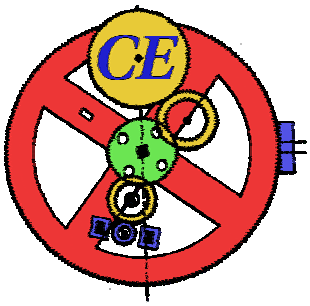

Computer Engineering Laboratory
Mekelweg 4, 2628 CD
Delft, The Netherlands
Web: http://ce.et.tudelft.nl

Embedded Processors: Characteristics and Trends

Stephan Wong, Stamatis Vassiliadis, and Sorin Cotofana

Technical Report:
CE-TR-2004-03

Abstract

In this report, we consider computational aspects of embedded systems and analyze briefly embedded processor characteristics, design styles, and project some possible design trends. We begin the presentation with a general discussion regarding embedded systems and provide a definition for such systems. Our proposed definition is intended to clarify and clearly distinguish embedded systems from application-specific systems. Consequently, we discuss various components of embedded systems and concentrate on issues related to design and development of embedded processors. We position embedded processor characteristics and consider design and implementation issues. Finally, we briefly discuss some future research directions.

TUDelft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Embedded Processors: Characteristics and Trends

Stephan Wong, Stamatis Vassiliadis, Sorin Cotofana
Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology,
Delft, The Netherlands
{Stephan, Sorin, Cotofana}@CE.ET.TUDelft.NL

Technical Report CE-TR-2004-03

Abstract

In this report, we consider computational aspects of embedded systems and analyze briefly embedded processor characteristics, design styles, and project some possible design trends. We begin the presentation with a general discussion regarding embedded systems and provide a definition for such systems. Our proposed definition is intended to clarify and clearly distinguish embedded systems from application-specific systems. Consequently, we discuss various components of embedded systems and concentrate on issues related to design and development of embedded processors. We position embedded processor characteristics and consider design and implementation issues. Finally, we briefly discuss some future research directions.

1 Introduction

The employment of embedded processors appears to grow in an exponential curve. For example, Figure 1 (from [3]) depicts marketing projections of shipments of embedded processors outpacing those of PC processors. It can be observed that in 1996, the sales of embedded processors was

just a bit more than half of the sales of PC processors. In 1997, the sales of embedded systems already equaled that of PC processors and is showing a much higher growth rate. In contrast, PC processors are only showing a gradual growth.

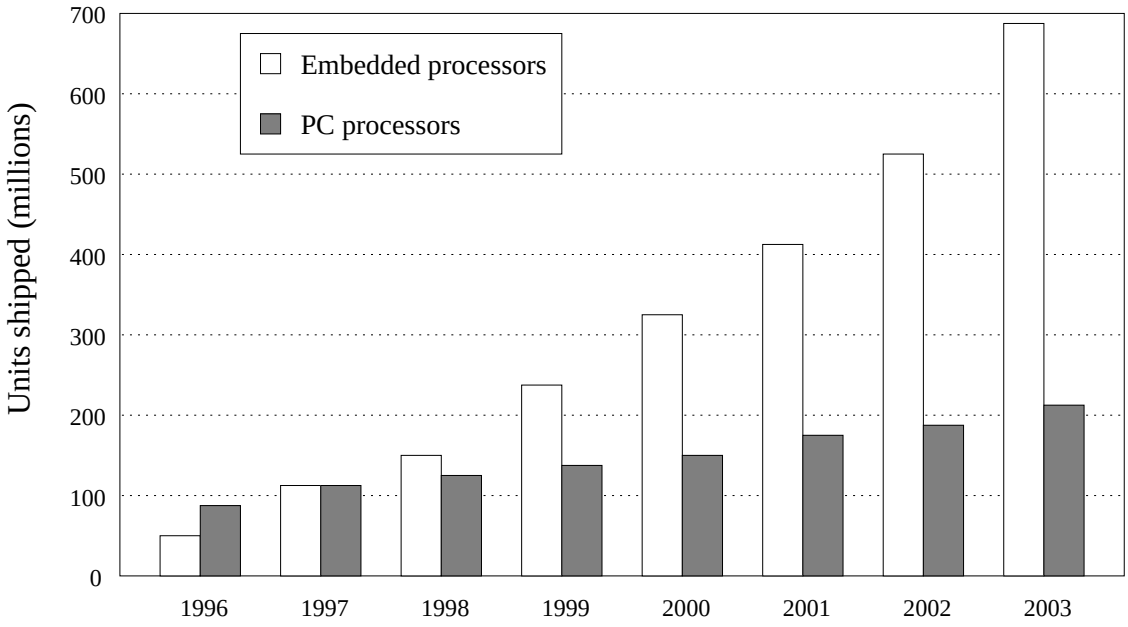

Figure 1. Marketing projections.

