Integrated Design of Automotive Control Systems

Uwe Kiencke, Timo Kytölä, Karl Joachim Neumann

University of Karlsruhe
Institute for Industrial Information Systems
Hertzstr. 16, 76187 Karlsruhe
Fax: 0049 - 721 - 755788

Abstract

The increasing complexity and the distributed architecture of automotive control systems, in combination with an economic pressure for shortened design periods, is a challenge for electronic engineers. In this paper, an integrated design approach is presented. For enhanced efficiency, a hierarchical approach is proposed, which is based on a uniform scheme to assign functional units to hardware units in different levels. The different design levels consider network, ECU (electronic control unit) and μC peripherals. Network design is simplified by the emerging OSEK/VDX standard. By an equivalent standardisation within the ECUs, functional models may be automatically assigned to software and hardware-objects.

1 Introduction

Electronic technology is applied in automotive systems as a basis for the implementation of new functional features. The application of electronics has provided higher performance, more comfort, a higher safety level and less exhaust emission for today's automobiles.

The cheap application of electronics depends on the fast progress of semiconductor technology. This means faster cycle times of logic components, growing complexity of integration and falling component prices.

New functions are still generated for automobiles, a process, which will probably continue during the next years. New control functions are needed to meet the ever tightening regulations for exhaust emission, the growing demands for comfort, the introduction of enhanced driver information and traffic control systems [1]. To be able to implement such new features, enhanced electronic architectures have to be applied.

There are two alternatives approaches to enter new control functions into an automotive electronic system: distributed or locally integrated (Figure 1). The distributed approach emerged with the first in-vehicle networks in the '80s. By this approach, different electronic control units are connected by a communication link through equivalent network interfaces [2]. In addition, autonomous intelligent sensors [3] and actuators can be connected to other units through the network. Networking is the basis for new top-down control approaches, independent of local ECU platforms. Examples are traction control, vehicle dynamic control and electronic torque control.

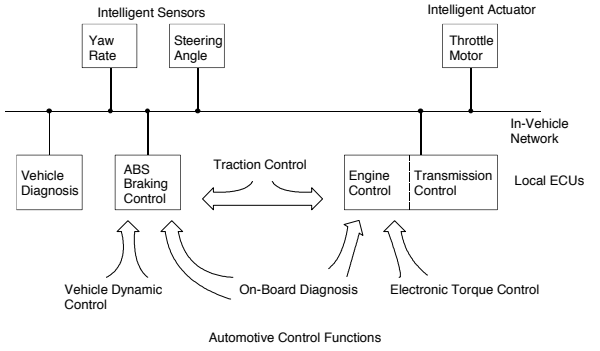

Fig. 1: Distributed architecture of automotive electronics

In some cases, it turns out to be more cost effective integrating separate control functions in one local unit. Such a local integration is especially attractive, when fast and frequent interactions between several control functions are needed. An example of this approach is the integration of engine and transmission control into one ECU.

Another way to implement new additional features like on-board diagnosis, which very heavily depends on remote sensor data and pre-processed information from other ECUs, is the introduction of a separate ECU.

This is then the platform for integrated vehicle models and control schemes exclusively determined by their physical behaviour, rather than by their random local assignment to ECUs. Distributed microelectronics gives control engineers the chance to take over the lead in the development of innovative automotive systems.

The implementation of new control functions in automobiles requires also a significant growth in μC performance. A way to boost μC performance is to increase parallelism within the Hardware by embedding enhanced peripherals and co-processors on the chip. This approach is a current trend in automotive electronics. Figure 2 shows an eventual future μC architecture containing enhanced peripheral units on the chip. Most of such peripherals already exist today, but are not yet used in combination for an integrated control approach.

