

# Trends and Challenges in Embedded Systems – CoDeVer and HiBu Experiences

Øystein Ra and Torbjørn Strøm  
Department of Computing  
Buskerud College  
N-3601 Kongsberg, NORWAY

## Abstract

Software programmable components interacting with dedicated hardware constitute embedded systems. Such systems are typically application specific systems containing software, hardware and communication channels tailored for a particular task. They are generally part of a larger system and are often candidates for (sub)system-on-a-chip realizations, SOCs; software offering features and flexibility and hardware offering performance. Apart from flexibility and performance, typical metrics include reliability, cost, size, weight, EMC and power constraints. Many such applications in the IT-systems industry have continually changing specifications, and success depends strongly on time-to-market. This calls for a suitable and improved process development model recognizing product life cycles and an efficient and integrated SW/HW-development path. We give a survey of technology trends, future possibilities and current limitations along with a presentation of the “CoDeVer” project and our own efforts within the framework of this project. CoDeVer results will be tested by Norwegian companies ahead of NIK 2002 and the conclusions presented at the conference.

## 1 Introduction

The development of the information society has had a major effect on all industries. To stay competitive, IT-companies mediating such effects have to invest in human talent, and to identify development methodologies, tools and techniques that are capable of efficiently exploiting the ever advancing semiconductor technology. Both the pace of change in this technology and the complexity of the problems to be solved, are increasing. Embedded systems technology will be a key building block in nearly all intelligent products. This is a yet-evolving product class calling on technology from other classes, e.g. software, microprocessors, memory, analog and mixed signal systems, and reprogrammable circuits (FPGAs).

The principal embedded system types comprise:

- *Reactive Systems* reacting continuously to the environment at the speed of the environment;
- *Interactive systems* reacting to the environment at their own speed;
- *Transformational systems* taking a body of input data and transforming it into a body of output data.

Among these our focus will be kept on *Reactive Systems*. Such systems comprise typical target areas as: *Consumer Electronics* (e.g. microwave ovens, digital cameras, compact disk players); *Telecommunications* (e.g. telephone switches, cellular phones); *Automotive* (e.g. engine controllers, anti-lock brake controllers); *Plant control* (e.g. robots, plant monitors, airplane control systems, missile guidance systems).

It is noteworthy that embedded system are often tightly constrained with regard to cost, reliability, size, performance and EMC and *power consumption*, i.e. they must often process data in real time while consuming a minimum of power to extend battery life and/or prevent the necessity of a cooling fan. The physical constraints set **embedded SW** aside from other SW.

The importance of embedded systems is illustrated by the statements "*It is now common knowledge that more than 70% of the development costs for complex systems such as automotive electronics and communication systems are due to software development...*" and "*For many products in the area of consumer electronics the amount of code is doubling every two year*" (A. Sangiovanni-Vincentelli, Univ. of Calif at Berkeley 1999), and by the following figure extracted from <http://www.computerworld.com/>

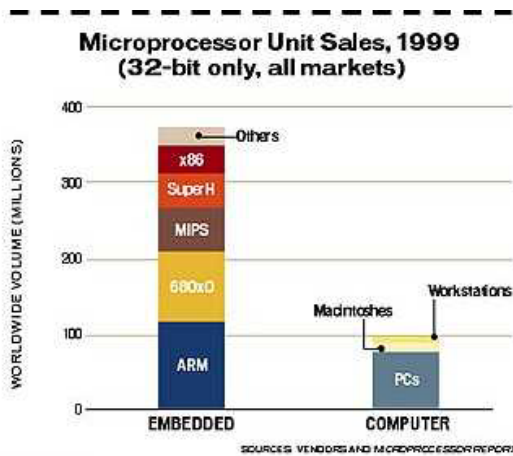


Figure 2: Even the high-performance 32-bit market is dominated by embedded microprocessors, whose unit sales eclipsed the number of processors sold for PCs and workstations last year.

- 94 % (5 billion chips) of the world market is embedded microprocessors vs 6% PC/WS!
- "Processors for PC's, workstations and servers get all the attention, but embedded microprocessors make the world go 'round'"
- "Old microprocessors rarely die, and they hardly ever fade away - they just become embedded";
- 64 bits processors now follow the same trend

In Norway prominent companies undertake development and production of embedded systems, notably within: radio and satellite communication, embedded sensor systems (including automotive applications), video projectors, computer peripherals, interconnect equipment, and distributed process control and surveillance systems. To exploit technological progress such companies will have to master *System-On-Chip (SOC)* techniques, with *SOCs* rapidly becoming the key enabling method of efficient embedded system implementation. *SOCs* most closely resemble *ASICs*, and are evolved most directly from these which they subsume, reduced design cost and higher levels of integration being the principal goal. The difference from *ASICs* in the restricted sense

lies in the typical SOC tendency to maximize reuse of existing "cores" to minimize the the amount of the chip that is newly created. Reused blocks typically encompass analog and high- volume custom cores and CPUs and blocks of software technology. One key challenge is to incept, create and maintain reusable blocks to make them available to SOC designers. The utility of SOCs also depend on validation for reuse-based systems to be easier than for direct designs. The following sections will pursue prospects and current bottlenecks of SOCs and conclude with CoDeVer-HiBu efforts on the promotion of efficient progress in development methods and techniques.