Some of the driving forces behind the fast expansion of the embedded processors market relates to: the proliferation of computing technologies to traditionally non-computing domains, e.g., automotive industry; the fast advances in VLSI technologies; and the tendency to replace analog signal processing with digital signal processing. While it is commonly accepted that embedded processors are playing an important part in our daily life, the term embedded processors is nebulous and to some it refers to just about everything except processors found in desktop computers, workstations, servers, and mainframes. There are clearly two questions that could be asked. Is it actually important to precisely define embedded processors and in case it is, what are actually embedded processors? The answer to the questions has a direct influence on the understanding of their characteristics and constraints which greatly influence their architecture, design, and realization and consequently in our opinion they should be addressed. Answering what embedded processors are and elaborating on their characteristics are the main focuses of this positional report. We view processors mostly from a hardware architectural point of view and therefore software and operating system issues are not addressed by this report. Additionally, to put issues into perspective, we also discuss embedded systems in general. The presentation is structured as follows. Section 2

discusses the distinction between an embedded system and systems in general. Furthermore, the section also formulate our definition of an embedded system. Section 3 explores the requirements of such systems and categorizes them. Section 4 presents a view of what we think are embedded processors and discusses embedded processor characteristics. Section 5 concludes the presentation by stating some design trends of embedded processors.

2 On the Meaning of Embedded Systems

Generally speaking, digital systems have been traditionally classified either as general-purpose or application-specific. General-purpose systems are not customized for any specific application and they are characterized by the fact that the end-user can program them to perform a broad range of different applications. Examples of general-purpose systems include: desktop computers, workstations, and server systems. In contrast, application-specific systems are designed for dedicated applications and cannot be efficiently used for other application than the one they were designed for. Such application-specific systems can be found in process control, networking and telecommunications, home appliances, consumer-electronics devices, etc.

As application-specific systems are usually integral parts of (embedded in) a larger and often non-electronic systems, there is the tendency to refer to them as embedded systems. There is obviously a conflict in terminology. Clearly, very few systems are not contained within a larger system. Furthermore, either application-specific systems are embedded or not. If all application-specific systems are embedded then one of the two terms is redundant and should not be used. In case they are not then a definition has to be found to distinguish between the two terms. In the remainder of the presentation, because of the market reality, we assume that the two terms are not synonymous and provide a position in how to distinguish between the two terms. The larger system, to which we will refer from now on in this presentation as the **embedding system**, is usually a process typically comprising both a physical system and a human operator. At its turn the embedding system is communicating with an external environment. This general embedding model is graphically depicted in Figure 2.

As it can be observed in Figure 2, the embedded system may also directly communicate with the environment and more than one embedded system can be contained by an embedding system. The key issue of the model is that the embedded system is providing a dedicated service to the embedding system. This service may be a response to external stimuli and/or data coming directly