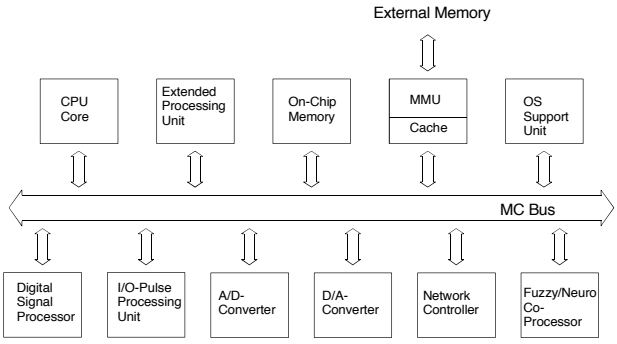

Fig. 2: Enhanced μC structure

The limits for the application of electronics in automobiles will not only be determined by the progress of the semiconductor technology, but also by the ability of designers to handle the growing complexity of distributed automotive electronic systems. The application of complex electronics also requires an enhancement of the design methods in order to shorten development cycles and to ensure the design quality.

2 Integrated System Design

Figure 3 illustrates the design problem of automotive electronics. On the network level, the system designer has to decide, which functions are to be implemented on which network node, or ECU. In addition to the function assignment for nodes, the communication bus has to be dimensioned in a way, that application specific response times for the communication are met. The designer has also to consider hardware cost requirements, which limit the implementation possibilities.

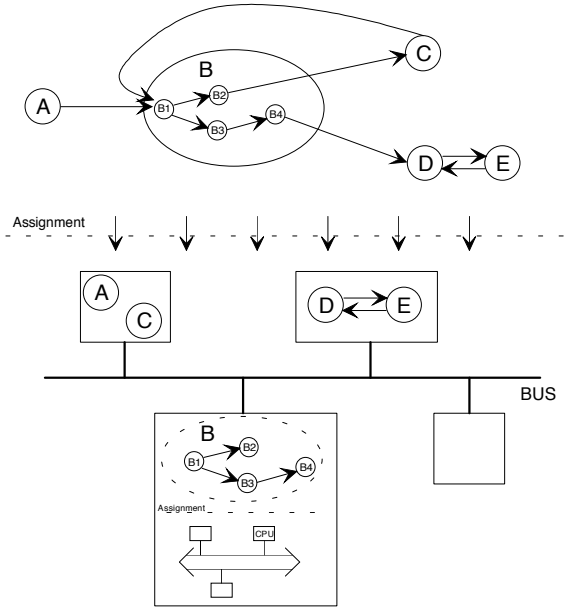

Fig. 3: Hierarchical design problem

The designer is confronted with a similar problem, when a hardware platform should be designed for an ECU. Typically, an ECU hardware consists of a μC, which has different peripherals assisting the main CPU (see Figure 2). The designer has to decide, which special tasks should be implemented on peripherals to fulfil application specific performance requirements and cost limitations.

The main factors influencing the design in each level are summarised in Figure 4.

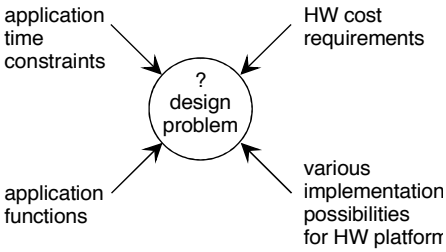

Fig. 4: Factors influencing the design of electronic control systems

This rather complex cost-performance-problem makes it necessary to develop new integrated methods, which allow the automation of the design process. In an integrated approach, the functions of an automotive control system are first described using a suitable modelling method, but without considering any hardware solutions. After this, functional tasks are assigned to system components in a hierarchical manner. On the highest level, the functional system model is partitioned and the partitions are assigned to network nodes, or ECUs. On the next design level, a hardware platform is designed for each ECU to implement the functionality, which was assigned for that ECU by the partitioning on the network level.

Figure 5 shows a basic scheme for a design process handling a partitioning problem [4]. The included design steps are introduced in the following sub-sections. This basic scheme can be applied for each design level illustrated in Figure 3. Sections “Network Design“ and “ECU Design“ describe more detailed these two design levels, i.e. network and ECU design.