## 2 Technology Trends

Chip technology has undergone a very rapid improvement over several decades. The principal categories of improvement trends are shown in the table below:

TREND	EXAMPLE
Integration Level	Components/chip, Moore's Law
Cost	Cost per function
Speed	Microprocessor clock rate, GHz
Power	Laptop or cell phone battery life
Compactness	Small and lightweight products
Functionality	Non-volatile memory, Imager

In particular, progress in semiconductor technology influences processor capabilities as evidenced by

*"Computer architects are indebted to integrated circuit technology for these gains because it provides fuel for this revolution by shrinking densities and giving us more transistors. CMOS technology has followed Moore's law for 20 years. The real magic that computer architects have managed is not only in using those faster transistors - giving a better clock rate - but also in making good use of the exploding number of transistors. Each linear shrink gives you a linear improvement in clock rate and a quadratic increase in the number of transistors."* (John Hennessy 1999 [1]).

These improvements have come about because of the industry's ability to exponentially shrink the geometrical feature sizes occurring in integrated circuits. The primary trend and a necessary condition for the other improvements, is the integration level expressed as Moore's Law, stating that the number of functionalities per chip and that CPU performance (MIPS) double every 18-24 months. This is an observation/prediction made by Gordon Moore in 1965 prior to co-founding Intel, and it remains true up to the present time (2002) and probably at least a decade into the future as predicted by **ITRS**, (*The International Technology Roadmap for Semiconductors*, <http://public.itrs.net>).

### 3 Bottlenecks

At some point the laws of physics will stop Moore's Laws, and product developers will have to turn to other technologies to increase size and performance. But there is a general consensus that this is at least 10 years away so at the present time Moore's law generously offers very wide limits on design spaces from a "real estate" viewpoint. However, it is recognized that owing to the difficulty of forever scaling conventional planar CMOS technology new and novel devices will be needed eventually.

Implementation of non-CMOS device structures, including interconnect and memory, will drive crucial changes in process, materials, physics and design. Nevertheless, at the present time, problems of scaling do not present the major bottleneck in "real estate". There are gaps opening up between the potential technological capacity as predicted by Moore's law and the actual exploitation of this capacity as regards system performance and efficacy as well as productivity of system developers. The currently most severe limiting factors on the above mentioned design space are physical as well as methodological and may be listed as follows:

**Memory performance** - Memory performance now severely lags behind processor performance. To bridge this gap there is an urgent need for new memory architectures and usage strategies.

**Power dissipation** - Another increasingly aggravating limitation is constituted by the rapid build up of power dissipation attendant to current software and hardware architectures and design practices. At the time of writing (2002) there seems to be a consensus within the embedded systems community that power efficiency (as measured in giga operations per watt, say) needs to be improved by **three orders of magnitude** in the next few years. The improvement will be required for actual product complexity and performance of post-PC silicon systems to track feature size as line width approaches 0.07 microns, possibly as soon as in 2006. This, in turn, calls for novel computing platforms, based on increased parallelism (allowing slow clock; just enough, just in time voltage), distributed hierarchical storage and energy awareness of compilers and operating systems. Prospects of improved data transfer and storage schemes need to be exhaustively exploration since data transfer energy significantly dominates operation energy.

**Productivity**- Other issues than technical ones also need to be dealt with. The mounting complexity is accompanied by a serious increase in development-time. Moreover, processor and SOC developers are reporting that verification efforts are exceeding 70% of the total design effort, and this figure concerns hardware verification as an isolated issue. Emerging necessity of efficient co-verification with software (application software, OS, and communication drivers) adds to the problem; and soaring team sizes infer increased complexity in management of staff, data and development processes.

Currently there is a serious gap opening up between growth rates of theoretical complexity and productivity. SOC products forecasted for the end of this decade may require some 1000 person years if this gap is allowed to widen according to current trends; so if one wanted to restrain a design team to about 100 persons, one would need to start today to be able to finish in time! It appears that not process technology but design productivity will limit the filling of the fab lines unless productivity is significantly improved.