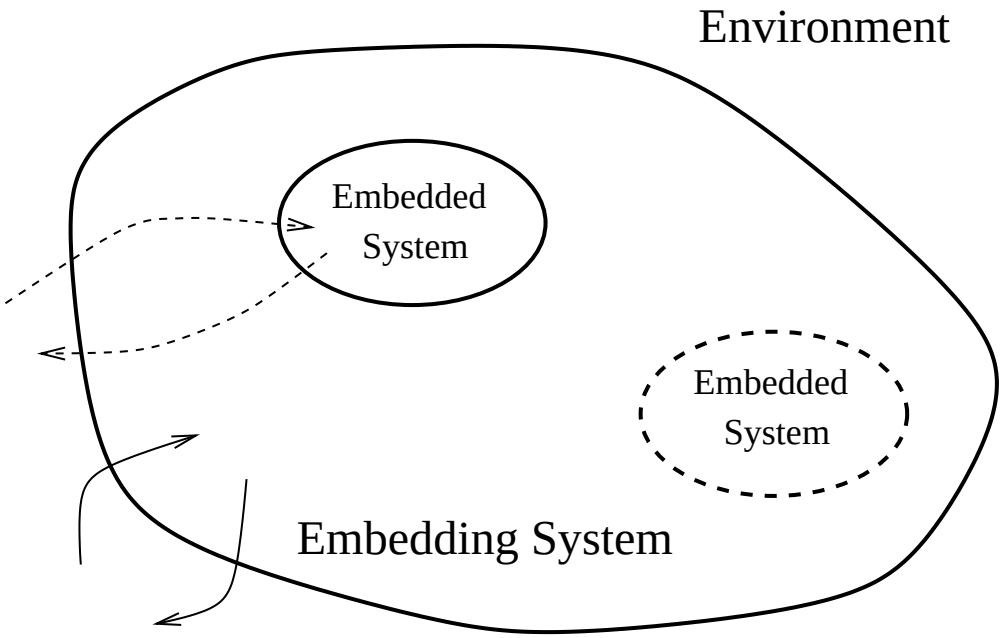

Figure 2. Embedding Model

from the environment, the embedding system, or other peer embedded system(s).

To clarify the model, we assume the example depicted in Figure 3. In this particular case, an electronic system is embedded within an external process (plant). The external process comprises a physical system and human operators performing supervising and parameter setting activities. One essential characteristic of the physical system is that it follows its own dynamics and in case of failure the previous state can not be restored. This property implies that the electronic system has to respond to certain stimuli within prescribed time constraints dictated by the external process (environment). The communication between the external process and the embedded system is performed via suitable sensors and actuators. Within this communication layer data pre- and post-processing might be also present. Due to the particularities of the plant the computational resources, the sensors and the actuators may be physically distributed. This implies that the embedded system by itself has to assume the character of a distributed system based on clustering of computational elements (processors). Within such a system the clusters communicate by message passing, rather than by shared memory, through a common communication subsystem. Within each cluster, communication via shared memory is in principle available but this does not forbid the presence of a high performance intra-cluster communication system. As a final remark on the system in Figure 3, we want to point out that such an organization is essentially a heterogeneous one: within the same cluster there may be different specialized processors, e.g., signal processors,

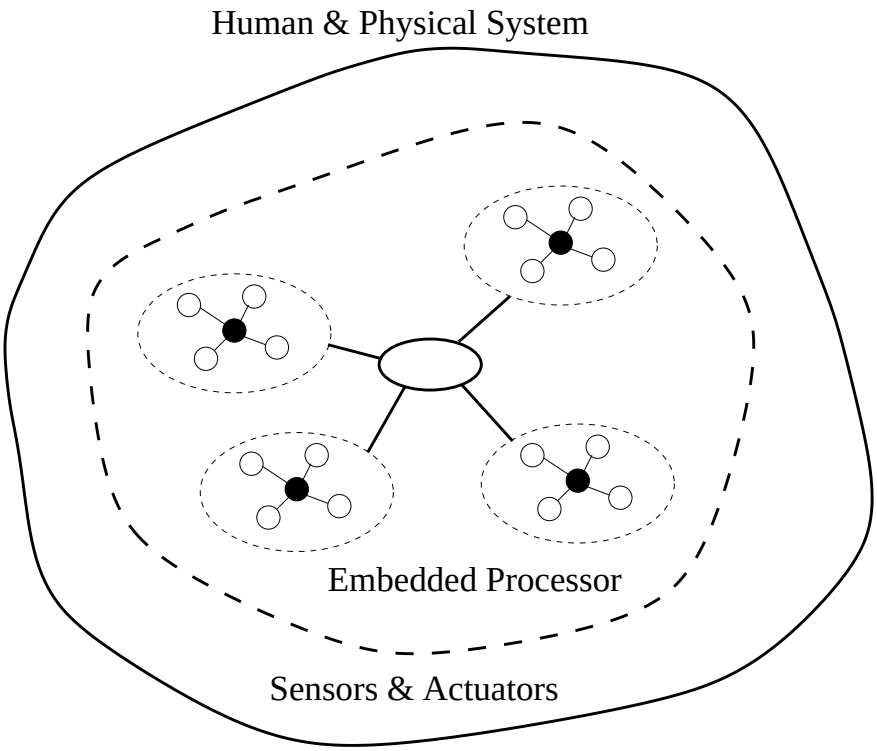

Figure 3. An Embedded System Example

communication processors, the overall set of functions performed by a cluster may be significantly different from cluster to cluster, groups of sensors and/or actuators may be connected only to a processor (cluster) or a group of processors (clusters), etc.

