domain. We then discuss the main concepts and skills for embedded systems design that we focus on in Ghent University. In Section 3, we introduce the practical exercises to illustrate these concepts and skills in more detail. Finally, Section 4 explains our way of interacting with the students and our yearly student evaluation of the course.

2. WHY TEACHING EMBEDDED SYSTEMS DESIGN?

2.1 Embedded Systems Design

The technological revolution is mainly driven by Moore's law, stating that the number of transistors per chip is doubling every 18 months. This law originally (in the '60s) was an empirical observation made by IBM 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 been arisen out of the hardware world, mainly 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.

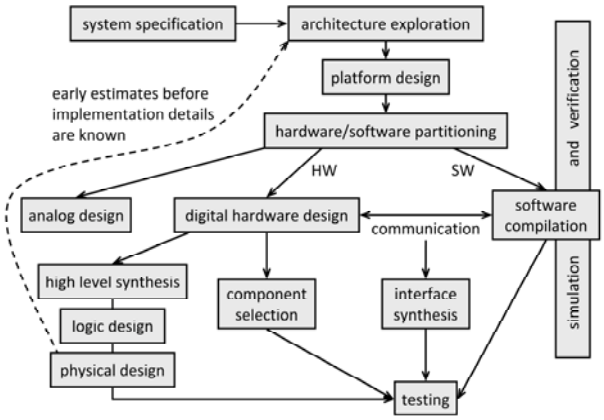

Figure 1: The design methodology for designing complex systems.

Given the strong link between hardware and software design in today's systems, university education should focus on hardware/software co-design and train new electrical engineers and computer scientists in this new cross-disciplinary domain.

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*, which is the main subject of this paper. It is drafted to be the basic complex (embedded) systems design course, providing a broad overview of the systems design methodology (see Figure 1). Within this methodology, many hooks are provided for other courses that go much deeper into the details of several aspects of complex systems design, such as analog design, sensors and actuators, digital hardware design (ASICs and FPGAs), advanced computer architectures, scheduling and RTOS (Real Time Operating Systems), interface design etc. Not all these courses are available at our university (yet), but students can choose several detailed courses within their curriculum that fit into this complex systems design framework.

Our course on *Complex Systems Design Methodology* follows the design flow described in Figure 1 and focuses on the front end design steps of specification, architecture exploration and hardware/software partitioning. Fully in line with the backpack approach, the remaining design steps are only briefly touched upon and presented as hooks for further courses, such as existing courses on hardware/software co-design (that is more focused on digital hardware design), a course on analog design and several courses on software architectures, software design and compilation. However, in the second part of the course, we do focus on some specific optimisation aspects for embedded systems design. We first emphasise early performance estimates (of speed, area, power, cost, etc.) that are needed for a good architecture exploration. A chapter is also devoted to optimisations of data locality (to optimise memory structures and

data transfers). And because interfaces between hardware and software 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 begins 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 embedded systems design methodology. Designing always is 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 hardware component. This *hardware/software partitioning* again induces a very important trade-off, that of flexibility versus computing strength (time needed to perform a certain task).

Imparting the above theoretical concepts to the students is not sufficient, however. When teaching a course on complex systems design, *hands-on sessions* are very important. Indeed, students can only learn the intricacies of complex systems design by hands-on experience. Hence, we specifically paid attention to setting up relevant practical exercise sessions for the students. For each of the main steps in the flow, important concepts and skills are illustrated in practical examples and exercises, as described in Section 3.

3. HANDS-ON SESSIONS

Our practical sessions elaborate on the basic parts of the embedded systems design flow presented in Figure 1: specification and architecture exploration. The hardware/software 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.

3.1 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 evenly important, commonly used system specification languages have no means to describe them. Therefore, we deal with non-functional properties during architecture exploration in Section 3.3.

In this course several specification languages are discussed. For the hands-on exercises we have selected only two of them: StateCharts [4] and Petri nets [6]. 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 loosing 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 system description. Attention is also given to interpreting the behaviour of systems by means of a formal specification.

3.1.1 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 is concurrently in several states, one of each AND part. OR super states indicate that the system is either in one of several states.