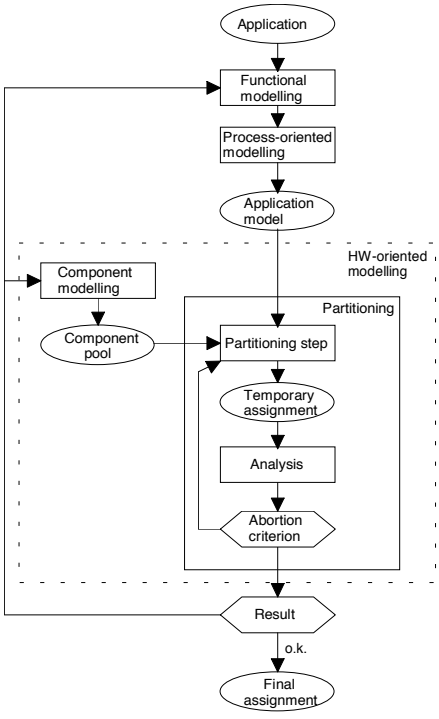

Fig. 5: System design with a partitioning problem

2.1 Functional modelling

In this step, the behaviour of the application is described with an appropriate granularity using a suitable modelling method. In addition, eventual arrangements for fault tolerance are embedded into the functional model. The step should also contain a functional verification, which includes e.g. a deadlock investigation.

Beyond application functions, also the characteristic of activating events and application specific time constraints are included in the model. As an example, an activating event may have a certain repetition characteristic, such as:

- periodic
  The succeeding activating events appear after a constant period.
- quasi-periodic
  Only a minimal time interval between any two events is known.
- aperiodic
  No exact quantities (like in the above cases) are known.

For tasks, which are triggered by events, a certain execution response characteristic is required. The required response characteristics are described as time constraints. Different cases can be defined by distinguishing, how to deal with an application specific time limit for a task:

- hard
  An exceeding of the time limit is not allowed.
- medium
  Exceeding of the time limits for a task are allowed for a certain portion of all activations.
- soft
  No time limit exists. The task is executed as soon as possible without violating the response characteristics of other more critical tasks. Average execution statistic can be derived.

2.2 Process-oriented modelling

A process-oriented model describes the interaction between the processes (i.e. tasks) in a system (see Figure 6). The process-oriented model can be derived from the functional model by grouping functions into processes, which represent units, that can be executed in parallel.

2.3 Hardware-oriented modelling

This design phase includes the performance modelling of different hardware components, which form a pool of available resources, and a functional partitioning in process-level. Functional partitioning divides the various system processes into groups and assigns each group to a resource from the pool. The partitioning is made in accordance to the performance goals and cost requirements. Figure 6 shows an assignment case using ECU functions as an example.

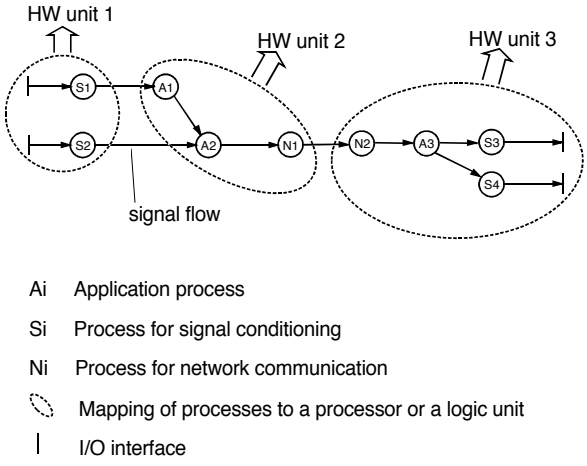

Fig. 6: Hardware-software partitioning

The partitioning is performed in an iterative manner, as illustrated in Figure 5. A partitioning algorithm tries to reach the design goals through small modifications of the previous temporary assignment. The partitioning is assisted by an analysis, which is responsible for examining, if the time constraints and cost requirements are fulfilled.

If the result of the partitioning does not meet the design goals, another iteration cycle of the system design must be made by returning to one of the previous modelling steps. Otherwise, the system design can proceed to the next more detailed design level.

2.4 Analysis

The temporary assignment can be analysed in two different ways. The *performance analysis* exams, if the application specific time constraints are met. The analysis needs the following input information for each process, which is derived from the previous modelling phases:

- execution form: parallel or quasi-parallel
- execution time using the chosen active resource (in case of a software implementation, a 100% availability of the resource is assumed for the execution time estimation)
- time limit (if exist)
- repetition characteristic of the activation signal
- response characteristic (including the percentage portion for the allowed exceedings of the time limit by a medium response characteristic)

For processes having a hard response characteristic, a 100% guarantee for meeting the time limits is needed. This leads to a worst-case arithmetic analysis. In case of a quasi-parallel execution of a process group, the chosen scheduling algorithm has a central role by the determination of the worst-case situation.