We note here however that even if a large number of embedded systems follow the organization depicted in Figure 3 not all of them include all the elements depicted in the figure. For example, an I/O processor embedded within a workstation can be seen as an embedded system but in this case there is no need for sensors or actuators in order to exchange information between the I/O processor (the embedded system) and the workstation (the embedding system). Consequently, we can consider the sensors-actuators layer depicted in Figure 3 as an interface layer which may or may not need to make use of such devices.

Examples of embedded systems include: fax machines, cellular phones, printers, CD players, microwave ovens, washing machines, car engine controllers, etc.. The concept of an embedded system is very broad and most readers would agree that each of the previously mentioned example performs embedded computing, but a comprehensive definition of embedded systems has not yet achieved wide acceptance. Instead of attempting a general definition, as far as we are aware of, when referring to embedded systems people prefer to give examples or define an embedded system

by describing its main properties. In the remainder of this section, we introduce a general definition of embedded systems and discuss the characteristics of embedded systems as they follow from our definition.

Definition *Embedded systems are (inexpensive) mass-produced elements of a larger system providing a dedicated, possibly time constrained, service to that system.*

Before we proceed with the presentation of the main characteristics of the embedded systems we would like to make some comments on our one sentence definition:

- The fact that the embedded system is providing a dedicated service[1] to the embedding system is actually the key characteristic of an embedded system, and theoretically speaking it should be enough to define the notion of embedded system. However, embedded systems and all the issues related to the specification and design of such systems is very much anchored into the utilization of embedded systems mostly in everyday human activities. Consequently, it is our opinion that when we refer to embedded systems as mass-produced elements we draw the separation line between application-specific systems and embedded systems.
- We included the possibility for time constrained behavior in our definition, because even if it is not characteristic to all the embedded systems it constitutes a particularity of a very large class of them, namely the real-time embedded systems.

3 Embedded System Requirements

Generally speaking, the embedded system requirements incorporating digital processing are determined by its immediate environment (the embedding system). Still, we can classify the embedded system requirements in:

- Functional requirements
- Temporal requirements
- Dependability requirements

[1] The nature of the service is not relevant in this context.

The functional requirements are defined by the operations that the embedded system need to perform which are in their turn determined by the service required by the embedding system. Later on, we will refer to these services as applications as perceived from the embedded system's point of view. The functional requirements can be further categorized into three categories:

- Data gathering is performed in order to obtain information from the embedding system or even the environment surrounding the embedding system. The actual data gathering is performed by sensors or receivers. Usually, in the case of sensors, analog data must be converted into a digital one after which a limited set of operation must be performed before storing the data. In this report, we only cover those embedded systems in the digital domain.
- Data transformation is performed on the gathered (digital data) in order to display, to send, etc., and to base decisions on the data. Operations could be varying from very simple (trivial) to complex such as encryption, compression, etc..
- Control is performed after based on the (transformed) data a decision is taken to act on the environment. After having decided to act, data must be generated (in a certain) way before it is passed to either an actuator or a transmitter.

As a final remark on functional requirements, we would like to mention that even though many embedded systems perform activities from all three categories, not all of them do so. For example, a system that is only intended for the gathering information from the environment does not need to control its immediate environment. However, all of them have to be able to perform (digital) *data processing* in order to achieve their main reason to exist, i.e., deliver the dedicated service to the embedding system. Consequently, we can claim that any embedded system has demands for large amounts of specific purpose computation and that data processing is an ubiquitous activity in embedded systems.