It seems safe to predict that new design metrics must include not only requirements as to architectural innovation, but also take into account a need for new co-development methods, skills, tools, research and business models. The technological advance certainly opens up vast possibilities for innovation, but only provided that the attendant complexity can be managed. It should be kept in mind that if product complexity is measured on a linear scale, Moore's law leaves us exponentially shorter time in which to develop such products if they are to remain quality products fully exploiting Moore's law. *ITRS* states that to avoid exponentially increasing development costs, overall productivity of designed functions must itself improve exponentially on the time axis. Given the importance of embedded systems, it is important to overcome bottlenecks represented by current development practices in the face of increasing complexity [3]:

*1-Current methods for designing embedded systems specify and design hardware and software separately;*

*2-Marketing groups determine system requirements and pass them to system architects;*

*3-External system requirements become translated into imprecise specifications in natural language, and these non-formal specifications are handed over to separate HW and SW groups who work independently from each other;*

*4-Hardware-software partitions are decided a priori and are adhered to as much as possible to avoid extensive redesign;*

*5-Processors and IP cores choices are based on prior knowledge. HW and SW are integrated late in the design cycle, typically by patching SW to make up for HW deficiencies. Designers often strive to make everything fit in software, and off-load only some parts to hardware to meet timing constraints.*

*6-Lack of a well-defined design flow makes specification revision difficult, and directly impacts time-to-market. Embedded processors are typically not used to full potential.*

## 4 CoDeVer – Organization and themes

CoDeVer is a 4-year Norwegian Government (NFR) founded research project, the consortium consisting of several Research Institutes, NTNU, HiBu, and industry partners in product development and the SOC/ASIC consultancy area. The vision is: "Establish methods with tools support for abstract system level design and implementation, for real-time embedded systems". The project has been working on a web-based methodology handbook for co-design and co-verification, and has built relationships with leading research centers within this field in Europe. The handbook will be tested by Norwegian companies through a "roadshow" to be undertaken by Sintef and HiBu during September and October 2002 and results will be ready for presentation at NIK 2002.

The CoDeVer focus is on an enabling methodology that promotes a well-defined development flow based on:

*1-System level design and validation by providing executable and functional specifications that can be debugged prior to costly analysis, design and implementation;*

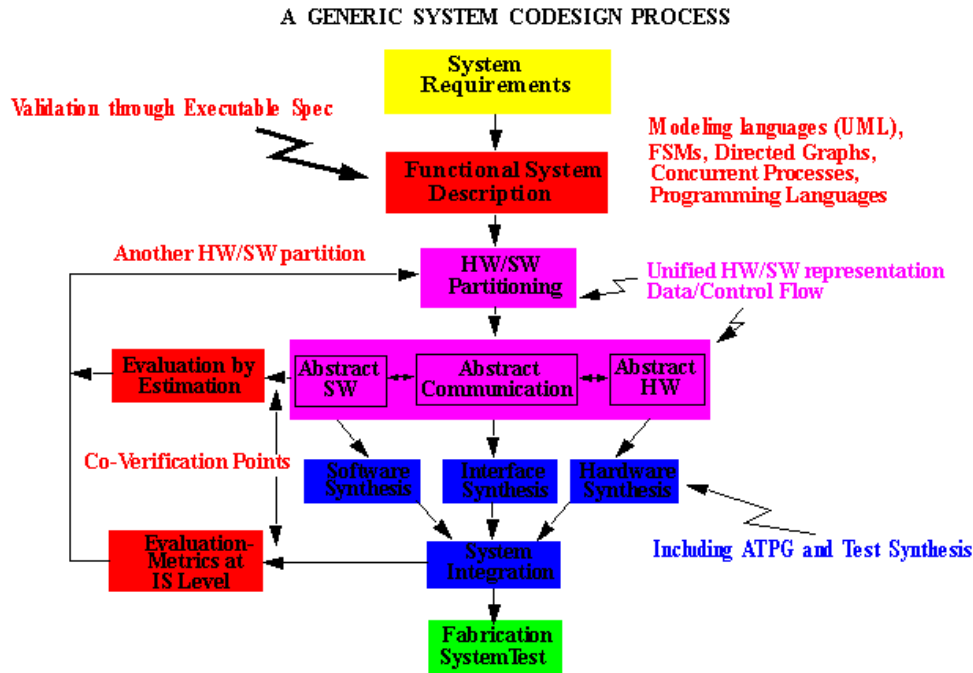
*2- HW/SW co-analysis, co-design and co-evaluation through iterative partitioning and communication selection;*

*3-Automatic interface synthesis and multilevel co-simulation;*

*4- Co-implementation by enabling SW and HW developers to work in parallel and co-verify optimized SW and HW against a common specification and within a common testbench.*

## 5 HiBu experiences

The development flow suggested to CoDeVer by HiBu is illustrated below.



and a main task for HiBu is the merger of this flow with the real-time oriented COMET process model. The COMET (Concurrent Object Modeling and architectural design mETHod) is a design method for concurrent applications - in particular, distributed and real-time applications, [4]. COMET has been derived for classical real-time systems where the separation of HW and SW modules is clear at the outset. Such an *a priori* separation is not applicable to efficient SOC realization of embedded systems where the starting point is a collection of cooperating and communicating tasks, and where the successful mapping of these tasks onto either SW or HW is a main problem. The development process described by COMET is an object-oriented, highly iterative software life cycle model based on the UML use case concept. The process is compatible with the well known Unified Software Development Process.