For processes with a medium response characteristic, exceedings of the time limit for a task are tolerated for a certain portion of all activations. This performance requirement is examined by a statistical analysis. E.g. the simulation can be

performed upon the basis of an enhanced Petri-Net, which is derived from the process-oriented model. In case of a statistical performance analysis, a probability distribution (e.g. Poisson) has to be additionally defined for quasi-periodic and aperiodic task activation signals.

Obviously, both arts of the performance analysis are needed for a system including processes with hard and medium response characteristics.

In addition to the functional and performance aspects, the cost requirements can be examined by a *cost analysis*.

3 Network Design

During the network design phase the application functions are assigned to the nodes of the target system. This assignment must consider the requirements of the application, especially the time constraints. The network design results in a process-oriented partitioning of the application, i.e. each ECU is assigned a set of processes to be executed on it. In addition a high level specification of the network is determined, comprising the network structure and the performance requirements for the components of the network.

A network consists of a set of nodes (ECUs) and a communication system connecting the entire nodes as depicted in Figure 1. On the level of network design, each ECU is assumed as consisting of a CPU with main memory (RAM and ROM), a communication interface, and some sensors and actuators. In comparison with the ECU design, described in the next section, this view of an ECU abstracts from the manifold components an ECU may consist of. The communication system normally consists of a single bus, but nevertheless it also may consist of several bus systems connected via gateways. The component modelling therefore must enable the network designer to describe arbitrary kinds of network structures and the performance characteristics of the network components.

As mentioned in section “Integrated System Design“, processes are the units of assignment, whereby the processes are derived from the verified functional model by grouping of functions. The functional model normally is of a very low granularity, and therefore a lot of dependencies and communication links exist between the units of the functional model. With regard to the network design the grouping of functions to processes should minimise these dependencies and communication links and lead to processes that can be executed in parallel. The resulting process graph contains:

- precedence relation between processes, i.e. requirements on the order of execution
- time constraints of processes, e.g. deadlines, periods, etc.
- communication relationship between processes,
- requirements for assignment, i.e. some processes may only be assigned to special nodes, e.g. dependence to a specific sensor that is not available at all nodes.

Another approach to get a process-oriented model is the use of a pool of standard processes or objects for the application modelling. Advantages of such a pool are:

- reusability of verified processes
- the tedious task of derivation of processes from the functional model can be omitted

Input of the partitioning algorithm is the process graph of the application and a network model. After the first iteration also the result of the analysis of the previous temporary assignment can be used for further assignments. The partitioning algorithm has to determine an assignment of the processes to ECUs by use of these inputs and with respect to the following constraints/ requirements:

- all requirements/constraints specified in the process graph, e.g. time constraints, precedences, etc.
- the execution of a process may depend on the node it is assigned
- the communication via the network takes time

The main target or constraint of the partitioning is to find an assignment that meets all time constraints. Moreover, the partitioning algorithm may have additional targets like e.g. minimisation of wiring costs, number of nodes, or bus baud rate.

During partitioning a process may be assigned to different nodes. This portation of processes requires more or less expensive adaptations of the process implementations. The costs for such adaptations may significantly reduced, if the application software is based on standardised interfaces for communication, operating system and network management, as introduced by the Franco-German co-operation OSEK/VDX (abbreviation of Open Sytems and their Corresponding Interfaces for Automotive Electronics/Vehicle Distributed eXecutive).

Subsequent to each partitioning step the analysis checks whether the determined temporary assignment fulfils the given constraints. The analysis of the overall system, not seldom comprising hundreds of processes, tends to be extremely complex. Also a simulation of the overall system may be very time expensive. In order to reduce the complexity of analysis and the time demand for simulation, it therefore seems to be beneficial to decompose the analysis into two stages:

- Analysis of the communication system
  Within this stage it only is checked whether the communication system is capable to overcome the message transfer with respect to the given time constraints. The processes are only considered as far as they have an impact to the network communication. For each node it is assumed that all processes assigned to this node fulfil their time constraints. Obviously the dependence of messages on other messages or

on application data must be considered. As additional results the communication load and its variation in dependence on the portation of processes may be determined.