Apart from the functional requirements, many embedded systems also have to deal with temporal requirements induced by the fact that some tasks (parts of the dedicated service they provide to the embedding system) have deadlines. For hard real-time deadlines, the system has to deliver the service, i.e., the task has to be completed, before the deadline. All the tasks with hard real-time deadlines are critical and a failure to complete any of them before the deadline can have catastrophical results. The situation is quite different when the system is coping with soft real-time deadlines. In this case, the essential tasks, e.g., display update, have to be always completed even when they

cannot be delivered before their deadlines. All the other tasks which are non-essential, e.g., connection establishment, may be absorbed and eventually rescheduled if they cannot be completed in time.

The last class of requirements we investigate in relation with embedded systems are dependability requirements. They relate to the quality of the service the embedded system delivers to the embedding system during an extended interval of time (life time). The most important measures of dependability are:

- Reliability is the probability measure indicating the embedded systems time of being operational within a certain time span.
- Maintainability is a measure of the time to repair a system after a failure occurrence. There is a fundamental conflict between reliability and maintainability which preclude simultaneously high values for both of them. A maintainable design asks for serviceable interfaces and for the partition of the system into small replaceable units and these requirements have a bad influence on the system reliability. High maintainability is also more expensive because of the additional costs related to the implementation of serviceable interfaces. Embedded systems are usually designed for high reliability at the expense of maintainability because of safety reasons and also because being mass-produced systems they are very much sensitive to the production costs.
- Availability measures the fraction of time that the embedded system is available to provide its services to the embedding systems with respect to the total time, i.e., the time in usage added with the time to repair the embedded system.

4 Embedded Processors and their Characteristics

An example organization of an embedded system incorporating digital processing is graphically depicted in Figure 4. We included sensors and actuators in Figure 4 as potentially essential parts because a large number of practical systems designs include such components.

Before proceeding we provide a precise position on what an embedded processor is. The reason such a description is required is to differentiate embedded processors from application-specific and general-purpose processors. The definition is needed because it establishes engineering practices required to design, implement, and fabricate the various devices. The need to differentiate between