The HiBU/ CoDeVer report "Towards a common software/hardware development process using UML", [5], will describe our extension of the COMET method to include hardware/software design and partitioning considerations, and our findings as to where extensions of COMET are called for will be highlighted at NIK 2002. Efforts are devoted to the seamless inclusion of HW/SW partitioning into COMET. The purpose of a hardware/software partitioning step is to decide how the functionality of the system should be mapped onto hardware and software, given a set of functional and non-functional constraints.

This partitioning is often preceded by system partitioning whereby coarsely granulated systems tasks (objects, functionalities) are mapped onto separate physical system units (chips and/or boards). Subsequently each of these would evolve into collaborating tasks to be allocated and bound to hardware or software units by the hardware/software partitioning process.

Hardware/software partitioning is a difficult problem that requires good workable heuristic methods. The basis for this process may be the model descriptions from the analysis and/or design phase or other additional representations of the design. Basically, we map functions with moderate speed requirements to processors and implement these in software, while functions with fast function requirements are mapped to ASICs. Data is allocated to distinct memories. What we are looking for is a well-balanced solution from an economic and efficiency perspective. This may seem a simple task, but in real life this is quite a hard problem. Some important mapping issues are:

*1-Availability of components (HW and SW). Availability of components depends on both technical and non-technical considerations such as previous experience, vendors, availability of supporting tools such as compilers, operating systems documentation etc;*

*2-Required memory size. Memory size and strategies are often key parameters of a system; in particular since memory performance lags that of microprocessors, and memory access rate has a significant influence on power consumption;*

*3-Communication overhead. Appropriate protocols and networks/bus-types must be chosen carefully, not only to satisfy speed and cost requirements, but also to limit power dissipation;*

*4-Availability of IP functional blocks. Availability of pre-designed, pre-verified, complex functional blocks that may be easily introduced into our design is becoming more and more important;*

*5-Use of re-configurable hardware. Hardware blocks realized in terms of programmable logic, typically advanced FPGAs, are flexible, cater for speedy implementation through automated synthesis, and may be reconfigured to serve different purposes at different times. However, they are expensive and power greedy and are not suitable for high-volume, low-power products (i.e. handheld devices).*

A partitioning algorithm, if available, determines which functions are to be implemented in HW and which in SW, and a cost function is then invariably used by the algorithm to evaluate the "goodness" of a particular partition. However, the construction of a workable realistic cost function is not a trivial matter.

Several well known algorithms can be put to use ("greedy", "force directed", "simulated annealing", "evolutionary"), and the degree of automation ranges from fully automated to manual. The partitioning granularity, which refers to the size of the function units considered for allocation into either HW or SW, can be coarse-grained or fine-grained.

An early binding or a late binding strategy can be put to use. An early binding strategy commits functions to HW or SW at the beginning of the design process, with the goodness of a partition being evaluated by estimation techniques. A late binding strategy defers the partitioning decision until the goodness can be ascertained by co-evaluating a HW/SW implementation derived by synthesis and with respect to metrics at the instruction set level. Such parallel system metrics include execution time, efficiency, overhead ratio, utilization, redundancy and price. Two common approaches can be recognized, HW-oriented or SW-oriented:

*1-In a HW-oriented approach, an initial HW-dominated HW/SW partition is selected with HW-functions being gradually extracted and moved to SW;*

*2- A SW-oriented scheme, by contrast, start with an all-SW solution, then gradually extracting performance critical sections and implementing these in HW.*

In our work the SystemC based COWARE state-of-the art simulation and profiling tool is used to guide a SW-oriented scheme where an acceptable solution is sought through an iterative approach.

## References

- [1] Hennessy J: *The Future of Systems Research*, IEEE Computer, 27-32, August 1999.
- [2] De Micheli G and Sami G: *Hardware Software CO-DESIGN*, Kluwer Academic Publisher, 1996, ISBN 0-7923-3883-9
- [3] Tabbara B, Tabbara A, and Sangiovanni-Vincentelli A, *Function/Architecture Optimization and Co-Design of Embedded Systems*, Kluwer Academic Publishers, 2000, ISBN 0-7923-7985-3
- [4] Gomaa H, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison Wesley, 2000, ISBN 0-201-65793-7
- [5] Strøm T and Ra Ø, *Towards a common software/hardware development process using UML*, to be published by CoDeVer as part of the *CoDeVer Handbook*, September 2002.