- Node analysis
  The node analysis checks separately for each node the capability to manage the assigned processes with respect to the given time constraints. The processes assigned to other nodes are only considered if they communicate with processes of the node in question. Within the analysis the messages of such processes are generated, assuming that the communication system and the affected processes fulfil their time constraints.

Beyond these two analysis stages the analysis of dedicated function paths may be useful, especially if a function path consists of processes which are assigned to different nodes.

At the end of this section an additional aspect of the development of automotive systems should be outlined. The aim of an integrated development process is shown in Figure 7. The commercial contract between a car manufacturer and a supplier usually contains a functional specification which is recursively more detailed, until it describes all requirements of the contracted subsystem. The MSR project (Messen, Steuern, Regeln, i.e. Measuring, Regulating, Controlling), another multi company project in the German Automotive industry, supports this co-operation between car manufacturer and supplier [6, 7]. The aim here is an enhanced development efficiency by an improved information exchange between car manufacturers and their suppliers, on the basis of a common procedural model and uniform interfaces and tools. The MSR framework allows for the simulation of functions in advance, so that the subsequent development work necessitates significantly fewer modifications of target functions. The final MSR specification is done on the basis of the OSEK/VDX interface. From there, functional prototypes can be built and tested in vehicles. The suppliers use the same basis to develop their electronic control units for production purposes. Today's extremely costly and error-prone barrier between prototypes and final products is thus overcome.

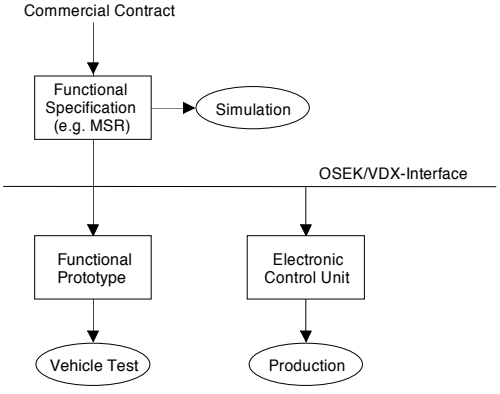

Fig. 7: Integrated development process

When several different suppliers must co-operate to integrate their individual subsystems into a complete system, the combination of the MSR and OSEK/VDX approaches is even more beneficial (Figure 8). Since the OSEK/VDX interface can integrate end products together with prototypes, different time scales or eventual time delays of one supplier no longer impede vehicle tests of the complete system at the car manufacturer.

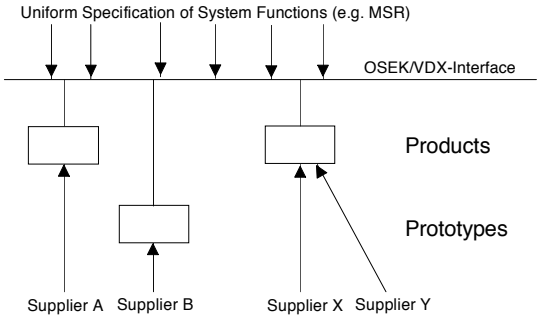

Fig. 8: Integration of automotive systems

4 ECU Design

The system tasks to be implemented on an ECU are determined by the network design, as explained in section “Network Design“. In addition to the control tasks, an ECU contains typically tasks for network communication and signal conditioning, among others. On this design level, a hardware platform is designed for each ECU to implement the assigned functionality.

An ECU hardware consists typically of a μC, which includes different peripherals assisting the main CPU. The basic μC structure was already illustrated in Figure 2. During the design in this level, the μC architecture consisting of an appropriate CPU and a set of peripherals is determined for an ECU. The design process also delivers an assignment of the different system tasks to the chosen hardware units. The design process has to ensure the fulfilment of application specific time and cost constraints. This first phase of the ECU design is called system design.

The result of the system design is a high-level specification describing the μC architecture, the required performance for each hardware unit and the functions to be implemented on each unit. The system design is followed by a detailed software design concerning the implementation of the functions on the CPU and eventually on a peripheral unit, if it consists of a co-processor. If the determined μC architecture does not exist completely, also a further detailed hardware design has to take place in addition to the software design.