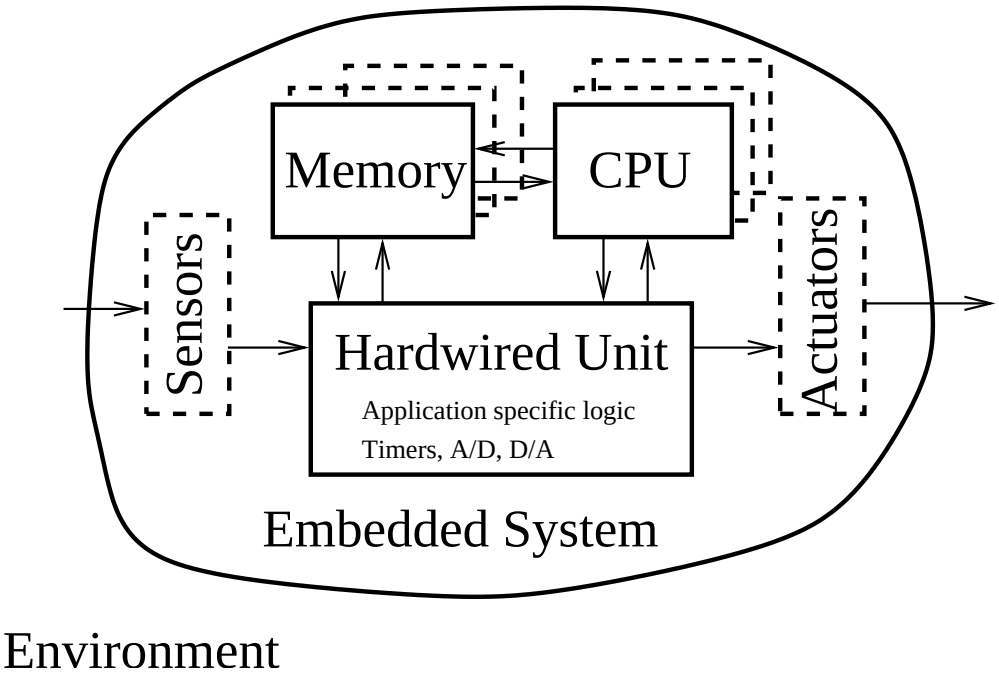

Figure 4. Essential Parts of an Processor-based Embedded System

processors used in mainframes, processors used in space equipment, and processors used in cellular phones is very obvious to anyone who have designed and developed such processors. Simply stated, a class of processors differentiate from another, because of requirements which translate to different design practices, engineering knowledge, and methods. Power dissipation, performance, size, software, reliability, etc. are viewed differently from the application-specific than the general-purpose computing point of view. For example, through history, application-specific processors are designed by extensively utilizing custom design techniques a lot more than in the design of general-purpose processors. General-purpose processors are designed with software compatibility in mind while application-specific processors are not. Application-specific processors concern only with a few applications at most, thus they are tuned towards their requirements while general-purpose processors operate on the opposite side of the spectrum. When considering for example space-related technology, even though the processing system can be designed from out of shelf components, the engineer has to know development methods and produce a design that incorporate, for example, error detection, isolation and correction, radiation protection, etc. Clearly, processors that are used in refrigerators require none of the engineering knowledge and design techniques described in the previous sentence. In almost every aspect, the categorization of processors have a great influence on the engineering practices. Embedded processors (potentially

viewed as extended microcontrollers) are part of numerous devices used by humans extensively in their everyday activities. This alone suggests that such processors cannot be expensive, they have to be mass-produced, and they have to operate mostly on room temperatures amongst other things. Their utilization and requirements alone suggests that the engineering practice and constraints are sufficiently different from application-specific and general-purpose processors. Given that the embedded processor is an embedded system per se, it can be concluded that:

Definition *An embedded processor is an (inexpensive) mass-produced processing element of a larger system providing dedicated computations and other, possibly real-time, services to that system.*

In the light of the previously stated definitions, we analyze in the following the main characteristics of embedded processors and the implications these characteristic have on the processor specification and design process. Generally speaking, the definition infers that embedded processors are: mass-produced application-specific mostly statically programmed processors.

The first and probably the most important characteristic is that embedded processors are a special kind of **application-specific** processors. Given that the description of the service to be provided by the processor (job characteristics) is a priori known the processor can be and should be optimized for an application. In other words embedded processors are definitely not general-purpose processors (traditionally). Moreover, the fact that the job characteristics are known before the hardware is designed is opening the road for **hardware/software co-design** [2] methodologies, i.e., the cooperative and concurrent design of both hardware and software components of the processor.

Another important characteristic of embedded processors is their **static structure**. When considering an embedded processor, the end user has very limited access to programming. Most of the software is provided by the processor integrator and/or application developer, is resident on ROM memories, and runs on processors which are not visible to the end user. The user can not change (reprogram) the basic actions of the processor, but it can program the processor to perform a sequence of basic actions.

Embedded processors are essentially non-homogeneous processors and this characteristic is induces by the **heterogeneous** character of the process within which the processor is embedded. A typical processor is not only mixing hardware design with software design but also mixes design styles within each of these categories. The heterogeneous character can be seen in many aspects

of the processors as follow:

- both analog and digital sub-processors may be present in the system;
- the topology of the system is rather irregular;
- the hardware may include microprocessors, micro-controllers, DSPs, ASICs, FPGAs, etc.
- the software may include various software modules as well as a multitasking real-time operating system.

Generally speaking, heterogeneity in conjunction with a large system complexity is making the design process even more complex and difficult to manage. However, one can say that heterogeneity is in the case of embedded processors design a necessary evil which provides better design flexibility.

The property which is differentiating the embedded processors from the application-specific processors is the fact that the first one are **mass-produced** elements. Because of that they are cost/performance sensitive and low cost is almost always an issue. Other consequences of the mass production characteristic are for example a high production volume, the need for a fast development, and a small time-to-market window. All of them have to be taken into account during the design process and consequently have to be embedded in the embedded processors design methodology. It also indicates low cost and extensive usage in human activities which have some interesting side effects, e.g., power constraints.

A large number of embedded processors perform **real-time** processing meaning that the jobs executed by the processor have **deadlines**. Roughly speaking the deadlines can be classified in: hard real-time deadlines and soft real-time deadlines.

A different perspective to view embedded processor is from the point of its architecture, its implementation, and its realization as described in [1].