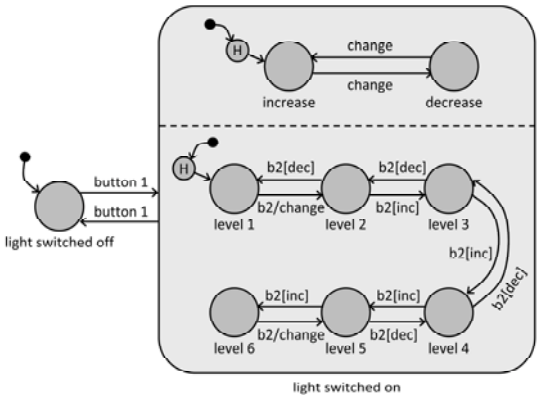

Figure 2: First version of the intelligent switch.

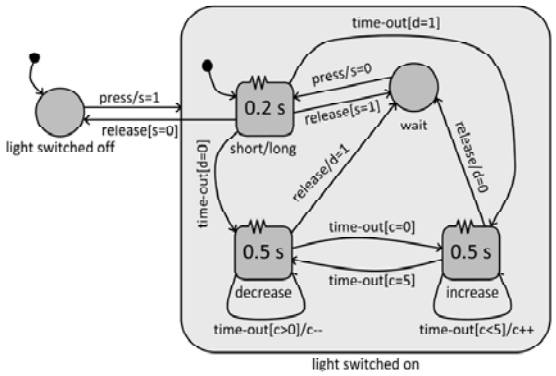

Figure 3: Second version of the intelligent switch.

In the first exercise we specify an intelligent light switch with 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.

3.1.2 Petri Nets

Petri nets originate from the Ph.D. thesis of Carl Adam Petri and have been augmented with several other types. 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 paper. 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.

3.2 Transaction-level modelling

The growing complexity of embedded systems (Section 2.1) has created the need for high level system modelling. 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 or of the communication architecture. The different possible implementations of the communication in the system are abstracted in generic communication channels which enables a general formulation and modelling of the system. Later in the design flow an implementation of a specific communication channel can be plugged in without affecting the overall system.

An important model for embedded systems is Kahn Process Networks [5]. 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 [3] 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.

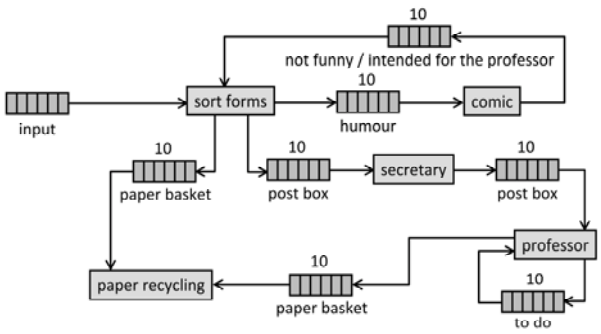

Figure 4: Transaction-level modelling of the administrative procedures in our Faculty of Engineering.

In this session we will therefore use SystemC to build and study a reasonably complex Kahn process network. As an example we have chosen the administrative procedures in our Faculty of Engineering. Forms enter this system and are processed by several persons. The processing of forms often leads to 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 exercises 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 exercises they will never forget this concept.

3.3 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.

For 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… In a class discussion various implementations of the RGB to YUV conversion are estimated, preparing the students for estimating the DCT by themselves.

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.

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…

3.4 Optimisation of hot code

Most embedded systems consist of small computationally intensive code parts. Such hot code plays an important role in 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. Especially optimisations which improve the data locality in programs such as the well known loop transformations proved to be lacking in other courses. We also want to illustrate the use of scratch pad memory and parallelisation.

Several optimisation techniques and concepts are discussed in these sessions: loop transformations, memory hierarchy, low level and higher level parallelism detection. Each concept is introduced with a short academic example. Afterwards the students can try for themselves on a real world example.

We have chosen a medical image processing application for our example: the cavity detector [2]. It is a relatively simple application but it nevertheless offers lots of optimisation possibilities which make it ideal for our hands-on sessions.

These optimisation and parallelisation sessions link education and the research in our own hardware and embedded systems research group.

4. THE EFFECT OF THE MESSAGE LIES IN THE HANDS OF THE MESSENGER

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 syllabus and each hands-on session. Based on the results of this evaluation we could improve the course to its current quality.

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. CONCLUSIONS

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

In this paper, 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 eye overview of embedded systems design and provides hooks for many other courses that elaborate on some aspects in a much greater detail, and

(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.

Our practical exercises, described in this paper, 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 and the recent addition of a follow-up hardware/software co-design course has shown that the students do master the concepts and skills, an indication that the right focus is present in the basic course this paper presents.

6. REFERENCES

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

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

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

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

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

[6] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Rheinisch-Westfälisches Institut für instrumentelle Mathematik an die Universität, Bonn, 1962.