Figure 9 shows the different phases of the complete ECU design. This integrated approach is called hardware-software co-design [8-12]. After the system partitioning has been done during the system design step, the actual hardware and software design can proceed separately in a conventional manner.

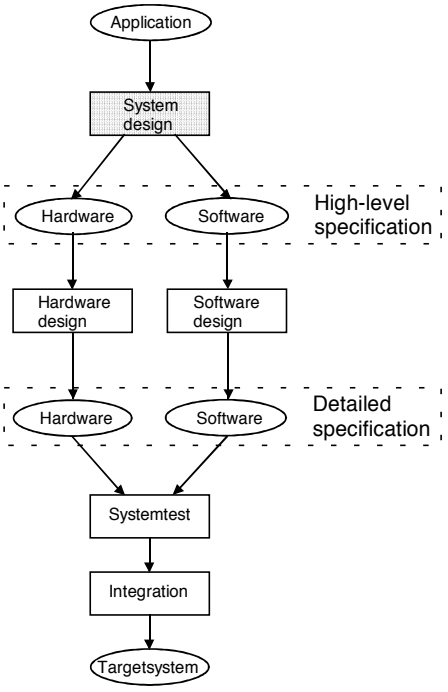

Fig. 9: Hardware-software co-design

Hardware-software co-design may be applied either by a semiconductor supplier or a customer to determine the μC architecture for a specific application or a market segment of applications.

The system design is performed according to the scheme introduced in section "Integrated System Design". The design process has to additionally consider OS (Operating System) functions, which are needed, when a group of processes share a CPU. An OS takes care e.g. for interaction between processes and for serialisation of the execution, i.e. scheduling. The needed OS features can be, if necessary, explicitly modelled by the designer or derived according to other aspects modelled by the designer and using a standard set of available OS functions. As an example, process priorities for a static-priority scheduling could be derived according to the available repetition characteristics of the activating events and the required response characteristics of the processes. A standard set of OS functions is made available by the OSEK/VDX project [5, 6].

Functional partitioning divides the various system functions into groups and assigns each group to a system component [10]. A process-level partitioning approach was already shown in <<Figure 6>>. The partitioning can be also done using lower granularity, i.e. instead of processes, smaller functional units are treated as indivisible. As an example, a statement-level partitioning would group statements together and assign the groups to available components.

The hierarchical approach introduced by network and ECU design can also be applied in further design levels to derive peripheral units for μCs supporting specific functional areas. The next section shows as an example a design method of signal conditioning peripherals. This method can be combined with the ECU design in a hierarchical manner. The design on ECU-level would determine, which signal conditioning functions are to be implemented on peripherals, i.e. not necessarily the complete signal conditioning should be ported into a peripheral unit. The design on ECU-level would also deliver application specific time constraints for the execution of the peripheral signal conditioning. The time constraints are then used as requirements, which guide the design of the signal conditioning peripherals.

4.1 Design of Signal Conditioning Peripherals

We are currently working on a hardware-software co-design method, which determines a peripheral hardware platform for the signal conditioning of a real-time control system. Signal conditioning transforms the input information received from the controlled technical process into a presentation required by the application. Input signal conditioning includes tasks like sampling, scaling, normalising, filtering, monitoring etc. Concerning the output to the technical process, the signal conditioning offers a higher level functional interface for the application. The output signal conditioning takes primarily care for an autonomous generation of output signal sequences of different kinds.

Signal conditioning includes often rather complex and extensive autonomous tasks, which may concern both input and output interfaces of the control unit. Such examples in automotive control are ignition and fuel injection, where output pulses are generated relative to the position of the crankshaft. By signal conditioning, very short response times are often required. As a trend to reduce the load of the application processor, signal conditioning is ported to a peripheral unit. This relieves the application processor from the most time critical tasks and makes it easier to design the application in a predictable manner. Figure 10 shows the structure of a control system, where the peripheral unit consists of a co-processor and an ASIC portion.