- **Architecture** The architecture of any computer system is defined to be the conceptual structure and functional behavior as seen by its immediate user. In the case of embedded processors, the immediate user can be other embedded systems or a programmer interacting with the embedded processor itself (via programming). The conceptual structure, e.g., data structures, and functional behavior, e.g., addition, of an embedded processor is determined by the functional requirements of the embedded processor. To think that the architecture can solely be defined by the functional requirements is wrong. For example, low-power design

issues sometimes also have an impact on the definition of the architecture as (beforehand known) highly power consuming operations are to be excluded from the architecture. The trends as we envisioned them are for the definition of complex functions specialized for the application. Multimedia extension are examples of that. "Minimal" instruction set are also envisioned for low power consumption.

- **Implementation** The implementation is defined to be the logical organization of the dataflows and controls of a computer system. The implementation of an embedded processor must perform the functionality as required from the architecture level and is therefore in part determined also by the functional requirements. However, the implementation is left open for the designer in order to achieve the required level of performance as dictated partly by the temporal requirements. To this end, the designer must design new units, come up with new organizations, incorporate a certain level of parallelism, utilize microcode to control the units, etc.. More recently, the usage of Field-Programmable Gate Arrays (FPGAs) in embedded processors have allowed flexibility in the design of an implementation. As technical advances in reconfigurable hardware continue at the current pace, we expect to see more of future implementations in such reconfigurable hardware.
- **Realization** The realization is defined to be the physical structure embodying the implementation. Examples of physical structures are CMOS transistors and Field-Programmable Gate Arrays. In the realization process of an embedded processor, many issues must be taken into consideration. Power economic structures must be utilized in order to comply with low-power design requirements. High performance structures (having shorter delay paths) must be utilized in order to meet speed requirements. Furthermore, the environment in which the embedded processor is placed can also dictate what structures to use or whether protective casings are needed.

The characteristics mentioned in the first part of this section can be easily translated into these three design processes of an embedded processor. The characteristic of embedded processors being application-specific or domain-specific[2] processors is exhibited by the fact that the architecture only contains those operations that really need support from the applications set. The characteristic of static structure exhibits itself by having a fixed architecture, a fixed implementation, and a fixed realization. The heterogeneity characteristic exhibits itself by requiring an architecture that

[2] A domain relates to the a number of similar applications, e.g., media.

is more general and thus allowing a higher level of programmability, an implementation that utilizes more common blocks and thus allowing flexibility, and a realization that is easily adoptable. The mass-produced characteristic is exhibiting itself in the realization process by only utilizing proven technology that is therefore cheap and reliable. The requirement of real-time processing exhibits itself by requiring architectural support for frequently used operations, extensively parallel (if possible) implementations, and realization incorporating adequately high-speed components.

Finally, a wide variety of design issues also have their impact on the architecture, implementation, and realization of an embedded processor. However, due to the vast variety of design issues, such as cost, performance, cost/performance ratio, high/low production volume, fast development, and small time-to-market windows, we refrain ourselves from discussing these issues in the light of architecture, implementation, realization of embedded processors. However, it must be clear that each design issue has a certain level of impact on the architecture, implementation, and realization of an embedded processor.

5 Trends and Research Directions

In this section, we discuss some trends we envision in embedded processors:

- Since the introduction of microcontrollers, embedded processors have been increasingly programmable engines. While simple fixed function processors are still abundant and many specialized hardwired embedded processor units still exist, the programmable embedded processor is making headway. Advantages of programmable embedded processors are easily identified. First, the embedded system becomes more flexible, easier to upgrade and to differentiate from a previous design. Second, late specification changes can be included. Third, the re-use of previously designed software modules (functions) becomes possible assuming that they were described at a processor independent abstraction level. This paradigm shift has as a side effect the migration from assembly to high level languages, in particular C. We expect this trend to continue in the future due to two reasons. First, the industry is battling for market share in the growing embedded processors market and thus it is demanding increasingly shorter time-to-market windows. Second, the heterogeneity of many systems requires many embedded processors within one embedding system with each one having their own design cycles. This situation is increasingly becoming undesirable as these design

cycles are hard to control and to synchronize and thus delaying the completion of the overall system. Utilizing a smaller set of embedded processors performing a wider range of services (e.g., via programming) is therefore becoming more desirable.

- We also envision the increased utilization of reconfigurable hardware extending the previously mentioned need for programmability in embedded processors. Configurable hardware structures performance will likely continue to increase in the future making it an good option over fixed hardware structures. We believe that two specific issues have to be addressed in such an approach: fast reconfigurations and contained reconfiguration files. The first issue is rather obvious to improve performance. The second is the consequence of the fact that memory will be claimed by both reconfiguration and embedded software implying that special attention is required for the design of embedded memory systems. In this direction, we envision microcode related structures to re-appear possibly with different names but with the same functionality. Code compaction techniques could be again an interesting research topic. Examples of architectures supporting reconfigurable hardware can be found in [5].
- The advantages of programmability in embedded processors are evident, but the danger exists that functions performed by such embedded processors are too generic and thus decreasing performance. We envision a third trend that entails the definition of complex functions specialized for the application and their incorporation into embedded processors. Such functions, implemented in hardware or firmware, have been shown to substantially improve performance and decrease the number of executed instructions. Both features are extremely important for embedded processors. For a specific example showing this trend we refer to [9]. We have shown in [9] that it is possible to reduce the number of executed instructions in MPEG-2 by a considerable amount (see Table 1) by substituting in hardware the functions discrete cosine transform (DCT) and sum of absolute differences (SAD).

  Clearly, reducing the number of executed instructions has to do with performance gains, but it may have little to do with the reduction of code size (another important aspect of embedded processing). Most definitely, it is desirable to reduce code sizes. Our research have strongly suggested that the functions we have considered for multimedia are able to considerably reduce the code size. For example, we have observed that it is possible to reduce the code substituting high-level 'C' code for the 'sum of absolute differences' (SAD) function with assembly code in order to incorporate the SAD instruction [8] (in this case, about 30 lines of compact 'C' code was substituted by 10 assembly instructions). Of course, these are not

total number of		difference
$instructions$		
default	137500555	0.0%
DCT	118757904	-13.63%
SAD	66016988	-51.99%
DCT & SAD	46291842	-66.33%

Table 1. Number of executed instructions in MPEG-2.

mature results as they are not addressing complete applications but they indicate that there is a promise to code reduction in embedded processing.

- Regarding the embedded systems design methodology, we can claim that even if considerable research effort has been put in this direction the majority of the nowadays embedded systems are designed with ad hoc approaches that are heavily based on manual design and early experience with similar products. We expect this to change in the future, especially taking into account that we expect future embedded processors to be more programmable.
- Some fruitful research directions exist for embedded processors that combine in single chips: application-specific instruction set processing, parameterizable processors, and reconfigurable computing. A number of consequences can also appear from such a co-existence having substantial influence on not only the architecture design and realization but also on:
  - Retargetable compilers;
  - System modeling, synthesis, and design space exploration;
  - System certification and performance evaluation; and
  - Hardware/software co-design (Maybe the most mature embedded systems related field but still requiring effort to complete the maturing of such technology.)

For additional discussion on trends in embedded processors, we refer the interested reader to [4]. For more extensive information on design and integration of embedded processors, the interested reader is referred to [7][6].

References

[1] G. Blaauw and F. Brooks. *Computer Architecture: Concepts and Evolution.* Addison-Wesley, 1997.

[2] M. Chiodo, P. Giusto, A. Jurecska, H. Hsieh, and A. Sangiovanni-Vincentelli. Hardware-Software Codesign of Embedded Systems. *IEEE Micro*, 14(4):26–36, August 1994.

[3] J. Hennessy. The Future of Systems Research. *Computer*, 32(8):27–33, August 1999.

[4] M. Schlett. Trends in Embedded Microprocessor Design. *Computer*, 31(8):44–49, August 1998.

[5] M. Sima, S. Vassiliadis, S. Cotofana, J. van Eijndhoven, and K. Vissers. Field-Programmable Custom Computing Machines. A Taxonomy. *Proceedings of the Conference on Field-Programmable Logic and Applications (FPL2002)*, pages 79–88, 2002.

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

[7] J. Valvano. *Embedded Microcomputer Systems*. Brooks/Cole, 2000.

[8] S. Vassiliadis, E. Hakkennes, S. Wong, and G. Pechanek. The Sum-Absolute-Difference Motion Estimation Accelerator. *Proceedings of the 24th Euromicro Conference*, 1998.

[9] S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN $\rho\mu$-coded Processor. *Proceedings of the Conference on Field Programmable Logic 2001 (FPL2001)*, pages 275–285, 2001.