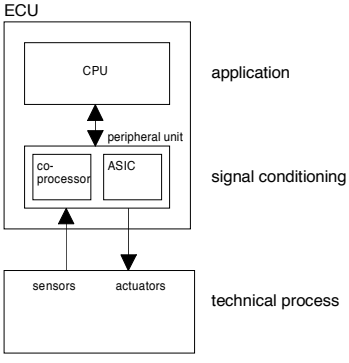

Fig. 10: System structure for peripheral signal conditioning

In our method, the peripheral hardware platform may consist of a co-processor and logic units, which represent an ASIC portion. A co-processor or an ASIC can also alone build the peripheral unit. The assignment of functional units to components has to consider application specific time constraints for the execution of the signal conditioning tasks.

The peripheral hardware platform is designed in a manner, that the time constraints are fulfilled under every circumstances. This leads to a worst-case performance analysis. Additionally, the assignment process should minimise the costs for the peripheral platform, when choosing components from an available component pool.

An object-oriented modelling is used in the hardware-software co-design method to describe the signal conditioning functions. The signal conditioning for a certain application is modelled connecting objects, each of which implements its own specific signal conditioning task. Figure 11 shows a small signal conditioning example for a square wave signal.

Due to the similar art of signal conditioning in different applications, a pool of reusable signal conditioning objects can be constructed to ease the modelling. Also, objects of the pool can be refined to make slightly different variations.

Object-oriented modelling is a way of thinking abstractly about a problem using real world concepts, rather than computer concepts [13]. Signal processing tasks can be modelled in a conventional manner using well-known elements like filters and timers, among others. An object combines all the data and operations, which are necessary to perform a signal conditioning task. An object-oriented communication model shows also the natural information flow between the elements. Object-oriented modelling has already been adapted in several related projects [14-16]. These projects concern code generation for signal processing in analysis and measurement applications.

In addition to the easy understandable presentation, object-oriented techniques offer further advantages. Although originally aimed for software design, object-oriented modelling can be also used for hardware design [17]. Therefore, an object-oriented modelling technique offers a uniform, implementation independent abstraction level as a framework for hardware-software co-design. Clearly defined interfaces also support design automation.

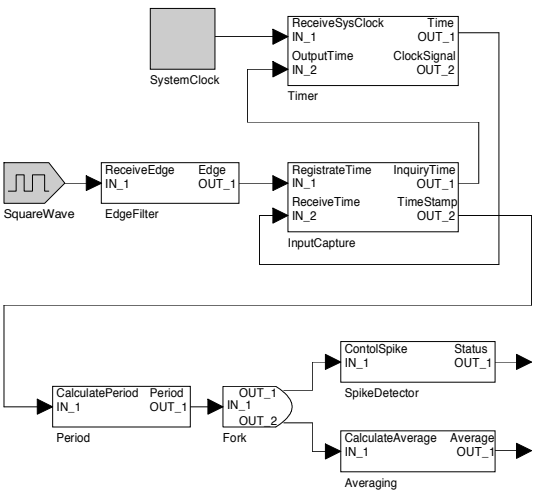

Fig. 11: An example for signal conditioning

Repetition characteristics of signals, which activate signal conditioning tasks, are described in the way shown in sub-section “Functional modelling“. Activating signals are: input signals from the controlled process containing events (i.e. interrupts and pulse sequences), clock signals for sampling purposes and commands from the application processor.

As time constraints, a required response time is assigned for each sequence of operations activated by an input signal. A required response time means a time bound, during which all operations of the corresponding sequence have to be completed after the activation. Further tighter time bounds can be optionally assigned inside an operation sequence.

The designer chooses a pool of components, which implement the modelled signal conditioning functions. The component pool, or allocation, can contain several components implementing the same function, but distinguishing concerning performance and cost.

We apply object-level partitioning, where objects represent the smallest indivisible functional units used in partitioning. Objects can be seen as autonomous functional units, which can be executed in parallel due to the object-oriented communication, i.e. message passing.

Data clustering is already used as an essential feature of the object-oriented modelling method. In data clustering, functional units using shared data are grouped together to reduce data access overhead. Through the indivisibility of an object, also the data consistency within an object can be ensured with a rather small overhead. The data consistency is ensured by performing the operations of an object in a non-pre-emptive manner.

The design method delivers as a result a specification containing the as objects modelled signal conditioning functions, the assignment of objects to peripheral components and a performance description for each component. The software for a co-processor can be generated from the specification with a rather small effort, because the functionality of each object is described in the "C"-language during the functional modelling phase. If a peripheral unit should be implemented as logic, the objects assigned to it form the functional specification for a detailed hardware design. Estimated execution times for the operations of the corresponding objects represent performance goals, which are to be met in the detailed hardware design. The estimated execution times are already required by the partitioning process.

5 Conclusion

The designer of automotive ECU is confronted with the problem, how to describe real-time function, and how to find a cost-effective software-hardware implementation that meets the real-time constraints. Starting with a functional modelling, functions are

grouped into parallel processes which are then assigned to hardware and software objects. The efficiency of such a partitioning is measured by a performance and cost analysis. The platform for such an approach is the emerging OSEK/VDX standard for distributed systems, as well as a pre-defined set of software and hardware objects for the final implementation. While the entire procedure is not yet available today, it shows a direction how to simultaneously decrease development times and product costs.

6 References

[1] J.J. Paulsen, "The state of automotive electronics in the year 2000: a perspective of the North American marketplace", IMechE, C391/KN1, 1989.

[2] H.-J. Mathony, K.-H. Kaiser, J. Unruh, "Serielle Kommunikation zwischen Steuergeräten", (German language), Proceedings of Elektronik im Kraftfahrzeugwesen, Esslingen, Germany, Expert Verlag, vol. 437, Jan. 1994.

[3] F. Heintz, E. Zabler, "Application Possibilities and Future Change of "Smart" Sensors in the Motor Vehicle", SAE Technical Paper No. 890304, International Congress and Exposition, Detroit, Michigan, 1989.

[4] U. Kiencke et al., "Architectural Trends in Automotive Electronics", IFAC Workshop on Advances in Automotive Control, Ascona, Swizerland, March 1995.

[5] Proceedings of the 1st International Workshop on Open Systems in Automotive Networks, October 9, 1995, University of Karlsruhe, ISBN 3-00-000259-6.

[6] K.G. Besel, T. Hirth, "Design systems for the MSR-Project", (German-language), VDI-Berichte Nr. 1009, pp. 503-516, 1992.

[7] J. Leohold, "The MSR-Project: Tool support for new ways of co-operation between vehicle manufacturer and supplier", (German-language), VDI-Berichte Nr. 1009, pp. 491-501, 1992.

[8] W. H. Wolf, "Hardware-Software Co-Design of Embedded Systems", Proceedings of the IEEE, vol. 82, no. 7, July 1994.

[9] E. D. Lagnese, D. E. Thomas, "Architectural Partitioning for System Level Synthesis of Integrated Circuits", IEEE Transactions on CAD, vol. 10, no. 7, July 1991, pp. 847-60.

[10] D. G. Gajski, F. Vahid, "Specification and Design of Embedded Hardware-Software Systems", IEEE Design & Test of Computers, Spring 1995, pp. 53-67.

[11] R. Ernst et al., "Hardware-Software Cosynthesis for Microcontrollers", IEEE Design & Test of Computers, Dec. 1993, pp. 64-75.

[12] R. K. Gupta, G. De Micheli, "Hardware-Software Cosynthesis for Digital Systems", IEEE Design & Test of Computers, Sept. 1993, pp. 29-41.

[13] J. Rumbaugh et al., "Object-Oriented Modelling and Design", Prentice-Hall, 1991.

[14] E. A. Lee et al., "Gabriel: A Design Environment for DSP", IEEE Trans. Acoust., Speech, Signal Processing, vol. 37, no. 11, Nov. 1989, pp. 1751-62.

[15] C. Rudolph,et al., "Objektorientierter Ansatz zur Signalverarbeitung mit rechnergestützen Meßsystemen", Technisches Messen, 58 (1991) 10, pp. 387-93 (In German).

[16] M. Karjalainen, "DSP Software Integration by Object-Oriented Programming: A Case Study of QuickSig", IEEE ASSP Magazine, Apr. 1990, pp. 21-31.

[17] S. Kumar et al., "Object-Oriented Techniques in Hardware Design", IEEE Computer, June 1994, pp. 64-70.